Composing Recommendations Using Computer Screen Images: A Deep Learning Recommender System for PC Users

by

Daniel Shapiro

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Ph.D. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Daniel Shapiro, Ottawa, Canada, 2017

Abstract

A new way to train a virtual assistant with unsupervised learning is presented in this thesis. Rather than integrating with a particular set of programs and interfaces, this new approach involves shallow integration between the virtual assistant and computer through machine vision. In effect the assistant interprets the computer screen in order to produce helpful recommendations to assist the computer user. In developing this new approach, called AVRA, the following methods are described: an unsupervised learning algorithm which enables the system to watch and learn from user behavior, a method for fast filtering of the text displayed on the computer screen, a deep learning classifier used to recognize key onscreen text in the presence of OCR translation errors, and a recommendation filtering algorithm to triage the many possible action recommendations. AVRA is compared to a similar commercial state-of-the-art system, to highlight how this work adds to the state of the art.

AVRA is a deep learning image processing and recommender system that can collaborate with the computer user to accomplish various tasks. This document presents a comprehensive overview of the development and possible applications of this novel virtual assistant technology. It detects onscreen tasks based upon the context it perceives by analyzing successive computer screen images with neural networks. AVRA is a recommender system, as it assists the user by producing action recommendations regarding onscreen tasks. In order to simplify the interaction between the user and AVRA, the system was designed to only produce action recommendations that can be accepted with a single mouse click. These action recommendations are produced without integration into each individual application executing on the computer. Furthermore, the action recommendations are personalized to the user's interests utilizing a history of the user's interaction.

Acknowledgements

I would like to thank my wife Liora and daughters Tehila, Shayna, and Gila for their strong support which propelled me to complete this work. My family and friends have been a constant source of support.

I also thank my thesis supervisor, Dr. Miodrag Bolic, for his patience and flexibility over the many years I have worked for him. I would also like to thank the Canadian federal and provincial governments for funding my research over the past many years, including NSERC, OCE, OGS, OBI, NRC-IRAP, and Mitacs.

Also many thanks to the professors who prepared this thesis template: Colin Campbell, Ray White, Stephen M. Carr and Wail Gueaieb. And thanks as well to CIFAR for funding the early work on deep learning that enabled the broad successes we see today with deep learning.

Contents

1 Introduction **1**
1.1 Research Motivation 4
1.2 Objective 7
1.3 Contributions 8
1.3.1 Thesis Questions 10
1.3.2 Thesis Statement 11
1.4 Scope of the Thesis 11
1.5 How to Read this Document 12

2 Background **14**
2.1 Neural Networks and Deep Learning 14
2.2 Quality Measurement for Systems and Classifiers 23
2.3 Recommender Systems (RS) 26
2.4 Mixed-Initiative (MI) Systems 31
2.5 Text Similarity Algorithms 37
2.6 Text Classification on OCR output 43

3 AVRA System Overview **49**
3.1 Measures of Effectiveness (MOEs) 49
3.2 Measures of Performance (MOPs) 50
3.3 AVRA System 51
3.4 Client-Server Architecture 52
3.5 User Interface 53
3.6 MI System Design 57
3.6.1 MI Design Decisions Based Upon Prior Art 57
3.6.2 User Feedback 59
3.7 Performance Evaluation 59

3.7.1 Execution Time . . . 59
3.7.2 Recall, Precision, Precision-Recall and ROC Curve . . . 67
3.7.3 Comparing Approaches with MOE and MOP . . . 72
3.8 Chapter Summary . . . 79

4 Supervised Learning Use Cases 82
4.1 Supreme Court of Canada . . . 83
4.2 Genetic Research . . . 85
4.3 Eclipse IDE . . . 87
4.4 Computer Desktop . . . 89
4.5 Console Programming . . . 90
4.6 Browsing Social Media . . . 92
4.7 Chapter Summary . . . 92

5 Context Recognition with a Convolutional Neural Network 95
5.1 Supervised Context Learning . . . 99
5.2 Precision and Recall of Context Recognition with CNN . . . 101
5.3 Chapter Summary . . . 113

6 Fast Context-specific Text Filtering 114
6.1 LazyJaroWinkler: A Fast Text Filter . . . 118
6.2 Trade-Off Between Filter Recall and Execution Time . . . 120
6.2.1 Filter Sensitivity to Hyperparameter Changes . . . 130
6.3 Chapter Summary . . . 134

7 Classification of OCR Output 137
7.1 Supervised Keyword Learning . . . 138
7.1.1 Baseline Keyword Recognition . . . 142
7.1.2 Supervised Keyword Learning with Memoization . . . 142
7.1.3 First Spell-check Approach . . . 144
7.1.4 Second Spell-check Approach . . . 144
7.1.5 Supervised Keyword Learning with Deep Learning . . . 145
7.2 Supervised Keyword Learning Accuracy . . . 153
7.2.1 Supervised Keyword Learning Results . . . 153
7.3 Chapter Summary . . . 159

8 Ranking Recommendations **160**

9 Unsupervised Learning **170**

9.1 Unsupervised Action Learning Without Context 171
9.2 Supervised Context Learning 173
9.3 Unsupervised Context Learning 175
9.4 Performance Evaluation for Unsupervised Learning 187
9.5 Chapter Summary 196

10 Epilogue **198**

10.1 Thesis Conclusion 198
10.2 Proposed Future Work 199

A Glossary of Terms **203**

Bibliography **232**

List of Tables

1.1 A non-comprehensive list of commercial virtual assistant technologies. . 5

3.1 User survey questions and responses, after AVRA demonstration 60
3.2 Theano's OpenMP benchmark timed with a vector of 200,000 elements . 62
3.3 AVRA DNN training time per epoch 62
3.4 Confusion matrix for assessing AVRA 70
3.5 Illustration of category labels for confusion matrix 73
3.6 Confusion matrix for a Google Now on Tap experiment 74
3.7 Comparing approaches with MOE and MOP 80

5.1 CNN features activated by console and/or Eclipse IDE screenshots 97
5.2 ROC curve qualitative analysis 105
5.3 Confusion matrix for assessing AVRA's CNN (K=1%) 111
5.4 Confusion matrix for assessing AVRA's CNN (K=95%) 112

6.1 Hyperparameter configurations for approaches 1-8 and 11 117
6.2 Hyperparameter configurations for approaches 9, 10, 12, 13 and 14 117
6.3 Execution time samples for specific hyperparameter configurations 127
6.4 Results counts for filter implementations 128
6.5 Average specificity, precision and recall for filter implementations 129
6.6 AVRA text filtering execution time comparison 131

7.1 Classification accuracy for the baseline OCR system 143
7.2 Examples of collisions in the dictionary keyspace 144
7.3 Classification accuracy for the dictionary-based approach 144
7.4 Classification accuracy for ASCII encoding scheme 153
7.5 Classification accuracy for various input vector encoding schemes 154
7.6 Classification accuracy for OCR approaches 158

9.1 Extending an existing context after observing the user 193
9.2 Creating a new context after observing the user 195
9.3 New keyword unsupervised learning from word embedding 197

List of Figures

1.1 An overview of the research problems associated with the thesis topic. . . 7

2.1 A typical Convolutional Neural Network (CNN) pipeline 17
2.2 Supervised learning problem illustration 20
2.3 Unsupervised learning problem illustration 20
2.4 Comparing Google Now on Tap response in-app v.s. screenshots 34
2.5 Text Classification Task. 39
2.6 Demonstrating OCR mistakes using PhotoOCR and tessetact-OCR . . . 47

3.1 AVRA System Overview. 51
3.2 AVRA's Graphical User Interface 54
3.3 Revised AVRA System Overview (DNNs removed) 64
3.4 Revised OCR capture process (image slices) 64
3.5 CNN latency as number of context increases 66
3.6 CNN latency per context 67
3.7 CNN recognizing content in side by side windows 68
3.8 AVRA recognition of occluded program windows 69
3.9 ROC curve for AVRA experiment 71
3.10 Precision-Recall curve for AVRA experiment 72
3.11 Google Now on Tap identifying multiple items 73
3.12 Original images of former U.S. presidents 74
3.13 Google Now on Tap recognition of occluded former U.S. presidents 75
3.14 ROC curve for Google Now on Tap experiment 76
3.15 Precision-Recall curve for Google Now on Tap experiment 77

4.1 Use case illustration: AVRA assisting with legal document reading 84
4.2 Use case illustration: AVRA assisting with genetics research 86
4.3 Use case illustration: AVRA assisting with a programming task 88

4.4 Use case illustration: AVRA assisting the user to start a new task 89
4.5 Use case illustration: AVRA assisting the user with console commands . 91
4.6 Use case illustration: AVRA assisting the user when browsing social media 93

5.1 Examples of console and Eclipse IDE screenshots 96
5.2 CNN style transfer examples 100
5.3 CNN example training images 103
5.4 Classification accuracy during CNN training 104
5.5 First of 4 ROC curve examples for CNN training 107
5.6 Second of 4 ROC curve examples for CNN training 108
5.7 Third of 4 ROC curve examples for CNN training 109
5.8 Fourth of 4 ROC curve examples for CNN training 110

6.1 Jaro hyperparameter impact: Objective vs. LIMIT 132
6.2 Jaro hyperparameter impact: Execution Time vs. LIMIT 133
6.3 Heat map of the correlation matrix for LazyJaroWinkler 135
6.4 t-SNE 2D view of the LazyJaroWinkler design space 136

7.1 Example of OCR correctly processing natural language 139
7.2 Example of OCR incorrectly processing an error message 139
7.3 System for recording and correcting OCR mistakes. 140
7.4 Conversion from text into input vectors using naive ASCII encoding. . . 148
7.5 Conversion from text into input vectors using binary ASCII encoding. . . 149
7.6 Conversion from text into input vectors using Morse encoding. 150
7.7 Classification accuracy for various DNN input vector encoding schemes . 155
7.8 Graphical overview of AVRA's DNN implementation 156
7.9 Classification accuracy on TESTING dataset for OCR approaches. 157

8.1 Example image for explaining how AVRA's RS works 163
8.2 Depiction of the information flow through AVRA's RS 163

9.1 Unsupervised learning with only one context 173
9.2 AVRA design overview 176
9.4 Block diagram for AVRA's unsupervised learning algorithm 182
9.3 Unsupervised learning with multiple contexts: Adding to an existing context 183
9.5 Unsupervised learning with multiple contexts: Adding to a new context . 184
9.6 Bot traversing screens to generate user activity data 188

9.7 t-SNE representation of keywords from 4 contexts in 2D space 189
9.8 t-SNE representation of images from 11 contexts (3D) 190
9.9 t-SNE representation of images from 11 contexts (2D) 191

Chapter 1

Introduction

Computers typically push the cognitive effort of solving a problem onto the user. Discovery of new tasks to perform on the computer is similarly left to the user's initiative. Even very descriptive prompts such as '`Unable to lock the administration directory. Are you root?`' typically do not recommend to the user an action with the option to accept it; for example '`To run this command as root, press Y:`'. Instead, these prompts generally require the user to tell the computer how to solve the problem based upon hints provided by the computer. This work addresses with a Recommender System (RS) this basic communication barrier between the computer and the user.

The RS in this work is called Automated Virtual Recommendation Agent (AVRA), and operates in tandem with a human user to solve a common goal, such as information retrieval. This approach works by offloading to the RS some of the cognitive pressure of understanding the problem and recommending a solution. The RS follows a series of steps in order to produce recommendations. First, it automatically detects the particular contexts appearing on the computer screen, followed by the detection of tasks within those contexts that the RS knows how to help the user with, and then it weighs these solution recommendations for the user so that they are customized to the user's past behavior, and finally the RS presents three recommendations to the user via a Graphical User Interface (GUI) containing three action triggering buttons. When the user presses one of these buttons, the action corresponding to the solution recommendation in that button is executed. These actions may be email composition, opening a browser window to a particular website, opening a document, etc.

Consider, for example, a case where the RS analyzes the computer screen and detects

that the user is doing something related to the domain of computer programming. The RS then proceeds to recognize (within the context of computer programming) the known error message `OutOfMemoryError` appearing onscreen. The RS therefore proposes to the user (through a GUI button) to launch a web page in a browser window. This web page describes the commands to execute in order to increase the memory space allocated to the program in question. The user can choose to click the button, launching the browser and opening the page, or simply ignore the recommendation.

Perhaps in the not so distant future, computers will operate as the user's "intellectual partners" to solve tasks together [267]. AVRA's design takes a Mixed-Initiative (MI) approach to human-computer interaction by applying an RS to assist the computer user on its own initiative. This approach is different from contemporary user assistance software, which typically provides a text box for the user to enter a command (e.g. Launchy [108]), voice recognition to speak commands to the computer (e.g. Google Chrome [78]), or other such user-driven computer interaction mechanisms. The difference in AVRA is that it observes the visible information on the computer screen to produce recommendations on its own.

AVRA is not tightly integrated into each specific application producing onscreen information. Rather, an image of the computer screen is analyzed in order to detect the presence of information which AVRA can use to recommend actions. This shallow integration of an RS into a personal computer system is novel. Fundamentally, computer users operate in a multitasking environment where a task identified in one program (e.g. bash) is researched in another program (e.g. chrome) and fixed in a third program (e.g. eclipse). Because of the shallow integration approach, AVRA has visibility over all programs visible to the user, and can therefore help with multitasking, whereas an isolated program restricted to text or voice commands has insufficient visibility into the workspace (i.e. the computer). Effectively, AVRA sees what the user sees.

This shallow integration approach generalizes to non-programming domains where the user researches items that appear onscreen. For example, when the user is browsing social media, the RS could detect the names of friends and recommend an action such as "Compose an e-mail to Daniel" when that friend's name appears onscreen. Another example is genetic research, where the user is reading a PDF document involving genetic research. The RS can detect this genetic research context, and propose to open a browser window to a web page detailing the relationship between a gene name recognized on the computer screen (e.g. 1-acylglycerol-3-phosphate O-acyltransferase 1) and other genes (e.g. There are 11 genes in the AGPAT/LPAAT family [80]).

It is amazing to see the fast and complex responses of machines to human generated queries. With Natural Language Understanding (NLU) these queries can even be issued as plain speech or text. Sidestepping the user having to type or speak a few words to launch an action comes at a cost. This cost includes lost screen real estate taken up by AVRA, and the annoyance of being bombarded with suggestions where the vast majority will be ignored. The benefit to consider is the lowered cognitive pressure placed on the user when interacting with the computer.

A computer's state can be defined at a high level (e.g list of executing programs) or at various lower levels (e.g. an image of the memory and hard disk, or the state of the program memory and processor registers, or simply by the image displayed on the computer screen). States capture the dynamic nature of computer components from transistors, which switch on and off, to programs which can contain many variables, each with their own state, to the aggregate state of the computer composed of the state of each program in memory. This work demonstrates that a trained classifier can associate one or more contexts to the current state of the computer by examining an image of the computer screen. Being aware of the context of onscreen information provides the RS with the ability to switch between modes of intent. For example, to activate "programming mode" the system will evaluate the computer screen, assess that the visible area of the screen implies that the user is programming, interpret the screen in that context, and then provide programming recommendations to the user based upon the detected onscreen keywords such as error messages. In another example, the system may assess that the user is reading online cartoons, and begin offering relevant entertainment content and other "consumption mode" recommendations. In a third example, the system may detect social media activity on the screen and enter into "networking mode" where it offers to email contacts seen onscreen.

AVRA provides personalized recommendations that adapt over time to the changing behavior of the user. The RS learns from the history of user interaction with the GUI to adapt recommendations to the user's needs, and maintains a traceable history for post-hoc analysis.

This proof of concept prototype has many limitations, including a strict learning algorithm, limited domain knowledge, privacy concerns over a system that collects computer screen images, a very basic user interface, and unanswered questions about commercial scalability of the technology. Also, this prototype was not validated with user testing. Rather it has been characterized quantitatively as a first step toward broader user acceptance testing.

1.1 Research Motivation

Clearly the computer has access to the information rendered on the computer monitor, as it is rendering the image. Why not integrate AVRA directly into each program running on the computer? Perhaps it is feasible to monitor all operating system events and messages with a service or daemon and learn to recommend actions based upon those messages. This approach would avoid image processing and related uncertainty. Although this is one possibility, it is very interesting for AVRA to have visual input from the computer screen rather than a data integration for several reasons. First, humans interact with computers through vision, and restricting AVRA to processing visual input is one step on the way to developing a mobile robot that can sit on the user's shoulder and look at and computer screen to then understand the content and speak recommendations. Second, integrating deeply into programs running on the computer is not easy. Applications are built using a wide variety of programming languages and compilers, hiding many key details from a generic application monitor. There is a lack of deep access into program state as programs run as stovepipe systems on top of the operating system. Maintaining interfaces into a large array of software programs on an average desktop is an integration nightmare. Even so, representing the state of every component of every program executing on a computer would be a challenge. Finally, processing images rather than creating programming interfaces exposes information that is not otherwise accessible. For example, visual processing exposes remote desktop information, the contents of images, document styles, and other information that is only apparent when analyzing image information.

This research was originally motivated by an initiative gap in the interaction between computers and computer users. Specifically, state of the art virtual assistance technologies focus on processing user-provided natural language or text input. Table 1.1 contains a non-comprehensive list of virtual assistant technologies. The goal of these systems is usually to identify the intent of the user from the user's input, transform the identified intent into a query, and then to return query results to the user. Some systems go further and attempt to support interactive dialog, reminders, and application launching. This command and dialog approach to virtual assistants puts the onus of specifying intent squarely onto the user. It is that cognitive pressure applied to user by the assistant - the need to have the user specify what the user wants - that is a limitation of existing work.

To elaborate further on this limitation in state of the art virtual assistants, consider the following motivating use case, which makes it clear that forcing the user to specify

Table 1.1: A non-comprehensive list of commercial virtual assistant technologies.

Project	Company	Voice Control	Assistant	Conversational	Image Processing
Google Now on Tap (Screen Search) [74] [18]	Google	NO	YES	NO	YES
M (beta) [193] [86] [29]	Facebook	YES	YES	YES	LIMITED [119]
Google assistant [128]	Google	YES	YES	YES	NO
Watson [97] [98]	IBM	YES	YES	YES	NO
Viv [113] [116]	Viv Labs	YES	YES	YES	NO
Hound [220]	SoundHound	YES	YES	YES	NO
Assistant.ai [20]	Api.ai	YES	YES	LIMITED	NO
Alexa [15]	Amazon	YES	YES	"Skills Kit"	NO
Google Now [73]	Google	YES	YES	NO	NO
Cortana [142]	Microsoft	YES	YES	NO	NO
Siri [21]	Apple	YES	YES	NO	NO
NextOS [166]	NextOS	YES	YES	NO	NO
Kite [215]	Kite	NO	YES	NO	NO

their intent is undesirable. A programmer using a personal computer compiles her program, and then the compiler returns with an error message. The programmer must first detect that the program did not compile correctly. Next she decides that she does not know how to resolve this onscreen error without assistance, and so she copies the error message text and pastes it into a search engine, in order to explore the Internet in search of a solution. Once an interesting page discussing a solution is found, the programmer may copy some text to the clipboard and paste it back into the integrated development environment. In the past, simply having an error message to specify to the programmer the details behind the problem was significant progress over the computer crashing with no explanation. However, contemporary computer users should expect more than just an error message. Users should expect meaningful assistance presented in the form of an action recommendation, rather than a problem report. It is the absence of this participation in problem solving on the part of the computer that motivates this work.

The research motivation driving this work is the unexplored intersection of whole-screen image processing with virtual assistant technology. Although it is not a comprehensive listing, Table 1.1 makes it clear that contemporary design efforts on assistant

technologies are focused elsewhere. While some virtual assistants can converse with users, no virtual assistants can yet see what the user sees, and then understand the computer screen in some intuitive way. Facebook M reportedly can discuss images with users as demonstrated in [119], although at this time the product is still in closed beta. The closest realization of this concept available today is Google Now on Tap, an assistant for mobile phones that takes an image of the phone screen and interprets the contents (text and images) to come up with cards (information recommendations). Employing this type of virtual assistant on a personal computer is a novel approach to helping the user with tasks. In the context of programming, the assistant can share the task of identifying the error message on the computer screen, looking up actions which may help the user with the error, and recommending only the most relevant action recommendations to the user. This approach puts little or no pressure on the mind of the user in the course of normal computer use.

Integrating whole-screen image processing with virtual assistant technology is a hard problem and an open research topic. Specifically, obtaining a deep analysis of what the computer screen really means is a very difficult problem that is not solved by this work. Instead, this work presents one specific solution for integrating whole-screen image processing with virtual assistant technology. More specifically, this work shows how such a system can be created so that others may follow the design patterns presented in this work. Even under this relaxed design constraint of integrating the domains of whole-screen image processing with virtual assistant technology, this motivating semi-autonomous virtual assistant idea is quite challenging to implement in practice. Consider as a motivating example an image capture from a computer screen showing text in a pop-up warning box in the foreground with a programming editor open in the background. Imagine writing a computer program that has only this image as an input, and must return 3 action recommendations to be presented to the user. How can one write such a program? Assisting the user in this situation is actually quite difficult. This hypothetical program (which is in fact the virtual assistant) must detect within the features of the image that the user is doing something related to computer programming, and then detect within the onscreen text the keyword representing the error message in the dialog box. Even assuming that the virtual assistant can obtain all of the onscreen text without spelling errors, and including bounding boxes for the onscreen location of the text, obtaining the context from the screen is still a big challenge. Furthermore, the program must associate the detected text with some action to recommend to the user.

Figure 1.1 presents the research problems that need to be tackled to make this desktop

assistant a reality. The four branches of research that come together in this thesis are desktop assistants, keyword recognition within noisy text, recommender systems, and unsupervised learning.

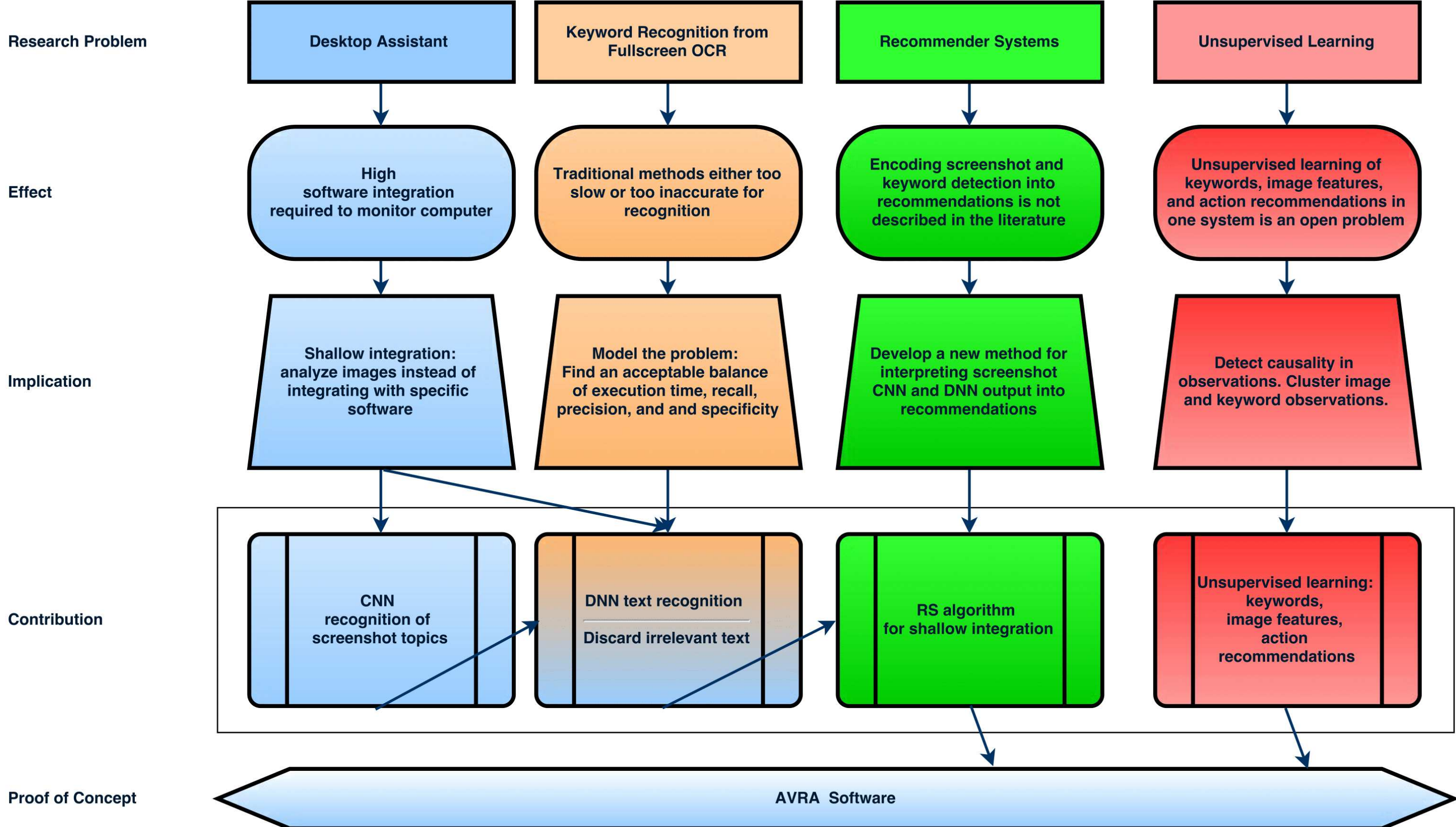


Figure 1.1: An overview of the research problems associated with the thesis topic.

1.2 Objective

The objective of this work is to characterize a proof-of-concept virtual assistant for personal computer users, assisting with onscreen tasks by interpreting an image of the computer screen and providing action recommendations. Accomplishing this objective provides new knowledge on desktop assistant technology, image and text processing, and unsupervised learning (Figure 1.1).

1.3 Contributions

The primary contribution of this work is to describe in detail a **virtual assistant that can be trained, observe the computer screen, learn action recommendations, and provide the user with timely action recommendations**. This is also the thesis objective. The key component of this contribution is unsupervised learning, discussed in Chapter 9. This new type of assistant is a high-initiative task partner for the desktop computer user which does not require tight integration into the operating system and executing programs in order to assist the user in making progress towards the completion of a task or the discovery of a new task. In order to make this primary contribution a reality, several sub-problems had to be solved, and these are also contributions of this work to the state of the art. The following contributions are related to sub-problems that were solved in completing the primary contribution of this work. Each contribution is also mapped out in Figure 1.1 to make it clear how these contributions fit into the overall thesis objective.

1. **Context Recognition with a Convolutional Neural Network (Chapter 5)**: Whole-screen image processing resolves the problem of having too many programs to integrate into when trying to understand the state of the computer. However, it introduces the problem of too few hints about the meaning of the image data. This first problem that arose in designing the system architecture for the virtual assistant is simply a lack of information about the meaning of the image. Assuming a dictionary of keywords is known to the virtual assistant, and each keyword is paired with an action recommendation, the ambiguity of the keywords is generally unacceptably high. For example, the error message "Error: NullPointerException" printed into a terminal window indicates an onscreen error message for which a solution should be recommended, while the same message presented in an online programming tutorial does not call for a recommendation to be presented to the user. In order to guide the interpretation of onscreen keywords, this work proposes a way to guide the interpretation using image features. This is called shallow application integration, because the generated recommendations can be provided without integration into the individual programs shown on the computer screen. This type of shallow integration is accomplished by processing and understanding images of the computer screen, rather than integrating into each and every program executing on the computer. Quantitatively this contribution is measured in terms of execution time, precision, recall, and hyperparameter settings. This contribution

is framed within the state of the art in Chapters 2.3 and 2.4.

2. **Fast context-specific text filtering (Chapter 6):** The contribution above is only a partial solution to the overall goal of completing the primary contribution, providing for context-specific slow extraction of text from an image. The extraction is too slow. The solution is to accelerate the text analysis with approximate string matching. Discarding text that is very highly likely to fail the classification task reduces the amount of text to be classified. A fast text similarity algorithm is described which is used to discard irrelevant candidate input text prior to context-specific classification. Quantitatively this contribution is measured in terms of execution time, precision, recall, specificity, and hyperparameter settings. This contribution is framed within the state of the art in Chapter 2.5.

3. **Deep Learning Classification of OCR Output (Chapter 7):** The two contributions above led to a system that identifies interesting text in a context-specific way. The next problem is to identify or discard the candidate onscreen keywords. Making this task even more challenging, the image to text conversion process called Optical Character Recognition (OCR) introduces spelling errors into the text. These errors are especially prevalent among long non-dictionary keywords that are essential in many applications (e.g. computer programming and genetics). The solution was to learn the mistakes made by the OCR software for the keywords of interest. When the errors are learned, 97% of the text can be detected accurately. The contribution here was to maximize keyword recognition accuracy within the OCR output. A deep learning neural network was used to learn and correct these commonly made OCR mistakes. Only when a specific topic is recognized onscreen are the keywords for that topic sought out by the virtual agent. This is a context-aware approach to text recognition. This contribution is framed within the state of the art in Chapter 2.6.

4. **Recommendation Filtering Algorithm (Chapter 8):** The three contributions above lead to a system that can provide many relevant action recommendations. However, sometimes there are too many actions to recommend and not enough space in the user interface to present them all. The system required a way to rank the available recommendations in a way that takes into account both the relevance based upon the user's behavior profile and also taking into account the classification confidence of the neural networks that the information that triggered

the recommended really was present on the computer screen. The contribution here was to find an useful way to integrate image recognition information with keyword recognition information such that recommendation quality is ranked. The solution is to apply a recommendation filtering algorithm which is a hybrid filtering approach incorporating a content-based, context-aware ranking of predictions modified by user history-based and probabilistic approaches. This contribution is framed within the state of the art in Chapter 2.3.

5. **Unsupervised Recommendation Learning (Chapter 9):** The final problem solved in this work relates to the limits of supervised training. No matter how much general knowledge is trained into the virtual assistant, it still needs to learn more in order to satisfy users. The solution was watch the user and then draw causal relationships from the user's actions. These actions could then be associated with topics using image recognition, image similarity, and semantic text similarity. The action recommendations produced by the virtual agent can therefore be personalized to the user's interests utilizing unsupervised learning. Overall the contribution was to describe how to perform unsupervised learning of new keywords, visual topics (contexts), and action recommendations. The unsupervised learning approach was developed to be compatible with supervised learning techniques. Therefore the virtual assistant can be trained on specific topics and also can learn from the user's actions. This contribution is framed within the state of the art in Chapter 2.1.

Prior to submitting this thesis, descriptions of various aspects of AVRA were submitted or accepted for publication. An overview of this work appears in [209]. Sections of Chapters 2.6, 7, 7.1, and 7.2 appear in [204]. Sections of Chapter 2.5 appear in [206], and sections of Chapter 9 appear in [208]. Part of the discussion in Chapter 4 appeared in [205].

1.3.1 Thesis Questions

Is it possible for a virtual assistant for personal computer users to interpret images of the computer screen to provide action recommendations? Could such a system learn unsupervised? How could one implement a proof of concept for this system?

1.3.2 Thesis Statement

A deep learning artificial intelligence can provide action recommendations related to onscreen messages. These action recommendations can be provided without integration into each individual program executing on the computer. These action recommendations can be provided within a reasonable response time, and can be acted upon with a single mouse click. These recommendations can be personalized to the user by utilizing the user's interaction history and unsupervised learning.

1.4 Scope of the Thesis

Included in scope of thesis are the following parameters:

- **PROTOTYPE**: Development of a proof-of-concept artificial intelligence performing the following tasks and sub-tasks.
- **DETECTION**: Context-specific detection of onscreen textual keywords in screen image captures.
- **RECOMMENDATION**: Recommendation of actions, and execution of an accepted action recommendation within a reasonable response time.
- **LEARNING**: Learning new keywords, visual contexts, and action recommendations using supervised and unsupervised learning.

Excluded from the scope of this work are the following topics:

- **USER TESTING**: As described in the research motivation, AVRA is useful as a launching pad demonstration that reveals the design patterns to follow in creating a virtual agent based upon whole-screen image processing. The point of this work is not to characterize the infinite variety of good and bad applications for which this type of system can be applied. Products or projects that are based on this work should include rigorous user testing to ensure that users are not annoyed by prompts from the system, and that the system is in fact accelerating the user's workflow.
- **USER INTERFACE**: Optimization of the graphical user interface design and related human factors engineering can be achieved following the completion of the

proof of concept prototype. In this work a 'good enough' user interface was developed to facilitate the practical implementation of the system prototype. Although the user interface is discussed in this work, it is not the focus of this work.

- **PRIVACY**: Privacy concerns related to this work are serious and numerous. Although they are not addressed as part of this work, a subset of these concerns are mentioned here. As the system in this work takes many screen captures of personal computer screens, there is a high chance of compromising data being acquired by the image recording and other components of the system. It may be necessary to limit the aggregation of personal data to avoid the bad behavior of one user being recommended to another user. Furthermore there may be commercially sensitive information captured onscreen. These concerns are very important and are left as future work.

- **COMMERCIAL CONSIDERATIONS**: Commercial considerations such as role-based access control, optimizing cloud architecture for cost and performance, database optimization for cost and performance, data backup and restore systems, and other non-academic considerations are not fully addressed in this work.

1.5 How to Read this Document

This thesis is organized as a set of standalone chapters. Each chapter begins by explaining what the chapter is about and situating the contents of the chapter into the objective of the thesis, which is to develop and characterize a proof-of-concept virtual assistant for personal computer users. Chapter 2 presents the reader with concepts, technologies and systems related to this work. AVRA's design is presented in Chapter 3. This design calls for several subsystems to be designed, which are described in detail in the following chapters. Next, a description of specific use cases is presented in Chapter 4. Chapter 5 describes the method used for recognition of features in images, and training the system to recognize these features with supervised learning. Chapter 6 compares methods for accelerating AVRA's text analysis functionality using a text similarity filter prior to text classification. Chapter 7 discusses the recognition of keywords in text extracted from images, and training the system to recognize these keywords with supervised learning. Chapter 8 presents a method for choosing which actions to recommend to the user based upon the user history and the contents displayed on the computer screen. An approach

allowing the system to learn autonomously is presented in Chapter 9. Chapter 10 contains a summary of this work and a discussion of future research directions.

Chapter 2

Background

This chapter provides the reader with some grounding in the neural network and software systems used in this work. Many of the tools and techniques leveraged in this work are quite new, and so the average practitioner may not be aware of the background information needed in order to fully understand this thesis. The prior art discussion in this work is subdivided into six topics: a general background on neural networks and deep learning in Section 2.1, quality measurement for systems and classifiers in Section 2.2, Recommender Systems (RS) in Section 2.3 related to the AVRA RS presented in Chapter 8, Mixed-Initiative (MI) Systems in Section 2.4 related to the AVRA system as a whole presented in Chapter 3, text similarity comparison algorithms in Section 2.5 related to the filtering algorithm presented in Chapter 6, and text classification in Section 2.6 related to correction of OCR output using deep learning techniques in Chapter 7.1.

2.1 Neural Networks and Deep Learning

"`The ability to predict is the essence of intelligence`" - Yann LeCun, NYU, October 2016 [118].

This section of the thesis presents a series of definitions and concepts that are used later on in the thesis. It is not a comprehensive overview of the topic. Rather, this section gives the reader enough information to understand the thesis terms and approach regarding data science, neural networks, deep learning and related fields. A comprehensive review on deep learning is [198].

A perceptron is a basic building block in neural networks, where the neuron computes an activation function that first sums each incoming signal multiplied by a weight, then

adds a bias, and finally processes this total using an activation function to see what the neuron should output [190]. Because of efficient gradient propagation through multiple layers among other advantages, the activation function is typically a rectified linear unit, or ReLU, performing $f(x) = max(0, x)$ [161]. A ReLU propagates positive results forward to the neuron's output.

An Artificial Neural Network (ANN) is a computational model of a neuron which combines many individual neurons into a network [198]. These networks can perform interesting calculations on data such as classification, memory, and computation. A Restricted Boltzmann Machine (RBM) [218] is a layer in a neural network where the neurons do not connect to each other (the restriction) but rather accept input from a previous layer (or the input to the overall network), and then feed this information forward (feed-forward).

Machine learning, first described in 1959 by Arthur Samuel, is a very broad area of research investigating how autonomous entities can learn without explicitly being programmed by humans [192]. Deep learning is a kind of machine learning applied in many fields, and so it is important to make the distinction here between Deep Learning (DL) and a Deep Neural Network (DNN). DL is an approach in artificial intelligence using many layers of neurons in a neural network to implement machine learning. The hallmarks of DL are the formation of hierarchical representations of data within the layers on the neural network, and the training of the neural network using a learning algorithm [70]. DL is a field studying neural networks that learn "to perceive / encode / predict / classify patterns or pattern sequences" [198], whereas a DNN is a particularly organized feed-forward stack of neural network layers used to assemble increasingly complex levels of feature representations [65]. Through backpropagation training (an algorithm using the chain rule for derivatives), DL ANNs can learn to model data representations at multiple levels of abstraction by adjusting the network weights through iterative steps to minimize or maximize an objective function [120].

A Deep Belief Network (DBN) is a stack of fully connected RBMs where each layer accepts pattern representations from the level below it, and learns to encode these patterns into output classes [198]. An early layer may be involved in crude tasks such as edge detection in an image input, while a middle layer among the many hidden layers may build up a representation of shapes (e.g. round, oval, square) and near the output very specific classes can be encoded into the neural network. A favourite example of this type of heirarchical representation in biological (human) neural networks is the activation of a specific neuron in the medial temporal lobe of the brain only when the subject was

shown a picture of celebrity Jennifer Aniston or in another case former U.S. president Bill Clinton [182]. In order to reach such a level of abstraction, many neural network layers are required.

Inputs to neural network (and the data structures flowing through the network) are commonly thought of as tensors. A tensor is simply a high dimensional data structure. A zero dimensional tensor is a scalar (e.g. 4). A one dimensional tensor is a vector (e.g. $[1.6, 9.7, 7.9, ...22.0]$). A 2-dimensional tensor is matrix. For example:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ & & \dots\dots\dots\dots & & \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix}$$

And an n-dimensional array is simply a tensor. The dimension count of a tensor is called the rank [264, page 35]. Consider that each tensor input to a neural network is a single point in a multidimensional vector space. A neural network learns to partition points in the space of all possible inputs into clusters using training data. A DNN partitions the vector space into classes associated to labels, and a training procedure is employed to signal to the neural network which outputs (e.g. classification = cat) correspond to which inputs (e.g. input = image of a cat). To quantize the signals from the final layer of a neural network into one single output classification, a multinomial logistic regression called softmax is used to create a probability distribution as the network output [14]. The "neural network confidence level" or "score" is simply the softmax output score for a given output class. The sum of the softmax outputs is 1.0, and so each output for a particular class to be recognized represents the confidence that the class is present in the data sent into the neural network. Argmax can be used to squash the softmax outputs (one value per output neuron), and encode the result information into a one-hot encoding. The highest valued softmax output can be taken as the classification output. This highest value can be accessed using an argmax unit, which returns the index of the neuron with the highest softmax output. For example, consider the output of a classifier trained to detect the words "cat" and "dog" in text input. Imagine that the softmax output is $[0, 1]$, followed by $argmax([0, 1])$ returning the index 1. Finally, in an index to class lookup table $[cat, dog]$ we find that the neural network is indicating that it thinks it saw the text *dog*.

A Convolutional Neural Network (CNN) is a classifier designed to process images and other array data without strictly following the fully connected structure of a DNN

[198] [120]. Instead, a CNN is structured as feed-forward stages containing convolution kernels called filter banks that do not communicate (connect to) other filter banks in the same layer. Each filter bank layer is followed by a neural network activation function layer (e.g. ReLU). A CNN is a deep learning image classifier which applies many small filters to an image in order to produce a hierarchical understanding of the contents of the image. Unlike DNNs, a CNN's layers are not fully connected. Layers near the input "understand" simple features such as edge detection, while deeper layers in the network summarize these low level features into higher-level concepts. The weights in the CNN correspond to the filter function performed by each small filter in the CNN. Each convolution layer is typically followed by a sub-sampling layer. The final layers in the output of a CNN form a fully connected multi-layer perceptron (a DNN) which classifies the high-level features into classes [121] [54]. As depicted in Figure 2.1, each filter bank and perceptron layer taken together form a feature map. At each level of the CNN, feature maps are subsampled using a procedure called max-pooling to create several smaller feature maps. The goal of a CNN is to reduce an image at the input to a set of features encoded into a vector (subsampling until the rank of the tensor is equal to 1, yielding an output vector), and then to perform feature classification with a fully connected DNN (e.g. to label the detected image features). In this work each pattern recognized by the CNN as a distinct class is called a context.

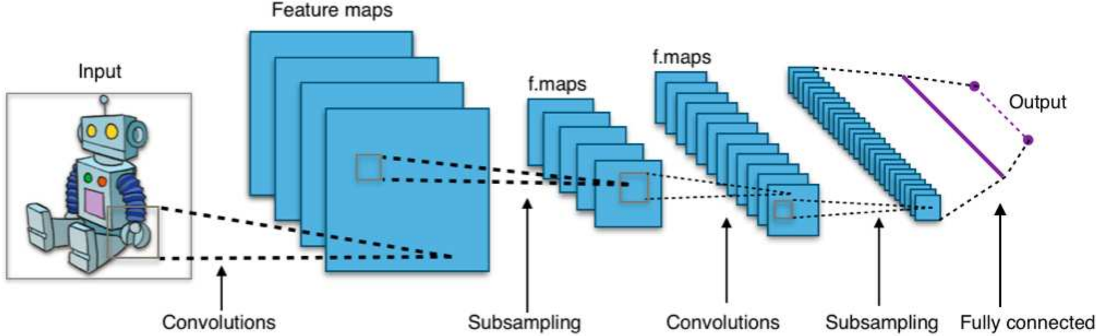


Figure 2.1: A typical Convolutional Neural Network (CNN) pipeline

DL includes three methods for learning: supervised, reinforcement, and unsupervised learning [120]. Supervised Learning (SL) is the best understood and involves training the classifier using many labeled examples. For example, images of cars accompanied by the label "car", alongside images of dogs accompanied by the label "dog" can be used to train a classifier to discriminate between images of cars and dogs. In supervised learning the classifier adjusts weights during each training iteration of processing the dataset in order

to minimize the classification error. Unlike supervised learning, reinforcement learning involves training without an immediate reward signal [227] [153]. Reinforcement learning is useful in use cases such as autonomous driving cars and strategy games, where the feedback to the learning system only arrives after some end state is reached, or after a significant delay. And finally, Unsupervised Learning (UL) is the process of learning without labeled examples organized into a dataset [88]. This form of learning gets no feedback, and therefore requires that the learner figure out a pattern from raw data and also figure out a metric for evaluating the accuracy of what was learned. In terms of information content, reinforcement learning predicts at most only a few bits per input sample (e.g. position of steering wheel and pedals to control car), supervised learning predicts at most a few thousand bits (e.g. class labels to add to an image), and finally unsupervised learning predicts anything to do with the input (e.g. given a video, predict images of the next few frames [184]) [118]. There are exceptions to this generalization about the maximum number of bits generated by these types of networks, but it is a useful heuristic for thinking about what type of approach to apply to a problem.

Transfer learning is an approach in ML where the training data is augmented by some other already trained model [172, page 243]. The advantage of using transfer learning is that it enables a model to start from some already trained set of learned features and extend this initial set of knowledge by training on additional data, rather than randomly initializing the weights and training from that random starting point. Transfer learning was accomplished in this work using an Inception v3 tensorflow CNN model that was trained on ImageNet images [2] [230] [6]. The model was extended by training a new last neural network layer on top of the existing fixed network that can recognize new classes of images after training. This final layer of the CNN received a 2048-dimensional input vector for each image, after which a softmax layer is added. As explained in [178], for N labels this CNN learns only N + 2048*N model parameters corresponding to the learned biases and weights. This is a vast decrease in the number of parameters to learn over training all layers of the model.

Training a neural network using supervised learning to perform a function such as classification can take a long time [171]. Consider for example the time required to do simple linear regression on sample points. The line of best fit slowly adjusts to fit the data until the training is completed. In a deep neural network, the system must process and backpropagate the data several times, optimizing in each layer the weights and bias for each neuron. The low-level training process notwithstanding, training artificial neural networks is a slow process. However, once the neural network is trained to be

a classifier, it can perform classification much faster than the training time. In general one can expect that runtime for a neural network is much faster (relatively speaking) than the training time. General Purpose Graphics Processing Units (GP-GPU) can be leveraged to accelerate neural network training time and runtime by executing compute operations in vector form on highly parallel and pipelined processing units rather than the more sequenial coponents of a Central Processing Unit (CPU) [171]. To accomplish this speedup, a software package such as TensorFlow or Theano is used to load and store data into and out of the GP-GPU memory from the hard disk [6] [31].

The training effectiveness of supervised learning can be enhanced by injecting random noise into the inputs to each layer of the neural network during the training process [37] [212], and by randomly dropping out inputs in each layer of the neural netowrk (dropout) [221]. Dropout and random noise injection each help to prevent the model from overfitting the data during training.

Feature engineering is the process of applying domain knowledge and human data analysis to strengthen predictive models such as neural networks [49] [203]. The idea of feature engineering is to reorganize existing data into new formats that the learning system can learn more easily. For example, humans perceive the direction "walk straight" more effectively than "walk in the ventral direction orthogonal to the plane formed by the corners of your torso that faces out from your eyes". These two statements contain the same information, but presenting this data in an easy to process format makes all the difference. In feature engineering that can mean re-encoding the data between acquisition and training.

The word "filtering" in this thesis is used to describe a particular feature engineering approach (Contribution 2: Fast filtering of text), but is also used to describe the prioritization of recommendations (Contribution 4: Recommendation filtering). These two types of "filtering" are totally unrelated.

Supervised learning can be thought of as a clustering problem, where a classifier must learn a linear boundary function between two sets of labeled points on a 2 dimensional graph. This idea is illustrated in Figure 2.2. In reality this graph could be of higher dimension, the classifier function could be nonlinear, and the graph could contain more than 2 classes, but the idea serves to illustrate the point of what a SL classifier is doing. It learns a classification function based upon labeled data in order to be able to classify novel data. Unsupervised learning can be thought of as a similar clustering problem, with all of the points having no labels, as shown in Figure 2.3. The main difference here is that in the unsupervised learning case, it is not known to the algorithm which points

belong on which side of the line. Worse yet, it is not known ahead of time how many clusters the data should break into.

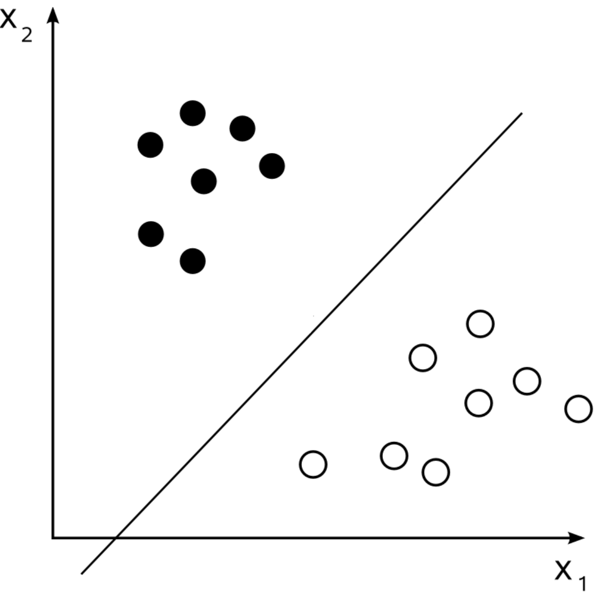


Figure 2.2: Supervised learning problem illustration

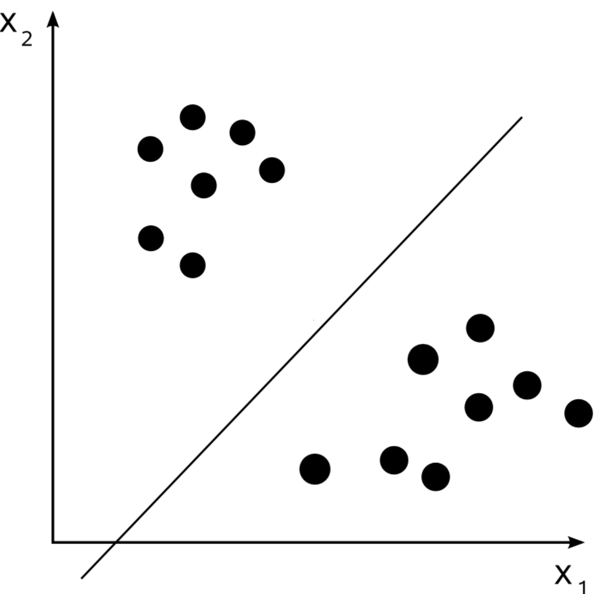


Figure 2.3: Unsupervised learning problem illustration

A good example of an unsupervised learning algorithm is Google News story clustering [53]. The system collects similar stories based on their content, and presents them to the user in an organized way. These stories are organized by their content, rather than by an editor. On a related note, using this same dataset, a 300-dimensional set of approximately 3 million vectors extracted/trained from the Google News dataset of approximately 100 billion words is used in this work [147].

In this work, unsupervised learning is considered in the domain of RS. This means learning new recommendations from unlabeled recordings of computer state and user action data. Unlike reinforcement learning and supervised learning, unlabeled data means that there is no error or reward feedback signal available to create a cost function based upon which the quality of new recommendations can be evaluated. The "right" answer is simply not known to the system. The UL algorithm must make its own decisions about creating image classes and text classes (creating keywords and contexts), and deciding the association between them (what keyword belongs in what context).

Neural network approaches to unsupervised learning include dimensionality reduction (such as t-sne with the goal to cluster high dimensional data into lower dimensional data while maintaining the information encoded in the distance between points [130]), Hebbian learning (learning through experience, popularly known by the idea that neurons that fire together, wire together [85]), hierarchical Self-Organizing Maps (SOMs take a population of samples and their attributes, and learns a representation of these points in a model which iteratively adjusts its weights over many iterations one "winner neuron" at a time [135] [186]), unsupervised word embedding (building a model that encodes the meaning between words as described further in Section 2.5 [148]), and generative adversarial networks (two neural networks, a verifier and a generator, face off where the objective of the generator is to synthesize fake data that fools the verifier into classifying the data as "real", while the objective of the verifier is to minimize the error when classifying the output of the generator [71] [184]). K-means is one of several non-neural clustering approaches, and it is similar to hierarchical SOM in that the centroid for each cluster is adjusted for many iterations until the points nearest to each centroid are not closer to another centroid [172, page 5].

Comparison of Probability Density Functions (PDFs) is a useful tool for selecting among discrete and finite elements such as recommendations from a database [41]. The distance between two PDFs is equivalent to a Bayesian probability [41]. In probabilistic modeling, the Markov condition (also called the Markov assumption) is the idea that one can assume that the current state of a Bayesian network of probabilities is independent

of past sequences of states, and so only the current state and the previous state need be considered in order to model the Baysian network [38, page 130]. Positing this assumption simplifies calculations considerably when modeling a Baysian network. DNNs and CNNs do not contain state information. A given input deterministically results in a given output. Other types of neural nets such as Long Short Term Memory (LSTM) or Recurrent Neural Nets (RNN) contain state information and in addition to containing memory units, the output of the neural network can feed back into the input [198]. RNN do not make the Markov assumption, and even so it is difficult for RNN to encapsulate long-term relationships [30]. In this work LSTM and RNN were not used, as there was a conscious effort to keep the Markovian assumption in an effort to ease the feasibility of developing an unsupervised learning capability, as further discussed in Chapter 9.

UL is often applied to facilitate the success of SL [198, page 14]. Often the UL is applied to encode raw data into a form that an SL algorithm can succeed with, where the SL algorithm would not succeed on the raw data. For example, pre-training autoencoder weights prior to applying SL with backpropagation [25], and pre-training RNN [197, page 17].

An important idea to present at this point is learning hierarchical representations. The idea presented previously above, that deeper RBM layers in a DNN encode more complex information, is only one type of hierarchy. Another type is building up a representation over time, learning new classes by integrating previously learned classes [198, page 8]. This becomes an important idea in Chapter 9 when learning new information with unsupervised learning is accomplished based upon all previously learned information.

This section concludes with a discussion on approaches to unsupervised learning of semantic word similarity. Learning semantic word similarity facilitates clustering words into related topics [248]. A reference textbook on learning semantic similarity is [84]. AVRA's approach is described in Chapter 9. AVRA's algorithm was developed because no suitable unsupervised learning algorithm was found in the literature that solves the specific image and text pattern combination learning problem that AVRA addresses. Relation extraction, named-entity extraction, information extraction, set expansion, and semantic similarity models all aim to expose relationships between entities represented by words or phrases in a corpus. This set of approaches to learning the semantic relationships between words is applied in many fields such as search engines and question answering systems [149].

AVRA's unsupervised learning algorithm finds the similarity of keywords to topics,

and the similarity of screen images to learned image features. To learn the relationship between keywords unsupervised, a corpus of online text is extracted and analyzed similar to the approaches of [47], [82], [174], [59], [241], [42], [150],[246], [247], [12], [149] and others. To model the semantic relationships between words in the corpus, word embedding is a common approach (e.g. [47], [248], [150], [126], [12], [149]). Approaches to unsupervised learning applied to semantic word similarity for a corpus obtained from the web include [47] (keyword extraction from spoken documents), [59] (named-entity extraction), [241] (synonym identification), [150] (identifying relationships in a medical corpus), [246] (set expansion), [149] (relation extraction), and [12]. AVRA follows the approach of [12] to iteratively grow topics one keyword at a time based upon the detected context. AVRA also uses the concept of a "Class Vector" introduced in [12] to represent each topic, and the cosine similarity was used in both approaches to measure the distance between vectors. Furthermore, [12] included a crawling mechanism and removed stop words. Named Entity Recognition (NER) was the goal of [12] and the context surrounding the named entity was textual, forming a Bag-of-Context-Words. AVRA instead focuses on keyword recognition (not NER), where the context is visual: what the computer screen looks like when the keyword is detected. Another difference is that [12] focused on different levels of query complexity (Focused, Very Focused, Unfocused) whereas AVRA makes no such distinction between keywords.

2.2 Quality Measurement for Systems and Classifiers

A confusion table or confusion matrix is a supervised learning classifier characterization entity which records classification results for some dataset after when assessed by a classifier [191] [140]. This type of table is useful for determining classification precision and recall (sometimes called sensitivity). Precision is the rate of true negative classification of data, while recall is the rate of true positive classification of data. The true negative rate of Equation 2.3 is also called the specificity.

The equation for precision from [191, page 780] is:

$$Precision = TP/(TP + FP) \tag{2.1}$$

The equation for recall from [191, page 902] is:

$$Recall(R) = TP/(TP + FN) \tag{2.2}$$

The equation for specificity from [191, page 902] is:

$$Specificity = TN/(TN + FP) \tag{2.3}$$

Understanding how variables in a multidimensional model are related is difficult when the model contains many hyperparameters. The Pearson correlation coefficient measures the association between independent variables [173, page 38]. It is a useful method used later on in this work to assess at a high level what variables in a model are affected by what other variables. Another way of assessing how variables in a multidimensional model are related to each other is to train a regression model on many discrete samples with the training goal to reproduce the data points in a lower dimensional space. This type of dimensionality reduction is very popular in machine learning, as it helps humans to make judgments and reason in 2 or 3 dimensions about data that exists in high dimensional spaces. This approach is particularly popular in assessing word embedding models where the "distance" between words in the model at high dimensional scale remains intact in the lower dimensional (squashed) representation. This is also called preserving local structure [129]. t-Distributed Stochastic Neighbor Embedding (t-SNE) [130] is one such dimensionality reduction tool, and is used in this work to visualize a multidimensional dataset.

The Receiver Operating Characteristic (ROC) is a graphical method of assessing the trade-off between the sensitivity and specificity probabilities [137]. The vertical axis of the graph represents the true positive rate of classification, and the horizontal axis represents the false positive rate of classification [23]. Ideally a classifier will have all true positive and no false positive classifications: an ROC curve drawn from bottom left to near the top left of the graph and then across to the top right. However, the ROC curve plotted on the graph represents the real (measured) trade-off between sensitivity and specificity from observations on test data. The overall accuracy of the classifier is measured by the area under the ROC curve, where 1 (the area of a unit square) is the best possible score. Two excellent practical sources relied on in this work to produce ROC curves are an example from Google using python and TensorFlow [185] [232] and a scikit-learn tutorial on plotting ROC curves for multi-class classifiers [201] [176]. On an ROC curve, results above the y=x line are considered meaningful, while below the line is considered worse than chance. Curves below this line can be produced by an algorithm using chance alone to classify the input. A convenient method for converting ROC area under the curve metric into quality labels used in [22] for recommender systems classification is the following: above 0.9 is "excellent", above 0.8 and up to 0.9 is "good",

above 0.7 and up to 0.8 is "fair", above 0.6 and up to 0.7 is "poor" and between 0.5 and 0.6 is considered a "fail". Below 0.5 is worse than chance, as so was not assigned a label. We will call less than 0.5 "worse than chance".

Another way of evaluating a classifier is with a precision-recall curve [199, page 158], which is commonly applied to text classification problems. The difference between ROC and precision-recall is subtle but important. Whereas ROC reveals the trade-off between the true positive rate (also called sensitivity and recall) against the true negative rate, the precision-recall curve instead reveals the trade-off between that same true positive rate and the precision of the classifier. Precision, measured by the precision-recall curve and defined in Equation 2.1, reveals how often a classifier is correct in the predictions it makes. And so the precision-recall curve is useful in selecting the level of precision required in order to achieve high recall. To assess the trade-off between precision and recall on the precision-recall curve, the F measure can be used [199, page 156]. The F measure computes the harmonic mean of the recall and precision. However, not that other trade-off points on the precision versus recall curve are valid points to select from if the weight of one measure is more important to the classifier outcome than the other (e.g. precision more important than recall). Another interesting point to think about is is the point where precision is equal to recall. This is called the Break Even Point (BEP) [199, page 161] [23].

In the trade-off between precision and recall of a classifier that can return multiple ranked results, it is interesting to evaluate the overall classifier performance in one metric. To assess the entire precision-recall curve with one metric, the Mean Average Precision (MAP) is applied [199, page 159]. It gives more weight to correct results the higher they are ranked.

When measuring latency and execution time, it is important to cast qualitative labels such as "slow" and "fast" into quantitative measures for comparison and analysis. This thesis refers many times to a "reasonable response time". Between 0.1 seconds and 1 second of delay, the user flow of thought is not interrupted, while 10 seconds is the limit for keeping the user's attention regaring an onscreen event [167, page 135]. Response times should be as fast as possible, but of course processing latency is a reality that can force the user to wait while the computer processes information. In this work a 3 second delay is considered a reasonable response time between information appearing onscreen and the appearance of a recommendation to the user. Note that the user may be unaware that the 3 second time window has started. The first notification to the user is the recommendation. The reasonable response time is instead measuring how long

after an event a recommendation is still timely. The usefulness of the recommendation likely decays to nothing over time as in a traditional soft real-time system.

The concept of speedup is useful for comparing the runtime of a program running on different machines, or with different configuration parameters on the same machine, or for comparing competing options to implement a solution. Speedup is defined as follows in [87]:

$$Speedup_{overall} = Execution\ time_{old} / Execution\ time_{new} \tag{2.4}$$

When reporting execution time for several samples, the statement "$X \pm Y$ seconds" means X seconds was the mean ($\bar{x}$) and the standard deviation (σ) was Y seconds.

To this point the thesis has presented approaches and metrics for evaluating classifiers, but not a framework for comparing two classifiers to each other, or for evaluating the success or failure of an approach. To that end, Measures of Effectiveness (MOEs) are defined to be able to evaluate how an implementation performs in comparison to the stated objectives (regardless of implementation details) [189], and Measures of Performance (MOPs) specify the technical performance characteristics the system must meet. Each MOP should relate back to one or more of the MOEs, which are the base requirements from which all system measurements derive meaning [189]. Both MOE and MOP correspond to measurements that can be recorded, although MOEs and MOPs may sometimes conflict, as the MOE may demand better performance than a real system measured by an MOP can deliver. In other words, the MOP measures a capability whereas an MOE measures a feature. Both MOE and MOP do not naturally aggregate into an overall score. Rather, these measures capture different dimensions of performance. Also, MOP and MOE are typically applied throughout the life of a project, while in this work the methodology was applied only in the final stages of data analysis.

2.3 Recommender Systems (RS)

Recommender Systems (RS) are systems which recommend items to a user, often based upon some data about the user and the items to be recommended. Advertisements appearing alongside search results are a good example of an RS application, where the search engine uses information such as the keyword(s) searched, the location of the searcher, the time of day, the bid of the advertisers, and other information to present useful targeted advertisements. A search engine itself is a recommender system, as it

recommends results based upon keywords. The review article [35] provides an overview of the RS state of the art, and the textbook [188] provides an in-depth explanation of the topic, methodologies and related issues. This section is based on those sources.

A key aspect of RS is scoring and ranking recommendations in order to present high-quality options to the user. Content-Based Filtering (CBF) is an approach where the recommendation score is increased if related items were rated positively in the past. CBF makes it more likely for items and topics preferred in the past to be recommended in the future [24] [175]. CBF provides a mechanism to rank recommendations where the recommendation score is increased if items related to the recommendation in question were rated positively in the past. As described in later chapters, CBF in AVRA involves items (action recommendations), and ratings (created by the user accepting an action recommendation). The score of a recommendation can be modified based on both topic and item similarity as described in [196], and the user's tagging history can be used to inform the likelihood of recommending an item as described in [68].

Scoring recommendations based upon context and user preferences is described in [251], which combines CBF with taxonomic preferences, and also is discussed in [136], which combined CBF with knowledge of the domain. Context-aware recommender systems can focus on using the context of the user (as in [243] where the user profile can guide the recommendation score) and/or the context of the recommendation (as in [10]) to modulate recommendation scores. The cold start problem is a situation where the RS has insufficient information about a user to make high-quality recommendations [35]. The input to the RS model can modify the output (recommendations) provided by the RS. For example, the confidence of the RS that the user will prefer a recommendation can rely on time-varying inputs outside of the RS, such as the time of day, weather the user is logged in, the page from which the user arrived to the current page, and many other examples of state information. In this work the classifier confidence levels represent state information. These confidence levels indicate the classifier confidence that graphical or textual information has been detected. The confidence of the system that it sees what it thinks it sees modifies the RS prediction.

Additional application-specific RS provide interesting context to this discussion on recommendation quality. Reverb [195] is an IDE plug-in that recommends previously visited web pages related to code being written by a programmer in the IDE. The idea is to reduce false-positive recommendations when offering the programmer a recipe to look up as she is writing a program. This increase in quality is accomplished by only recommending pages that have been previously visited by the user. A predecessor of

Reverb is Fishtail [194], which recommends web pages for the same purpose but without restricting recommendations using the user history as Reverb does. The consequence of recommending pages based only upon keywords is lower quality recommendations [160]. Another interesting example from application-specific RS is [215], an MI system that works in tandem with the programmer to expose programming recipes and other useful programming tips. The work of [215], [194], [195] and others integrate into specific programs, rather than processing images of the computer screen as a whole, which would loosen the integration between the applications and RS. However, in the unsupervised learning chapter, the idea of watching the action history of the user was employed. Employing probabilities to modify recommendations is a common approach to dealing with uncertainty in the RS domain [188, page 64].

Recommendations from a learning RS should change over time. Concept drift is the idea that the goal of a learning system can be a moving target [238]. User tastes change. Recent actions are more significant than actions performed long ago. A variety of recommendations can keep the user attentive to the RS, while consistent recommendations allow the user to rely on recommendation predictability. Concept drift can manifest when there is an aggregation of learning errors over time. Generally, a learning system that does no have the ability to forget or correct the objective, will over time increase the difference between the perceived objective and the true objective. As discussed in [256], when a learning system is approximating a concept based upon a hidden context, a change in the context can alter the concept that the learning system was approximating. The resulting concept drift can be corrected by forgetting examples and hypotheses over time, storing data (e.g. observations of use behaviour) for reuse when sufficient data is available to make more reliable decisions (e.g. learn a new context, or adjust an existing one), and monitoring the system's behavior (a loss function) so that the system obtains corrective feedback. Taking these ideas into more concrete terms, some ideas to avoid concept drift are: enforcing a time window on data acquisition and retention, learning new recommendations and adjusting existing ones based upon many examples rather than one-shot learning, correcting mistakes learned into the recommender system by integrating recommendation feedback. As [56] notes, a neural network that learns from observational data can experience concept drift if the underlying data distributions change over time. It the case of learning new recommendations with unsupervised learning, errors integrated into the model early in model development cause much more concept drift than those learned later on. An incorrect data point represents a large fraction of the overall model in a small model, but only a tiny fraction of a large model. This

is related to the cold start problem discussed above. Initializing an RS for new users with supervised learning helps to overcome the lack of high quality data, and initial concept drift.

The idea in this work for shallow integration between the RS and the computer relates to prior work whole-screen neural network image processing in [153] and high-level query processing proposed in [43]. The concept in [43] is that shallow clues in a query can hint at the correct databases to search in. Therefore, a query to many databases need not be written for each database, rather, a high-level intermediate query engine can dynamically steer the query to the right databases. The key information to understand from [43] is that the low-level database by database integration was bypassed, and instead a dynamic high-level database integration was achieved. The concept in [153] was to develop one deep-learning artificial intelligence that can play many different video games. The breadth of different applications played by the same neural network was an impressive validation of the power of reinforcement learning.

A recent development in RS using neural networks called wide and deep learning combines the speed and generality of shallow wide input feed-forward neural networks with the feature extraction capabilities of deep learning [48]. Sparse input data (mostly empty matrices linking events to actions) allow for shallow neural networks to recommend actions in general cases essentially by memorizing patterns, while special cases requiring high dimensional feature detection are not detected. Deep neural networks can learn to extract these hidden features using deep learning. Wide and deep learning combines these approaches into a single neural network with a deep branch and a shallow branch to the network, terminating at a shared softmax output. When tested to provide app installation recommendations in the Google Play app store, wide and deep learning outperformed (increased the number of times users accepted recommendations) compared to past approaches based on wide neural networks, deep neural networks, and the combination of wide and deep learning using separate neural networks.

Desktop assistant software for learning and predicting actions was described in [131]. In that work, a distinction is made between types of information extracted by the RS: action features describe items that happened recently on the computer, while state features describe the current state of the machine. Examples of action features are a history of program calls, the stream of keyboard characters, and onscreen streaming video. Examples of state features are a list of programs that are currently running, the current directory, and the current language settings. In [115] recommendations are mapped to function keys, and the goal of the RS is prediction of the next command to be typed into

a terminal program. Recommendations from this type of RS are based upon previous input (action features), but cannot see the current or past output resulting from the execution of these commands. The keyboard keys `F1` to `F5` were mapped to five different command recommendations presented onscreen to the user.

Some deep learning desktop assistants do not wait for user input to act on recommendations, as they form a narrow function. For example, the assistant described in [89] uses a camera and facial recognition to detect an approaching manager in order to hide the screen contents of a cubicle worker. A similar program can change the screen brightness to match the ambient lighting conditions in a room [44].

Types of personalized RS are content-based/cognitive (identify the common characteristics of items that have received a favorable rating from a user), collaborative filtering (obtain recommendation ideas by extracting data from similar users), and hybrid approaches between these two concepts. Relying on cognitive recommendations alone leads to overspecialization (only recommending items like those that were rated by the user, losing the novelty of the RS) [188, page 38]. Collaborative filtering of recommendations is the process of adding recommendation information that applies to one user `A` into the recommendation information for another user `B`. Counting and user preference for a recommendation as a a rating, the RS can create a database of user preferences organized by user. Collaborative filtering should provide users with novel suggestions the system predicts with high probability that the user will desire. Two approaches to implementing collaborative filtering are model-based and neighborhood. With the neighborhood approach to collaborative filtering, recommendation ratings stored in the RS database for all users can be used to predict ratings for a specific user. A user-based approach to the neighborhood collaborative filtering is to identify users with similar ratings patterns and to import these user preferences into the current user, even though the current user has not specified these preferences. An item-based approach to neighborhood collaborative filtering is to predict the ratings of a user based upon their past ratings for similar items. With the model-based approach to collaborative filtering, a predictive model is trained on recommendation history in order to produce/generate novel recommendations for users.

To accomplish collaborative filtering with the user-based neighborhood approach the first task is to identify for a user `A`, similar users such as user `B`. User `B` can be identified by comparing the recommendation contents of both users, combining into one score the similarity between any pair of users by scoring the Bayesian distance between their recommendation database entries. Another approach is to score the Bayesian distance

between their tastes for the learned recommendations. Given a close enough similarity, with a threshold defined either by the user or the system administrator, entries from user `A`'s database can be copied into the database of user `B`. This applies both to context information and to keyword information. Bayesian recommender systems are described in detail in [83] and simple example showing how Bayesian-inference can be applied to social network data with either user-based similarity of recommendation-based similarity is [152].

The effectiveness of an RS depends not only on accurate predictions. It also depends on on human factors, such as how best to convey information to user [188, page 21]. As described in [228] and [139], an RS must build trust with the user, perhaps by generating consistent results. It should also recommend novel recommendations to keep things fresh for the user, and should provide rich content. An example of rich content in the movie recommendation domain is a movie recommendation containing a title, description, community ratings, and a related image, rather than a title alone. These details about recommended items enable the user to become more involved in the RS process, and to think of the RS as an extension of their thinking process. The RS should also expose functionality to refine recommendations by blocking specific topics (e.g. recommend no western films), or specific items (e.g. no recommendations for this movie title). These ideas about how to convey information to the user continue in the next section of the prior art regarding mixed-initiative systems.

2.4 Mixed-Initiative (MI) Systems

The research motivation section of this thesis described the ability to shift the cognitive effort of human-computer interaction to the computer. MI systems are designed around the concept of assisting a human analyst to derive and take advantage of insights into data. In MI systems, the breakdown of work between the computer and the human focuses on the strength of each participant in the iterative problem solving activity [13]. An MI system is a system where a computer agent and human user interact collaboratively to accomplish a task. MI systems are designed to leverage the strengths and overcome the weaknesses of computers and machines as these two entities are good at different tasks. For example, computers can multiply large numbers quickly while humans cannot, whereas humans can reason with common sense in situations where computers cannot.

Automated assistant technology is available today for accessing content using natural

language (bots [224], [144]) and for controlling basic computer functions (e.g. facebook M (beta) [193], Google assistant [128], Watson [97], Viv [113], Hound [220], Assistant.ai [20], Alexa [15], Google Now [73], Cortana [142], NextOS [166], Siri [21]). Such agents can be applied in a variety of fields beyond computer assistance, including industrial, commercial, medical, entertainment and others [104]. It is useful here to make the distinction between a program and an intellignet assistant. A taxonomy of the properties of autonomous agents was presented in [63] including the following qualities: reactive, autonomous, goal-oriented, temporally continuous, communicative, learning, mobile, flexible, and character. Types of intelligent assistant mentioned in [170] are collaborative, smart, mobile, informational, reactive, hybrid, and smart. In deciding if an assistant is intelligent one can also consider the implementation details in addition to the software qualities [158].

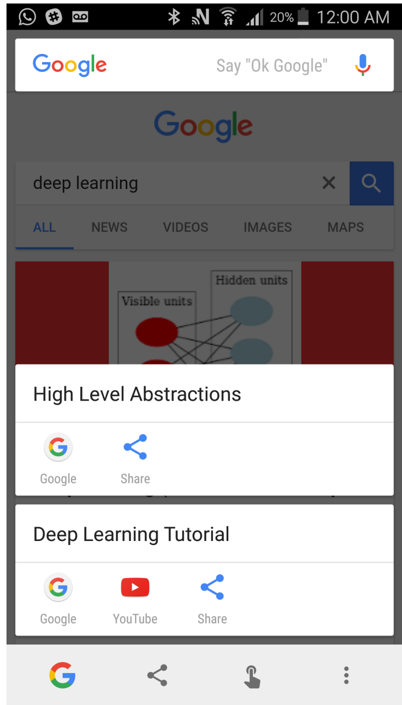
Context awareness involves detecting situation-based information in a computer screen image in order to narrow down the search for onscreen meaning to a select few contexts. Context awareness is achieved in this work using shallow integration. Cortana contains a proactive action recommendation feature that can recommend registered actions to the user based upon a history of observing the user on the computer and digesting user data such as calendar and search history [142]. According to Microsoft "Cortanas insights are situations or conditions based on her understanding of the users context or intent" [145]. Only registered proactive actions can be suggested to the user. For example, general interest categories can be activated by the user, and notifications regarding these topics can then be pushed to the user's screen.

Google Now [73] contains an extension called Google Now on Tap [74] (also called "screen search"). When the software is installed and the user holds down the Home button on a mobile phone, the program analyzes the screen and presents related information, applications to launch, and actions to perform. This seems like a very closely related idea to AVRA and so an experiment was performed for this thesis to understand how this feature is analyzing the screen. In order to assess the level of image processing performed, the results for pressing the Home button on a given page were compared to pressing the Home button when displaying an image (screenshot) of the same page. The onscreen picture is identical in terms of the pixels on the mobile device, and therefore any differences indicate that the program is parsing the information semantically rather than using image processing. It turns out that Google Now on Tap does analyze the OCR for screenshots compared to processing content displayed to the user from a particular web page or application. However, it appears the program is also using the page or app url

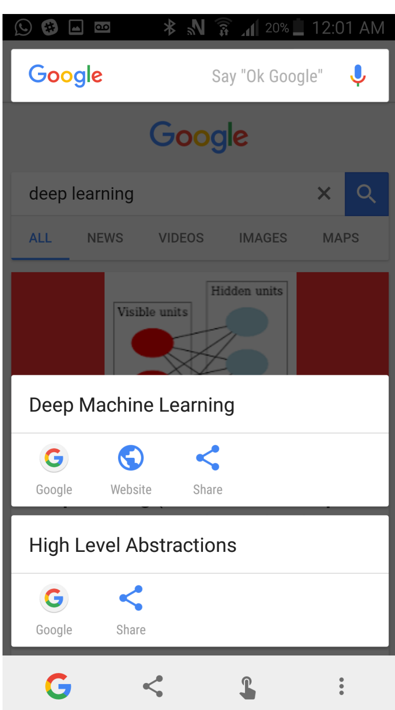
and displayed content as hints to the recommendation engine. In September 2016 the results for screenshots were much worse than for in-app images, whereas is January 2017 the results for these cases is much closer. Figure 2.4 shows two examples, one where content is detected for an application but not for an image of the application, and another where content is detected for both the application and an image of the application. The difference seems to be that OCR is feeding the recommendations text, but a CNN is not providing image labels. Furthermore the strange font in Figure 2.4d probably prevents the OCR from understanding the text in the image, and without any other features to assess, the system gives up.

Shallow Integration or Shallow Application Integration in this work is the approach to understand what is happening in the computer without integrating into each application running on the computer. This is accomplished by analyzing the a computer screen image to extract text and context information.

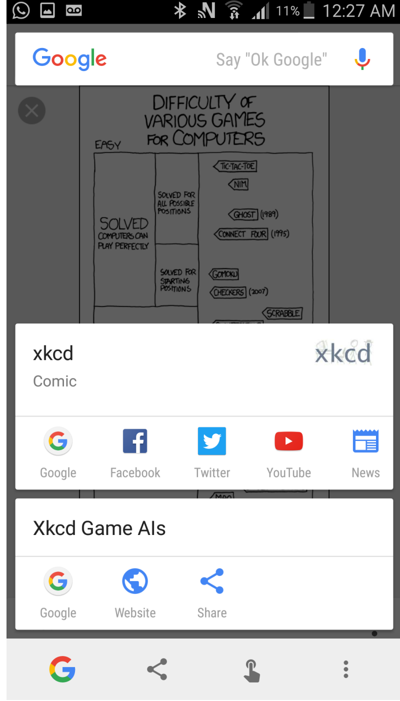
Computer assistants can come in the form of assistive technologies such as screen readers for the blind. This is an interesting area to think about because these screen reading assistants are tasked in one form or another with interpreting the screen and performing actions on the computer. The American Foundation for the Blind links to several products of varying capability to help blind users to operate a computer [16]. These systems generally do not meet the definition outlined by [63] for an intelligent assistant. For example, BRLTTY is software that reads commands as they are typed and as they appear onscreen [146]. BRLTTY only works inside a terminal window. It is not goal oriented or flexible, has no personality or character and does not learn [55]. Screen reader software enables users to perform many basic computer functions within a fixed framework. It reads parts of the screen, can announce where on the screen the cursor is located, and can perform actions when the user presses one of several hotkeys on the keyboard. The software operates by parameterizing the screen components and cycling through them. Screen readers understand the screen programattically using semantics rather than visually using images. This is easy to verify by creating a custom checkbox element in HTML and noticing that a screen reader announces the element as "group" rather than the correct "unchecked checkbox" [75]. Some screen readers accept voice commands and others are connected to OCR software in order to facilitate reading books and magazines for the reader, but these products do not interpret graphical information on the computer screen itself. A new wave of software for mobile phones performs real-time word translation including word recognition (e.g. word lens, now integrated into Google Translate [219] [33]). Mobile apps for object recognition such as Aipoly [11]



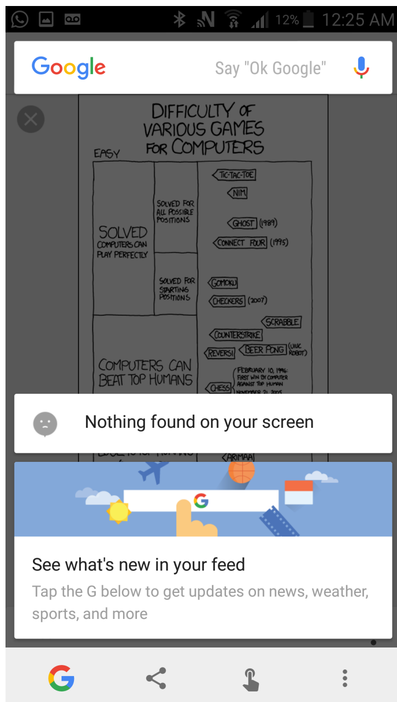
(a) Google Now on Tap result in-app



(b) Google Now on Tap result for screenshot



(c) Google Now on Tap result in-app. Background image by XKCD [159]



(d) Google Now on Tap result for screenshot. Background image by XKCD [159]

Figure 2.4: Comparing Google Now on Tap response in-app v.s. screenshots

translate images of objects into announced labels, and apps such as Wayfindr announce step by step walking directions to the user directions [250]. Aipoly contains learning neural networks, and in closer to an intelligent assistant than the aforementioned screen readers. It interprets image data from the environment in real-time and classifies it into labels.

A prompt is an action proposal or information message interrupting the user's focus. It is triggered based upon a timed event (e.g. after 8 seconds pop up an informational dialog box), the detection of a particular context (e.g. if function return is null, pop up an error message dialog box), or by some acquired intelligence (e.g. when an email marked urgent arrives in the user's email inbox, play a ringing sound to alert the user) [92].

Two sources for MI system design advice are [51], and [132]. The design principles (A1) through (A5) from [51], and (M1) through (M6) from [132] are listed below. These design principles are discussed further in Chapter 3.5 in relation to AVRA.

ADE

MI system design principles in [51] were based upon a meeting of subject matter experts discussing the design of the Active Data Environment (ADE) prototype, an MI system for automating sensemaking by recommending tasks in real time. The ADE system was tested against the MI dataset from [253] where the task is foraging through data in search of connections around which to form a hypothesis. The ADE system exposes data insights to the user without being asked for specific information from the user. ADE's automated recommendation system employs content-based pattern recognition mixed with task awareness to identify candidate recommendations. ADE recommends data and relationships based on a task model, enabling the user to co-reason with a virtual agent about data in a single spatial workspace. Task recommendations, based on user interaction history and pattern recognition, are generated graphically within the workspace and can be invoked by the ADE user. ADE's authors outline design principles for MI systems including (A1) through (A5) below.

(A1) User control and non-interference

ADE was designed to be flexible and allow users to remain in control of their process. Recommendations should enrich, not dictate, the process of foraging for data and synthesis of recommendations. ADE was designed to remove routine tasks for the user, without

adding additional tasks [57]. Another important aspect of any MI system design is the prompt mechanism employed by the agent to alert the computer user.

(A2) Recommendations should enhance the current task with timely useful information

(A3) Understanding what a recommendation signifies and why it is recommended

Reducing cognitive effort is discussed in [160] from the perspective of the user interface design for RS. Also in [160], there is a discussion on the user understanding the reason for a recommendation being presented, and the danger to the reputation of the tool in the mind of the user when no meaningful explanation for a recommendation is accessible. See also (M5) below.

(A4) Fast visual interpretation of recommendations

Recommendations should be "presented in compact visual form, to allow users to quickly assess the recommendation and explore or dismiss it with minimal effort." [51].

(A5) Retaining recommendation context

See also (M5) below.

MIVAS

MIVAS is another MI prototype system providing a template for MI system design [132]. Visual Analytics involves organizing data to form hypotheses. MI adds machine learning to VA, assisting the user in the task of sensemaking [132]. Reasoning about complex datasets requires detail-level and high-level thinking, and MI visual analytic systems complement human cognition with analytic techniques. The goal of the collaboration between human and virtual agent is producing insights leading to one or more conclusions regarding the data. More generally, adopting such a system should reduce the cognitive pressure on the human user by offloading some computation to the virtual agent. To that end, iterative sensemaking is the process of working with the data (often large data sets with semi-reliable structure) to produce intermediate results along the way to a conclusion [106] [265]. Each sensemaking iteration involves two steps: foraging and

synthesis. These steps are easier to perform quickly with high-level (pattern) and low-level (coding) assistance from a predictive intelligent system. MIVAS was designed to address the field of Big Data with MI as a three part system containing a human user, a visual workspace, and a predictive recommendation engine. MIVAS authors outline design principles for MI systems including (M1) through (M6) below.

(M1) Speed improvement

An MI solution for a task must provide a faster user experience than manually performing the same task without the MI system, or utilizing a non-intelligent programmed solution.

(M2) Data wrangling

(M3) Alternative discovery and comparison

(M4) Parametric interaction

MIVAS employs a feedback loop where a graphical visualization is manipulated by the user to provide feedback to the MI system, and in turn the system analyses the feedback and sets new model parameters, changing the data representation in the workspace.

(M5) History tracking and exploration

(M6) System agency and adaptation

2.5 Text Similarity Algorithms

To motivate this topic, consider a character-level text classification task performed by a deep neural network on noisy text. How can this task be accelerated? A neural network classifier takes candidate text as input, and outputs a classification, where each class trained into the neural network is associated with a unique keyword. A *keyword* in this work is a text string that a classifier can be trained to recognize. A text comparison algorithm could quickly discard candidate text prior to classification when the candidate text is not similar to any *keywords* known to the classifier.

Neural-network-based classification relies heavily on slow mathematical operations such as multiplication and division. Character-level keyword recognition in noisy text is one case where much of the data fed to a neural network classifier may be irrelevant, and processing the irrelevant text is computationally expensive. Consider, for example, a

situation where noisy text is recognized by a DNN as one of several keywords, or as noise. This noisy text is the output of an Optical Character Recognition (OCR) system which has processed a computer screen image into text. Classification of words in the OCR system's output should only be performed on text that is likely to be classified with some degree of confidence. This discarding of candidate text with a filtering algorithm prior to the neural network classification saves time. However, how can this filtering algorithm know which parts of the OCR output are similar to keywords that the neural network was trained to recognize? Is that not the purpose of the DNN classifier? The process of discarding input data prior to reaching the DNN is analogous to a funnel, where the first classifier discards clearly irrelevant text, and the DNN carefully classifies the surviving text. To decide which components of the text to discard, this initial classifier should use some objective measure such as the similarity between the candidate text and the set of keywords that the DNN was trained to classify. The filter should minimize latency while maintaining high specificity and recall. The trade-off between execution time, specificity, and recall is made on a case by case basis depending on the application.

To motivate the development of the filter, consider the execution time for this classification task without acceleration. A text snippet in this work is defined as a short text string extracted from the OCR of a computer screen image. These snippets can contain up to two space characters. As depicted in Figure 2.5, each task involves classifying 3,000 text snippets into one of 300 classes, where 99.5% of the candidate input texts (snippets) are not part of any class trained into the DNN. The expected latency for this task should be under 1 second on average. As a first experiment, a DNN was deployed with 784 inputs, 625 neurons in the first hidden layer, 625 neurons in the second hidden layer, and 5000 output neurons. The DNN was trained to recognize text containing spelling errors. 10 samples were collected for CPU and GPU execution time for this character-level text classification task. The mean execution time required to classify the 3,000 candidate text strings was 433.9 seconds on a CPU (1.80GHz Intel® Xeon®), and 333.1 seconds on a GRID K520 GPU. This task is accelerated in this work by filtering out candidate input text that is not likely to be classified accurately. As described in the following sections, this approach can be implemented to complete the task in under 1 second using only one computer, rather than a large GPU cluster.

The similarity of two text strings can be measured in terms of lexical or semantic similarity. Lexical similarity involves comparing sequences of characters to measure similarity, while semantic similarity involves studying a corpus to determine how similarly two words are used [69]. Hybrid measures are also possible [50]. For example, [100] mixed

semantic text similarity extracted from a corpus of text with lexical similarity obtained from comparing character sequences to improve document comparison. Text similarity metrics are applied in many fields including, for example, spelling correction, where an autocorrect tool recommends words with low edit distance from a non-dictionary word [36], and plagiarism detection, where similar texts are flagged for review [225].

Matching noisy text to a list of keywords is also referred to as fuzzy string matching, string matching allowing errors, and approximate string matching [163]. The Levenshtein distance is a lexical approach to quantifying text similarity by finding the minimum edit distance between two text strings [124]. The idea is to count the minimum number of operations (insertions, deletions and substitutions) required to transform one string of text into the other. The Needleman-Wunsch distance (also known as Sellers Algorithm) adds to the Levenshtein distance the idea of a distance dependent substitution cost [165]. It searches for approximate substring matches. The Smith-Waterman distance is a metric for genetic sequence matching [217]. It finds the longest common sub-expression between two strings with a smaller penalty than the Levenshtein distance for prefixes and suffixes adjacent to the common substring [162]. The Monge-Elkan metric is a variant of the Smith-Waterman distance function that uses variable costs depending on the substring gaps between the two strings [155].

Calculating an approximate edit distance rather than a true minimum edit distance can save execution time at the cost of accuracy [242]. Several approximate edit distance approaches such as [96], [17] and [125] provide better worst-case execution times than previous work due to improvements in estimating edit distance. The approach in these works was to reduce the per-iteration algorithmic complexity, emphasizing algorithmic improvements and utilizing low latency binary operations such as shifting, rather than slow mathematical operations such as multiplication, exponentiation and division.

The L1 distance (also called city block distance or block distance) is the sum of the absolute difference in value between words P and Q for each dimension d as in Equation

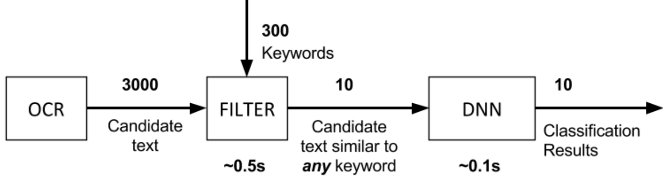


Figure 2.5: Text Classification Task.

2.5 [41]. It measures the length of a line connecting P and Q drawn along one dimension at a time. The L2 distance (also called euclidean distance) is the square root of the sum of the absolute difference squared in each dimension as in Equation 2.6 [41]. The L2 distance is the length of the line segment $\overline{PQ}$ in multidimensional space. Cosine similarity computes the normalized dot product of P and Q as in Equation 2.7 [133].

Natural Language Processing (NLP) is the study of understanding text or speech spoken or written by humans in a natural way. The goal in NLP is to come up with useful models to extract meaning from this data [32]. Word embedding is a technique for encoding the relationships between words [239] and Word2Vec is a popular word embedding tool [147] [237]. Word embedding involves creating and training a model that reconstructs the linguistic contexts of words as relationships in a high dimension vector space. Famously, a word embedding on a large natural language dataset can be used to algebraically find the vector for the word *queen* by summing the vectors for the words $king - man + woman$ [148]. Word embedding models trained on large datasets are useful knowledge engines as they capture a lot of general knowledge and context. Quoting from [148] "We find that these representations are surprisingly good at capturing syntactic and semantic regularities in language ... the male/female relationship is automatically learned". At the character level, word embedding means encoding letters as vectors, and words as sequences of vectors. This letter by letter encoding was used in this work to model the L1 and L2 distances. A similarity score is formed as the distance (e.g. L1 or L2) between points for each word in the constructed model. An average vector or paragraph vector is a summary vector created by averaging contents of many vectors and dividing by the total [117]. Average vectors are used for many purposes, including the creation of a vector for a paragraph when the vectors for the words of the paragraph are known. Common words that do not encode semantic meaning are called "stop words", and can be removed from a corpus prior to the conversion from text strings to word vectors. A commonly used stopword corpus for English text was devised by Porter [180]. When constructing a word embedding model, it is a good idea to remove common words with low information content such as "the" and "a" in a process called stop word removal. To get the same word presented in various forms to become the same vector, the variations of the word must be associated to the same vector during the tokenization process. Stemming is another text pre-processing approach that converts to a standard form each word in the corpus that will be converted into a word embedding model. More specifically, stemming is the process of chopping off parts of words (e.g. $retooling \rightarrow$ ~~re~~*tool*~~ing~~) [199]. Similar to AVRA, stemming was applied in [82] to process the results of a crawler-built corpus

prior to vectorization into a word embedding. Lemmatization finds words that mean the same thing and map them onto the same vector (e.g. $sees, sawseen, seeing$ should all get one vector in a word embedding model) [199]. Word frequency in documents contains useful information for building models of text data (e.g. tf-idf and word embedding) [199]. Stop words and stemming are applied during unsupervised learning in Chapter 9.3.

$$distance_{L_1}(P,Q) = \sum_{i=1}^{d} |P_i - Q_i| \tag{2.5}$$

$$distance_{L_2}(P,Q) = \sqrt{\sum_{i=1}^{d} |P_i - Q_i|^2} \tag{2.6}$$

$$similarity_{cos}(\vec{P},\vec{Q}) = \frac{p \bullet q}{|p||q|} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sqrt{\sum_{i=1}^{d} {P_i}^2}\sqrt{\sum_{i=1}^{d} {Q_i}^2}} \tag{2.7}$$

$$Jaro(P,Q) = \frac{1}{3}(\frac{|\sigma|}{|P|} + \frac{|\sigma|}{|Q|} + \frac{|\sigma| - 0.5t}{|\sigma|}) \tag{2.8}$$

$$JaroWinkler(P,Q) = Jaro(P,Q) + |\rho| f(1 - Jaro(P,Q)) \tag{2.9}$$

$$ratio(P,Q) = \frac{|P| + |Q| - Levenshtein(P,Q)}{|P| + |Q|} \tag{2.10}$$

The Jaro similarity is another edit distance metric shown in Equation 2.8 [103] [102]. It considers four quantities: the length of both strings to be compared ($|P|$ and $|Q|$), the number of “matching” characters σ, and t defined as the number of transpositions (characters that match but are out of sequence) [162]. Calculating σ, matching is true if two characters are not more distant that half the length of the shorter string i.e. for P_i and Q_j, $|i - j| \leq \frac{min(|P|,|Q|)}{2}$ [162].

Jaro-Winkler extends the Jaro similarity with the concept of rewarding a common prefix shared between the two strings [258]. As in Equation 2.9, the longest common prefix ρ and scaling factor f are used to increase the similarity of strings sharing a common prefix [162].

In this work the ratio metric implementation from [164] is considered, where the metric shown in Equation 2.10 is followed. The sum of the length of both compared strings ($|p| + |q|$) less the edit distance (Levenshtein distance) is divided by the length of both strings ($|p| + |q|$).

Locality Sensitive Hashing (LSH) involves constructing a one way hash function which quickly reduces an input string to a fixed-length hash with the special property that similar strings generate hash outputs with low Hamming distance between them [99]. The idea behind these hashes is to reduce the dimensionality of the data while preserving the similarity features encoded into the data. Several LSH hashes such as SimHash [46] and MinHash [210] are studied in detail in the literature. Among many other applications, these hashes are used to build an index for comparing documents to discover approximate similarity [134].

Some distance measurement algorithms do not apply to the fuzzy string matching problem considered in this work. For example, comparison algorithms such as mean squared error and Hamming distance work well for computing image similarity for images with identical dimensions, but do not work well for string similarity computations as the strings are often of different length. Truncating one of the strings to perform the comparison on equal length strings causes unacceptable information loss. q-grams [79] also called n-grams [114] are small substring snippets of a fixed length created from each of the strings to be compared [162]. This string representation is an efficient data structure for computing token-based similarity. The Jaccard similarity takes into account string similarity and diversity of two tokenized strings [162]. Tokenization-based methods Jaccard similarity, Monge-Elkan similarity, and n-grams were not considered for this work.

Semantic text similarity measurement is an evolving field with ongoing metric quality problems for various applications such as constructing word embeddings [62]. Lexical similarity metrics also suffer from quality problems when comparing short strings. In such cases the compared strings have a low edit distance even though they are not related. For example, the word 'a' is only one substitution away from the word 'I'. Therefore consider that this is an imperfect measure of text similarity. It does not account for context as, for example, a distributed word embedding does [239]. In fact, each of the string comparison algorithms presented in this work has some drawbacks and advantages based upon the constraints of the application and the data used to evaluate the algorithm's performance. The performance results presented in this work are expected to vary under datasets that differ significantly from those described below. This reality recalls the "no free lunch theorem" for effective optimization algorithms [259], which states that "for any algorithm, any elevated performance over one class of problems is offset by performance over another class" [260].

2.6 Text Classification on OCR output

This section of the thesis focuses on relevant prior art on understanding the output of Optical Character Recognition (OCR) systems. A recent survey on the general topic of text detection and extraction from images is [262].

OCR is the process of translating text from images into text strings in a computer by detecting and then recognizing text within the image data [198] [262]. Optical word recognition or word spotting is the process of detecting words as symbols in the way that objects are recognized in a scene, rather than detecting words as sequences of characters [245]. This idea of pulling out specific words from a noisy scene relates closely to Text Categorization (TC), the assignment of category labels to a document. In a book store example TC would involve an algorithm that analyzes the text in each book and sorts new books automatically into the right section of the store (e.g. fiction, science-fiction, poetry, science, kids books, etc.). With noisy TC, text is extracted from a noisy source such as OCR or speech to text, and in these cases the extraction process itself adds noise to the text [244]. The problem with training a classifier to recognize noisy text is that the classifier's model learns the noise features and is therefore less effective at understanding text without the added noise, or text with different noise characteristics [244]. In [244] it is proposed that each type of noise source can be paired with a classifier that is trained to see through that specific type of noise: text classifier 1 for OCR output, text classifier 2 for speech to text output, and so forth.

Regarding spelling, context, and grammar-related approaches to correcting OCR output, several examples are available in the literature where the text to be extracted is known in advance (e.g. [234], [151], [254]). The context of text computed as the wondow of words preceding and following each word can also be a helpful hint to OCR software as to the correct reading of text as in [254] where the OCR process attempts to extract text based upon the context surrounding it. A more narrow example of OCR correction using context awareness is [64], where transcription by OCR of mathematical formulae was improved by excluding non-syntactically valid candidate readings of the mathematical text. When processing ancient medical documents, [234] correct OCR output errors caused by processing archaic vocabulary. The approach in [234] used regular expression correction of common OCR spelling errors in conjunction with a grammar-aware and context-aware spell-checking system trained on medical terms. In [151], the goal was to extract metadata from documents using the FineReader OCR engine [8]. Rather than focusing exclusively on OCR, Support Vector Machine and Hidden Markov Model ap-

proaches were combined in [151] to build a model of the layout of the document in order to narrow the focus of the OCR system. This allowed the OCR to focus on specific locations in the document, backed by a text parsing search used to extract information fitting some expected data format. String pattern analysis was also included in the layout recognition and data analysis approach.

Spell-checking text involves learning from large sets of text in word dictionaries and online documents, and can achieve high accuracy on English language texts [252]. In [252] a spelling correction database for autocorrect systems is described where the problem involves identifying words, and then identifying how they are spelled correctly and incorrectly. The system operates without a human-designed dictionary of correct and incorrect spelling by processing a large corpus of text to learn words and their possible misspellings. In [26], an OCR post-processing algorithm using spell-checking was proposed. The system used the Google search engine results page search refinement spelling correction system (see suggested search refinements in [34, page 10]) as an online spelling correction database. The idea to correct common OCR errors with a spell-checking approach (e.g. [234], [151], [252], [26]) is explored in Section 7.1.4 as a means to correct OCR output mistakes in error message text. In [112], OCR output text is corrected using a dictionary-based approach paired with a trained regression model, where the system is trained with a noisy channel model to correct OCR errors. The idea in [26] and [112] to catalog common OCR errors and their corrections in a dictionary is employed in this work as discussed in Section 7.1.2, and is combined with deep learning to improve classification accuracy as discussed in Section 7.1.5.

Regarding black-box testing approaches to correcting OCR output, [213] addresses the problem of recognizing text from unusual typefaces or languages using OCR. For example, processing obsolete Indian font Bangla printed 100 years ago and now available as scanned documents. The approach begins with a baseline OCR engine used as a black box to process a set of unlabeled images. The OCR output text was then calculated as the centroid of a set of many candidate OCR words, measured by edit distance. This type of unsupervised pre-training is a well-known technique in deep learning for accelerating training and improving feature learning by exploiting unlabeled data [58].

When measureing the effectiveness of an OCR system in general, several quality metrics can be helpful. These metrics typically need to be compared to a ground truth, which can be provided by human transcription of the input to the OCR, or by sending already transcribed text into an image generator and then through the OCR system. Comparing the ground truth to the OCR output provides insight into where the errors in the text are

coming from. The Character Error Rate (CER) defined in Equation 2.11a represents the proportion of characters in a dataset that were incorrectly classified during image to text conversion [38, page 51] [61]. Similarly the Word Error Rate (WER) of Equation 2.12a represents the proportion of words in a dataset that were incorrectly classified during image to text conversion [61]. The CER and WER are not as important in a classifier trained only to detect specific onscreen *keywords*, which is the task considered in this thesis. The Term Error Rate (TER) is a more significant metric for TC on noisy text, as it measures the error rate for specific terms (*keywords*). Recall (ρ) for text classification is defined in this thesis in terms of TER for *keywords* that are learned terms known the text classifier, rather than all words presented to the OCR system. Therefore recall is in fact referring to "term recall".

$$CER = \frac{character\ deletions + character\ insertions + character\ substitutions}{total\ number\ of\ characters\ in\ reference} \tag{2.11a}$$

$$= \frac{LevenshteinEditDistance(ground\ truth, OCR\ output)}{total\ number\ of\ characters\ in\ reference} \tag{2.11b}$$

$$WER = \frac{word\ deletions + word\ insertions + word\ substitutions}{total\ number\ of\ characters\ in\ reference} \tag{2.12a}$$

$$= \frac{WordEditDistance(ground\ truth, OCR\ output)}{total\ number\ of\ words\ in\ reference} \tag{2.12b}$$

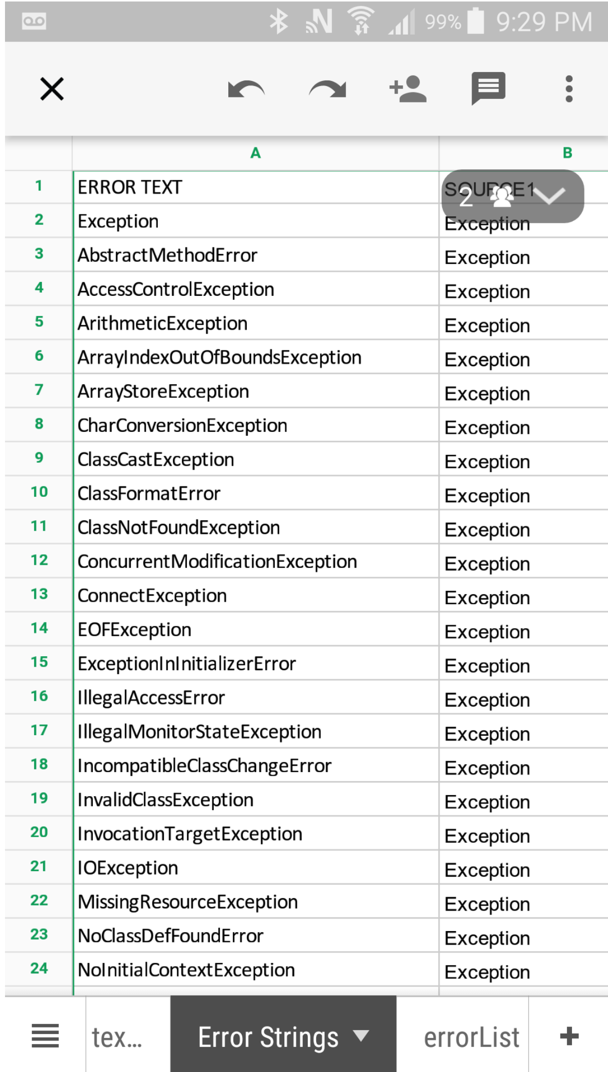
Regarding deep learning approaches to text classification when analyzing character-level information, [266] analyzed character-level data to characterize text while [33] and [101] extract text from images. All of these approaches utilize character-level text processing of some kind, and none of them assumes that the text to be processed is known apriori. The goal in [266] was to classify text, including text containing spelling errors. An example of a classification task in [266] is the prediction of the user rating of a product (e.g. 0-5 stars) given a user's comment on the product. [266] used a deep convolutional neural network with memory units (9 layers deep with 6 convolutional layers and 3 fully-connected layers) and sends text character by character into the neural network (multiple vectors per word). In this work a shallow (3 fully connected layers) deep neural network will be described for representing particular words and their misspellings (1 vector per word) in Chapter 7.

The International Conference on Document Analysis and Recognition (ICDAR) dataset is a well-known benchmark for word recognition which does not assume that the text to

be recognized is known apriori, and presents very challenging fonts and image artifacts that are not expected in onscreen error messages [107]. ICDAR TASK 1.3 involves extracting text from computer-originating images, and is of particular interest to discuss in the context of this work. Why not use ICDAR data in this work? As it turns out, the images in the ICDAR dataset are quite dissimilar from typical screenshots containing, for example, error message text. And so ICDAR is a poor test for the type of detection problem addressed in this work.

As mentioned earlier, it was proposed in [244] that differing OCR use cases result in different problems and solutions. For example, OCR failures from low quality camera images [66], and blur from low resolution documents [177] require different solutions than the fade of a scanned aging paper document [234]. In this work, concerned with extraction of error messages from onscreen text, low image resolution and non-dictionary keywords appear to be the main cause of OCR errors. The goal in [33] was text extraction from smartphone images on a character by character basis. Traditional OCR systems tend to fail at character extraction from images in conditions such as high blur, low resolution, low contrast, high image noise, and other distortions. Character level classification was achieved with a deep learning neural network with 5 hidden layers. The input layer consists of histograms of oriented gradients coefficients and three geometry features. The output layer is a softmax over 99 character classes plus a noise class. Just as this character classification system extracts features using a histogram with bins, the system in this work uses letter frequency encoding at the word level as a secondary feature of the error message text. [33] extracts text very quickly, with a mean processing time of 600 ms per image. For the cropped word recognition task the accuracy was 90.39% on the Street View Text dataset, and on the previously discussed ICDAR 2013 Robust Reading Competition TASK 1.3 was 82.21% [107]. The system in [33] is available for experimentation. An input to the system is shown in Figure 2.6 (A) and the OCR output is shown in Figure 2.6 (B). Of the 23 onscreen error messages, only 15 were correctly processed into text by [33]. The OCR output for tesseract-OCR is shown in Figure 2.6 (C) where 15 of 23 keywords were misspelled as a result of OCR processing. A common spelling mistake was to swap a lowercase L character for an uppercase I character. This example provides further motivation for the development of a specialized tool for error message detection.

In [101] the goal was extraction of text from images, and was accomplished with a convolutional neural network including character-level examination of the text in each image. In this work the goal is to accurately detect specific error message text with very



	A	B
1	ERROR TEXT	SOURCE 1
2	Exception	Exception
3	AbstractMethodError	Exception
4	AccessControlException	Exception
5	ArithmeticException	Exception
6	ArrayIndexOutOfBoundsException	Exception
7	ArrayStoreException	Exception
8	CharConversionException	Exception
9	ClassCastException	Exception
10	ClassFormatError	Exception
11	ClassNotFoundException	Exception
12	ConcurrentModificationException	Exception
13	ConnectException	Exception
14	EOFException	Exception
15	ExceptionInInitializerError	Exception
16	IllegalAccessError	Exception
17	IllegalMonitorStateException	Exception
18	IncompatibleClassChangeError	Exception
19	InvalidClassException	Exception
20	InvocationTargetException	Exception
21	IOException	Exception
22	MissingResourceException	Exception
23	NoClassDefFoundError	Exception
24	NoInitialContextException	Exception

(a) A cropped image of a webpage containing error message keywords

EOFException 15 Exception nInitializerError 16 llegalAccessError noitpecxEetatSrotinoMlagell estate 17 18 IncompatibleClassChangeError 19 InvalidClassException 20 InvocationTargetException lOException 21 22 MissingResourceException 23 NoClassDefFoundError 24 NoInitialContextException xceptio Exception exception

(b) OCR output containing 8 spelling mistakes for 23 keywords processed by PhotoOCR

```
10 11 12 13 14 15 16 17 18 19 no a +; E ERROR TEXT Exception AbstractMethodError Exception AccessControI Exception
Exception ArithmeticException Exception Arrayl ndexOutOfBoundsException Exception ArrayStoreException Exception Cha
rConversionException Exception CIassCast Exception Exception CIassFormatError Exception CIassNotFoundException Exception
ConcurrentModificationException Exception Connect Exception Exception EOFException Exception Exceptionl nI
nitializerError Exception IIIegaIAccessError Exception IIlegaIMonitorStateException Exception IncompatibIeCIassCha
ngeError Exception I nva IidCIassException Exception I nvocationTa rgetException Exception IOException Exception
MissingResourceException Exception NoCIassDefFoundError Exception NoI nitiaIContextException Exception tex... Error
Strings V errorList
```

(c) OCR output containing 15 spelling mistakes for 23 keywords processed by tessetact-OCR

Figure 2.6: OCR of error message text using PhotoOCR [33] and tessetact-OCR [111]

high accuracy, a much different task than extracting arbitrary text from an arbitrary image with high accuracy. The detection of onscreen *keywords* in whole screen desktop screen captures. Approaches to correcting OCR output are discussed including spell-check and dictionary-based methods. Recent work on character-level classification of text using deep learning is discussed.

Chapter 3

AVRA System Overview

This chapter outlines the design of a virtual assistant for personal computer users. First, the system requirements are defined in terms of MOE and MOP. AVRA and another system are compared in this chapter in relation to these MOE and MOP. Next, the basic functionality of the system and implementation considerations are discussed. The function of an assistant like AVRA is to interpret an image of the computer screen and provide action recommendations. The details of sub-component implementation are contained in other chapters. This chapter is more concerned with the high-level description of the system, and how the subcomponents fit together. For example, the essential qualities of Mixed Initiative systems were designed into AVRA, as discussed later on in this Chapter.

3.1 Measures of Effectiveness (MOEs)

MOE defines what the user needs, while MOP defines the user requirements. Rather than defining MOEs only for AVRA, it is helpful to step back and define MOEs for the general problem defined in the thesis statement relating to an assistant with shallow integration. This is accomplished by decomposing the thesis statement into measurable components. Recall that the thesis statement was: "A deep learning artificial intelligence can provide action recommendations related to onscreen messages. These action recommendations can be provided without integration into each individual program executing on the computer. These action recommendations can be provided within a reasonable response time, and can be acted upon with a single mouse click. These recommendations can be personalized to the user by utilizing the user's interaction history and unsupervised

learning." The following is a list of 7 MOEs derived from this thesis statement:

- **MOE1**: Recommendations related to onscreen messages are produced when expected (THRESHOLD=[classification quality is "good" or "excellent" according to the scale from [22]])
- **MOE2**: Adding new recommendations does not require new software integration (THRESHOLD=[$\#integrations/(new\ context) = 0$])
- **MOE3**: Recommendations are provided within a reasonable response time (THRESHOLD=[On desktop with $4800x3600$ pixel resolution, detection to recommendation latency$< 3\ seconds$])
- **MOE4**: Recommendations are seen and executed with a single mouse click by the user (THRESHOLD=YES)
- **MOE5**: Recommendations are personalized to the user (THRESHOLD=YES)
- **MOE6**: Recommendations can be added to the system through supervised learning (THRESHOLD=YES)
- **MOE7**: Recommendations can be added to the system through unsupervised learning (THRESHOLD=YES)

Of the seven user needs captured by these MOEs, four entail achieving a skill (MOE4, MOE5, MOE6, MOE7), one is a limit on the form of the implementation (MOE2), and two MOEs involve a limit on the system requirements (MOE1, MOE3). Note that the level of personalization of the RS is not captured in MOE5.

3.2 Measures of Performance (MOPs)

Switching perspective from the user to the technical requirements, the following MOPs are defined based upon MOE1 and MOE3:

- **MOP1 for MOE3**: Scalability of execution time with many contexts and keywords learned into the system (THRESHOLD=[The latency from detection to recommendation is maintained when many contexts and keywords are trained into the system])

- **MOP2 for MOE1**: Detect multiple onscreen recommendation opportunities at once (THRESHOLD=[80% of possible recommendations are detected and recommended])
- **MOP3 for MOE1**: Detect multiple onscreen recommendation opportunities when recommendation features overlap on the screen (THRESHOLD=YES)

3.3 AVRA System

AVRA is an item-based and content-based/cognitive RS designed to scale to many users as a web service. Collaborative filtering has not yet been implemented. It is a client-server system where the client captures an image of the screen which is then interpreted by a server-side program. The server processes the image by parsing and classifying the text and graphics on the computer screen. The server responds to the client's image submission with a set of 3 personalized action recommendations presented to the user as buttons in a graphical user interface shown in Figure 3.2. The three icon-decorated buttons of the AVRA GUI are loaded with one solution recommendation per button for the user to evaluate.

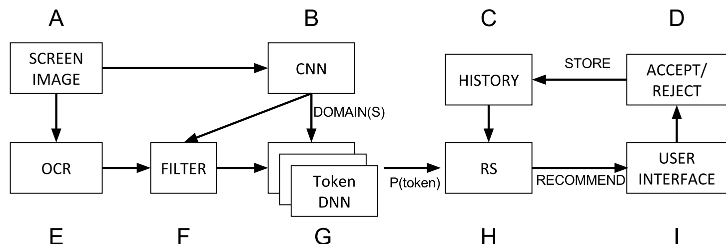


Figure 3.1: AVRA System Overview. See also revised design in Figure 3.3

AVRA is initialized with a corpus of graphical context-detection capabilities (*contexts*), context-specific terms (*keywords*), and action recommendations corresponding to each *keyword*/*context* pair. Figure 3.1 shows how each screen capture is processed by a Convolutional Neural Network (CNN) (Figure 3.1 *B* and Chapter 5) to identify what contexts should be explored for this image. Optical Character Recognition (OCR) (Figure 3.1 *E*) converts the screen image (Figure 3.1 *A*) into a string of text to be processed. The raw text from the OCR system (Figure 3.1 *E*) is cut into many small text segments which are then filtered (Figure 3.1 *F* and Chapter 6), selecting for similarity to the terms within

each context (Figure 3.1 *B*). A Deep Neural Network (DNN) for each activated context (Figure 3.1 *G* and Chapter 7) processes the text output of the OCR tool to detect *keywords* within the text that are associated with recommendations stored in the RS database. The recommendations are filtered (Figure 3.1 *H* and Chapter 8) to identify the top 3 ranked recommendations to be presented in the user interface (Figure 3.1 *I* and Chapter 3.5). The user may accept recommendations by pressing a button in the user interface. Clicking a GUI button causes the recommended action(s) to be executed on the computer (Figure 3.1 *D*) and the user history is also updated (Figure 3.1 *C*).

Consider where the contributions of this work appear in Figure 3.1. Shallow application integration involves the combination of all modules, while the context awareness contribution is encapsulated in the way the CNN is used to extract a context from the screen capture image (Figure 3.1 *B*). Fast text filtering for a given context is achieved in the filter module (Figure 3.1 *F*), and recommendation filtering is accomplished with the module at (Figure 3.1 *H*).

3.4 Client-Server Architecture

Placing the RS on the server rather than the client avoids the need for the deep learning software components to be installed on the client's machine. This thin-client approach facilitates leveraging GP-GPU acceleration on the server-side (in the cloud), which is often unavailable on the client's local machine. As the client computer may not have the power to run the deep learning neural networking code, there is a need to offload the heavy computation to allow for the system to run on a PC with a reasonable response time. Furthermore, this thin-client architecture makes the user profile accessible from multiple computers as a user moves from machine to machine. Therefore, AVRA prototype consists of a client-server framework. The client runs Ubuntu Linux or Microsoft Windows along with a python-based client application. The user interface component utilizes the Tkinter user interface library [1], records information in a sqlite3 file-based database [95], and takes images of the computer screen with cross-platform desktop image capture package pyscreenshot [3].

The server consists of a Virtual Machine (VM) running Ubuntu Linux operating system. The server was initially deployed on a low cost VM at DigitalOcean with 2 CPUs, 2 GB RAM and 40 GB SSDs storage ($10/month USD in March 2016). This proved insufficient in terms of RAM and so the VM was resized to 4GB RAM ($20/month USD in March 2016). This 4GB RAM 2CPU VM was tested with OpenMP and without

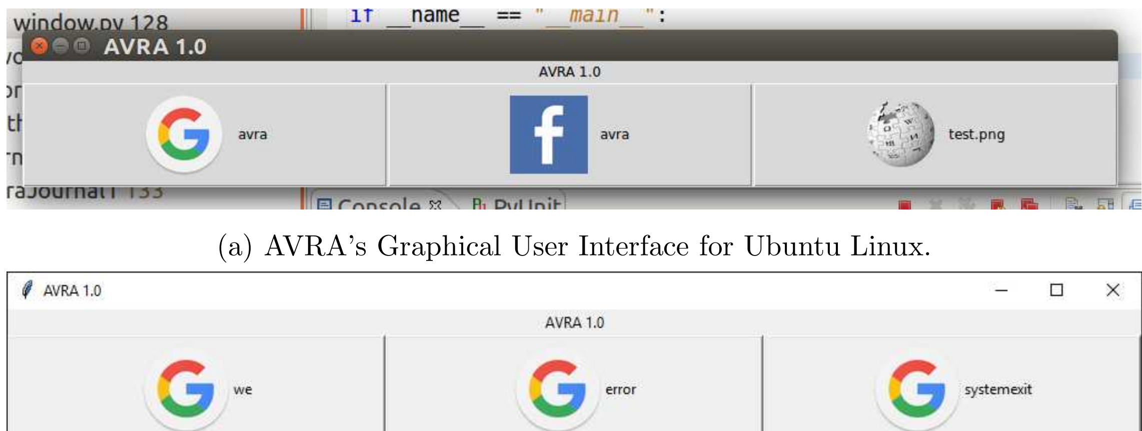
to assess the value of accelerating the theano neural networking code on a low-cost VM (See Chapter 3.7.1) [31] [45]. Ultimately, the speedup achieved using OpenMP was not significant enough to train the neural networks within a reasonable timeframe, and so, for training the DNNs, a GPU-ready VM was needed. To accomplish this, an Amazon Web Services (AWS) instance of type g2.2xlarge and a GRID K520 GPU was configured with an existing AMI containing theano for cuda 6.5 [28] ($440/month USD in March 2016). Execution time observations for these server configurations are presented in Chapter 3.7.

Unless otherwise specified, all results in this work are reported based upon the DigitalOcean VM with 4GB RAM without OpenMP. Supervised learning results were obtained using the AWS GPU-accelerated VM. Installed on the server are docker for image management, werkzeug to act as python application server [233], and backed by an apache2 protocol server for HTTP [19]. Also installed on the server were the theano and tensorflow libraries for neural network modeling, a sqlite3 server-side database, tesseract-OCR for OCR image processing, tmux for detaching programs from sessions, and several asynchronous server jobs written in python and Java implementing the algorithms described in this work [6] [31] [111] [236]. Purely for convenience, some server jobs such as image generation were written in Java, while others such as database management written in python.

3.5 User Interface

This section describes AVRA's GUI module, where the interaction between the human user and RS is accomplished via 3 buttons in a GUI shown in Figure 3.2. These buttons are decorated with meaningful icons making it clear what action recommendations are being offered to the user, and reducing the footprint of the GUI on the screen. Further insight into why the recommendation was made is available as a tooltip when the user's mouse scrolls over a GUI button. The GUI is highly parametric, limiting the scope of the interaction between the RS and the user. The visible computer screen area is the AVRA workspace, while the feedback accepted by AVRA is limited to button clicks.

Following the approach of [115] to offer user actions that can be accepted by the user by pressing one key/button, AVRA uses three onscreen buttons to expose recommendations. As seen in Table 1.1, the virtual assistant field is currently focused on dialog-based interaction systems, and so AVRA was designed to instead explore the interaction focused on visual representations. As discussed in [181], graphics-based interaction with the MI system helps the user to more effectively concentrate on decision making. AVRA's



(a) AVRA's Graphical User Interface for Ubuntu Linux.

(b) AVRA's Graphical User Interface for Microsoft Windows.

Figure 3.2: AVRA's Graphical User Interface

3 button UI design was influenced by similar interface designs including the "Smart Reply" feature of Google Inbox implemented in TensorFlow. It offers up to 3 candidate email message replies as draft responses [52]. Unlike Smart Reply, AVRA's UI needs to specify multiple items per button: an action, a short snippet of text describing the data used by the action, and a longer tooltip message about the recommended action. The representation of these 3 items per button is accomplished using action icons in the GUI rather than action description text, saving real estate in the GUI that is better utilized by the action data description text. The tooltip hides the longer full-text description of the recommended action. The tooltip text feature allows the user to glean additional insight by reading a longer text explanation for the proposed action recommendation. Having compacted the form factor of the GUI, there is a tendency to increase the number of GUI buttons. However, including too many buttons in the GUI would create a paradox of choice where the interface is adding cognitive pressure onto the user by forcing her to think about many possible options, rather than achieving the goal this system was designed for, which is to reduce that cognitive pressure [200].

AVRA must behave predictably, and so the GUI buttons display action recommendations as binary choices, limiting scope of user feedback and limiting scope of AVRA's freedom to act. This parametric interaction makes the user experience significantly more intuitive than a voice command system, because the user does not know with certainty when issuing a voice commend that the computer will comply correctly with the command, whereas AVRA advertises what actions it proposes to perform prior to the user

commanding the action to be performed. Furthermore, the GUI only updates the recommendations when the mouse is stationary and not hovering over the GUI. This way, if the user sees a recommendation in the GUI and begins to move the mouse toward the buttons, the recommendation will not be replaced, annoying the user. Similarly, if the user is pondering a recommendation, perhaps reading the tooltip text while hovering the mouse over a button in the GUI, the action recommendation should not be replaced. This type of confidence building is important in MI system design and is discussed in [255] and [132]. On the topic of reliability, the GUI process executes in a separate thread from the client-server communication process in order to avoid client-server communication from producing lag in the responsiveness of the GUI.

In AVRA, prompts are intentionally monotonous and occur at even multi-second intervals (e.g. button updates every 6 seconds). Ideally the GUI updates should to be as innocuous as updates to the clock in the taskbar. However, the GUI updates are not as innocuous, and so the impact on the user experience of many prompts over time is addressed here.

[92] describes a smart home application for assisting the elderly which involves prompting the user many times. The system learns rules that define when activities normally occur and utilizes these rules to automate prompting. Whereas [92] focuses on waiting to prompt until the notification is needed, AVRA focuses on recommending quickly based upon the current content on the computer screen, or not at all. AVRA does not have a temporal sense when it comes to providing action recommendations. A key takeaway from [92] is that better measures are needed to quantify the risk and benefit of prompting the user when many notifications are called for. Microsoft's BusyBody was one attempt to better understand the interruptability of the user [93]. The goal was to manage notifications coming from many different applications. BusyBody was trained by periodically asking users about their interruptability while recording during each answer the computer's context in terms of what is happening on the computer. Specifically, the system recorded the computer activity history, meeting status, physical location, the time of day, and other factors [93]. After this training phase, the BusyBody model could predict the interruptibility of the user with a cost function which takes the computer's context as an input. Another approach to limiting the damage of repeatedly prompting the user, described in [110], is to limit the prompting in the following ways: based upon the past actions of the user (e.g. once an action is performed, stop offering that action for a number of minutes), based upon the context when activities are typically performed (e.g. offer traffic information just before leaving work, rather than just after arriving

at home), and based upon action sequences (e.g. recommend actions based upon ontological or heirarchical knowledge of action sequences). Note that knowledge of action sequences is different than knowledge of relationships between actions themselves which was discussed in Chapter 2.3 regarding CBF [251] and [136].

RS user interface design is discussed in [160] from the perspective that the user interface is a major driver of user adoption of RS technology, and concludes that GUI design is a factor in the user's trust of the RS recommendations. According to [160], exposing the capability to explore and comprehend recommendations in the GUI can be a negative influence on the user experience. In [160] the virtual assistant "Clippy" from Microsoft Word is presented as an example of good RS design combined with bad GUI design. Clippy was considered by many users to be too intrusive, even though the recommendations provided were typically useful. In contrast, less intrusive features such as spellcheck, autocomplete and autocorrect are now widely adopted in many technologies. Two independent factors influencing GUI design in RS are obviousness of the recommendations and cognitive effort required to use the interface [160]. Cognitive effort in AVRA is reduced by limiting the number of buttons in the GUI, restricting the action recommendation type, and restricting the complexity of the action recommendations. The obviousness of the recommendations in the GUI is enhanced by the graphical icons used to symbolize various tasks. For example, a mail icon next to a person's name (e.g. "Daniel") is more obvious than the sentence "Compose an e-mail to Daniel". The former contains only one word (the name), and can be interpreted at a glance without really reading into the text of the recommendation with full focus.

Infrequently, the AVRA user may find recommendations from AVRA useful and click a button. More commonly, a bottleneck in the user's process such as an onscreen error or interesting term causes the user to look at the AVRA user interface and check if interesting action recommendations are available. This hand-off approach to human-computer interaction minimizes the interference of the tool. However, when the buttons of the AVRA GUI are updated, the change in color and shape of the icons can be distracting. This is particularly true when all 3 buttons are updated simultaneously. On a practical level, the user may elect to hide the user interface when she does not wish to be disturbed by GUI updates.

3.6 MI System Design

The design principles for MI system design were discussed in the prior art and are discussed in this section of the thesis in relation to AVRA. Also, to gauge the value of AVRA to a user, five colleagues with engineering backgrounds were given a demo of AVRA, and then asked to fill out a short anonymous survey.

3.6.1 MI Design Decisions Based Upon Prior Art

(A1) User control and non-interference: The user is always in control of AVRA, and AVRA does not place a high burden on the user in terms of management of the tool itself.

(A2) Recommendations should enhance the current task with timely useful information: AVRA recommends solutions in within a reasonable response time, ensuring that solutions to detected onscreen errors are available should the user choose to seek assistance.

(A3) Understanding what a recommendation signifies and why it is recommended: When the user seeks more information on why a recommendation was made, AVRA provides limited recommendation context with a tooltip explanation.

(A4) Fast visual interpretation of recommendations: The AVRA user interface therefore takes up a small onscreen footprint, and compactly exposes insights to the user.

(A5) Retaining recommendation context: Unlike ADE, AVRA does not provide a natural-language rationale for recommendations. Rather, the history of each recommendation can only be traced back programmatically by querying a database containing timestamped records of all system events. The rationale behind this design decision is to keep the user experience very simple. This is the same design decision made in Smart Reply [52].

(M1) Speed improvement: It is not critical to show that adopting a given solution or system is faster operationally than manually looking up information. This may indeed be the case, but it is not claimed in the thesis statement. Rather, the goal in developing this solution is to have the computer perform cognitive actions on behalf of the user. A speed improvement over the user performing those sequential actions may result from the user skipping steps such as detection of onscreen information, interpretation of the detected information, retrieval of recommendations based upon the information interpretation, ranking in a cognitive process the available recommendations, and acting on the recommendation, perhaps by opening a program and typing.

AVRA can achieve a speed improvement for the user by taking over several sequential actions including detection of onscreen information, interpretation of the detected information, retrieval of recommendations based upon the information interpretation, ranking of the recommendations, and exposing the recommended actions as options that can be accepted with a single click of the mouse. Often the recommended actions would require several mouse clicks to accomplish, and so the last step alone should provide a speed improvement.

(M2) Data wrangling: Extraction of data for the user is accomplished in AVRA at several levels. Wrangling image data into meaningful keywords is one aspect (Chapters 6 and 7), while understanding the context of the screen image is another (Chapter 5). Data wrangling for supervised learning is required to train AVRA.

(M3) Alternative discovery and comparison: AVRA recommends more than one action at the same time. If AVRA has learned how to resolve one keyword such as an error message in several ways, it can recommend several alternatives.

(M4) Parametric interaction: MIVAS' approach to limit the scope of the interaction between the user and the MI system is apparent in AVRA's design. User interface in AVRA is intentionally limited. The AVRA user can either click a button to accept a recommendation or ignore the recommendation. AVRA was designed with the principles of user control and non-interference in mind. Specifically, the RS recommendations can be leveraged or ignored, based on the domain knowledge and decisions taken by the user. If the user does not want to accept the current recommendations then she need not interact with AVRA at all.

(M5) History tracking and exploration: The history of each recommendation can be traced back programmatically by querying a database containing timestamped records of all system events. As mentioned for (A5) above, there is a tradeoff between simple user interface design and exposing in detail the reason for recommendations. AVRA's design falls on the side of user interaction simplification at the expense of a complex explanation to the user about the source of the generated recommendation. However, the reason for the modification is accessible from log files programmatically for research and debugging purposes.

(M6) System agency and adaptation: AVRA demonstrates agency by working in parallel with the user to identify and recommend solutions for onscreen issues. AVRA adapts by adjusting recommendation scoring over time according to the user's past preferences regarding action recommendations.

3.6.2 User Feedback

To gauge the value of AVRA to a user, five colleagues with engineering backgrounds were given a demo of AVRA, and then asked the following 5 questions into an anonymous survey.

- Having observed how AVRA works, do you think AVRA will save time for programming? (YES/NO)
- Would you use this tool (AVRA), or one like it? (YES/NO)
- How often do you use voice search? (Hourly/Daily/Sometimes/Rarely/Never)
- How often do you use text search? (Hourly/Daily/Sometimes/Rarely/Never)
- How often do you search the internet for solutions? (Hourly/Daily/Sometimes/Rarely/Never)

The demo consisted of an example program presented in the Eclipse IDE that throws an error when compiled. The participant was shown how to manually search for the explanation of the error by copying text from the console window and pasting it into a browser-based search engine. Next the participant was shown how AVRA can recommend the same search and execute it with the click of a button. The results of the user survey are presented in Table 3.1. The few individuals surveyed indicated that a tool like AVRA has some utility.

3.7 Performance Evaluation

This Section is on the evaluation of AVRA and a similar state of the art system in relation to the MOE and MOP defined earlier in the chapter. The metrics used to measure performance are noted, and the results of measurements are reported. These results are then discussed. As a result of these performance experiments, the design of AVRA was modified to improve execution time.

3.7.1 Execution Time

Performance in this section is evaluated on the basis of execution time. The requirements MOE3 and MOP1 are that the system react within a reasonable response time after training, providing feedback to the user on the contents of the computer screen.

Table 3.1: User survey questions and responses, after AVRA demonstration

Participant	Do you think AVRA will save time for programming?	Would you use this tool, or one like it?	How often do you use voice search?	How often do you use text search?	How often do you search the internet for solutions?
1	YES	YES	Never	Hourly	Hourly
2	YES	YES	Never	Hourly	Hourly
3	YES	YES	Sometimes	Hourly	Hourly
4	YES	YES	Never	Hourly	Hourly
5	YES	YES	Daily	Hourly	Hourly

Acceleration of the training time for AVRA is discussed. First OpenMP and multicore approaches are reported. Next, GP-GPU acceleration with CUDA is presented.

AVRA is a prototype desktop computer assistant. It is interesting to compare the performance of AVRA against a state of the art system that understands text and images. Whereas Google Home, Cortana and Alexa are focused on speech, Google Now on Tap (GNoT) [74] contains the necessary machinery to make an analogy with AVRA. However, one drawback is that GNoT is only available at this time on mobile phones. Therefore, the measurements reported here were collected on a mobile phone, rather than a desktop computer.

OpenMP and Multicore DNN Training Acceleration

Each DNN in AVRA is trained for 150 epochs as shown in Figure 7.7, and each epoch requires approximately 250 seconds to complete on a low cost VM. This execution time is related to the size of the training and testing sets, and so these figures are different for each DNN. Assuming 50 DNNs with 150 epochs each, and an average epoch execution time of 250 seconds, the training time on a single VM would be: (50 DNNs) x (150 epochs) x (250 seconds) which is approximately 520 hours. This slow training time and similarly slow runtime do not agree with MOP1, which requires that a reasonable response time is achieved at scale. Even if every DNN is trained on a different dedicated VM, the execution time becomes (150 epochs) x (250 seconds) which is approximately 10.5 hours. Reducing the number of epochs would reduce the accuracy of the DNN classification as shown in Figure 7.7. This leaves the epoch execution time as the variable to be reduced.

OpenMP is a shared memory parallel programming API suitable for parallelizing applications on a multiprocessor computer [45]. Theano is compatible with OpenMP and ships with a benchmark for testing the speedup provided by OpenMP [127]. This benchmark was used to generate the data in Table 3.2 which is a generic view of accelerating theano with OpenMP.

The results from theano's benchmark indicate that configuring theano with 2 OpenMP threads is the best option for reducing epoch execution time. Perhaps this result is a product of the VM having 2 CPUs and OpenMP mapping one OpenMP thread to each CPU. However, testing with AVRA's DNN code reported in Table 3.3 indicates that 3 OpenMP threads is best. This disagreement between the generic benchmark and the AVRA code itself is a reminder that often benchmarks are a poor proxy for testing performance improvements in computer architecture [87]. Note that even though a 2x speedup

Table 3.2: Theano's OpenMP benchmark timed with a vector of 200,000 elements

OpenMP Threads	CPU Cores	RAM (GB)	Operation Type	Time without OpenMP (s)	Time with OpenMP (s)	Speedup
1	2	4	Fast	0.000113	0.000099	1.14
2	2	4	Fast	0.000110	0.000066	1.67
3	2	4	Fast	0.000114	0.000137	0.83
4	2	4	Fast	0.000111	0.000128	0.87
1	2	4	Slow	0.006590	0.006091	1.08
2	2	4	Slow	0.006208	0.003060	2.03
3	2	4	Slow	0.006107	0.004127	1.48
4	2	4	Slow	0.006253	0.003738	1.67

Table 3.3: AVRA DNN training time per epoch. The average baseline execution time is calculated as (263+251)/2 which is 257.

OpenMP Threads	CPU Cores	RAM (GB)	Epoch	Training Time (s)	Speedup from Average Baseline
1	2	4	0	263	N/A
1	2	4	1	251	N/A
2	2	4	0	196	1.3
2	2	4	1	220	1.2
3	2	4	0	151	1.7
3	2	4	1	173	1.5

was achieved by using OpenMP, this only reduced the per-DNN execution time from 10.5 hours down to 5.25 hours ($> 150s$ / training epoch), which is still impractically slow. In the next section GP-GPU acceleration is leveraged to further reduce the per-epoch execution time of DNN training.

DNN Training Acceleration using GPU and CUDA

An AWS instance of type g2.2xlarge with a GRID K520 GPU was configured with an existing AMI containing theano for cuda 6.5 [28]. The system was tested with the same DNN code and data used to produce the data in Table 3.3 above, although OpenMP was not enabled.

CUDA is a parallel computing programming model which enables GPU hardware acceleration of computations [169]. The per-epoch execution time was reduced from 151 seconds without the using CUDA and the GPU, to 6.5 seconds when the GPU is used. This represents a 23x speedup relative to the best OpenMP result in Table 3.3. With a 6.5 second epoch execution time, training a DNN can take (6.5 seconds) x (150 epochs) which is approximately 16 minutes and 15 seconds. Note that the epoch execution time is highly variable based upon the size of the training data, and the number of classes trained into the DNN. With OpenMP, multicore, and GPU acceleration, the time required to interpret the computer screen with AVRA did not scale as the number of DNNs increased. With one DNN the latency from changes on the screen to updates in the GUI was approximately 40 seconds. Execution time scaled linearly so that 10 DNNs required 400s to analyze the screen and update the GUI. This challenge to meet MOP1 led to a change in approach, leaving behind the slow but precise DNNs as future work, and instead using the text filter as the main text classifier in AVRA.

Execution Time Measurement for AVRA

The revised design is shown in Figure 3.3, removing the DNNs and connecting the output of the text filter to the RS. The image capturing and OCR in AVRA were modified as well. Instead of capturing the OCR of the whole screen, the fullscreen image was processed into text in slices, with each slice processed in a separate thread. The advantage of this approach was much faster OCR, but the downside was that this approach would miss text sliced at the lines between slices where the image was cut. To counter this, a second set of image slices offset by half of the slice height was also processed by OCR as well. This ensured that no onscreen text was missed by the OCR process. Figure 3.4 shows how a screen can be cut into $2n-1$ slices. Each thread is responsible for extracting text from its image slice and uploading the text to the server. Testing various settings for n yielded that $n = 6$ had the lowest execution time on average for a computer screen of size $4800x3600$ pixels using an Intel Core i7 3.6GHz CPU, 16GB DDR3 RAM, an SSD hard disk, and 2 GeForce GTX 770 GPUs. For 5 trials with fullscreen color images, $n = 6$ (11 image slices) led to an OCR execution time for a given slice of 4.4 ± 1.5 seconds per slice, and an overall OCR execution time of 8.5 ± 1.4 seconds per image. For 5 trials with fullscreen grayscale images, $n = 6$ (11 image slices) led to an execution time of 2.9 ± 0.5 seconds per slice, and an overall execution time of 5.4 ± 0.9 seconds per image. The advantage of using color images was increased CNN precision and recall. To balance MOE3 and MOP1 (reasonable response time of results and scalability) with MOE1 (high

recall) AVRA was set to process color images to maintain the recall level at the expense of execution time.

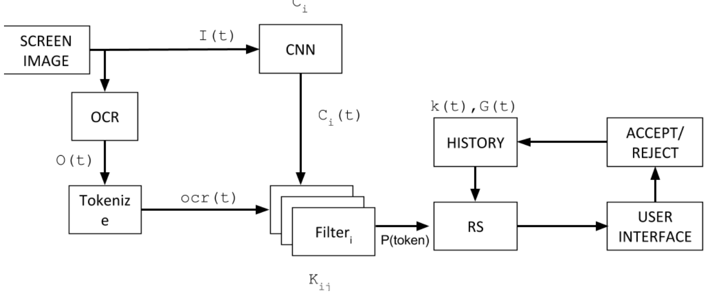


Figure 3.3: Revised AVRA System Overview, after removing DNN to reduce execution time.

Figure 3.4: Revised OCR capture process, splitting fulscreen images into overlapping subtasks in parallel threads to reduce OCR execution time. There are always $2n - 1$ slices produced, and in this case $n = 2$.

Consider an example where AVRA is trained to recognize 11 contexts containing 2,103 *keywords* overall. Tracking the flow of information through the color image processing design described above, and rounding to the nearest second, an image captured at time 0 passed enough information for the CNN to complete context recognition after 4 seconds, and the first context-filtered OCR results emerge from the text filter one second later. The RS results were available 4 seconds later, with another 2 seconds required to update the GUI. The minimum time between information appearing onscreen and a recommendation displaying on the GUI was therefore 11 seconds.

Improving on this design, the RS and filter were moved into the same file, obviating the database communication between these two modules. Tracking the flow of information through the new design and again rounding to the nearest second, an image captured

at time 0 passed enough information for the CNN to complete context recognition after 4 seconds, and the first context-filtered OCR results emerged from the text filter combined with the RS after 2 seconds, with an average of less than one second required to update the GUI. The minimum time between information appearing onscreen and a recommendation displaying on the GUI was therefore 6 seconds.

For a small computer screen size, AVRA's execution time was dramatically faster than the desktop image performance described above. In order to make a fair comparison between GNoT and AVRA, the same end-to-end execution time from detection onscreen to recommendation in a button was measured. To collect data from GNoT and AVRA, the search text "looking for restaurant English text" was inserted into the Google Images API along with the restriction that the dimensions of the image results be exactly $1080x1920$ pixels, corresponding to the dimensions of the mobile phone used for testing. The first 50 results were saved and then transferred to the phone. This dataset of images is hence referred to as $SMALL_IMAGES$.

To simulate the latency of image capture, a region of the desktop $1080x1920$ pixels was captured into a file for each processed image. After this simulated image capture delay, the CNN and OCR processes of AVRA were passed one of the static images from $SMALL_IMAGES$. The total execution time required to fully process all 50 images of $SMALL_IMAGES$ through the OCR, CNN, text filter, RS, and GUI was 176.0 seconds. The average execution time per image was 3.52 ± 1.51 seconds.

To ensure that AVRA can execute relatively quickly when many contexts have been trained into the CNN, ten latency samples were recorded for AVRA CNNs trained with 5, 50, 100, 200, and 400 contexts. Each sample was obtained by recording the CNN output after processing an image with dimensions $4800x3600$ pixels using an Intel Core i7 3.6GHz CPU, 16GB DDR3 RAM, an SSD hard disk, and 2 GeForce GTX 770 GPUs. The results, shown in Figure 3.5, reveal that the execution time grows with the number of added contexts. However, the incremental cost of adding contexts decreases with the number of contexts added, as shown in Figure 3.6. Furthermore, the execution time at 400 contexts remained low, at 3.97 ± 0.10 seconds.

To achieve scalability, AVRA will need to address many scenarios collaboratively with the user. This means learning or being trained to understand many context and associated keywords. Perhaps tens of thousands of context and millions of keywords. To scale in this manner, unsupervised learning can help to fill in gaps in supervised learning as described in Chapter 9. Another key technology that can enable AVRA to scale is collaborative filtering of recommendations.

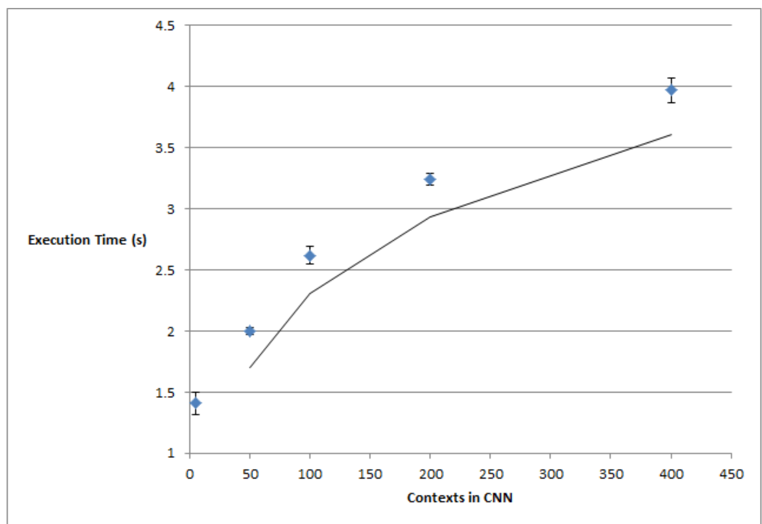


Figure 3.5: CNN latency as number of context increases: The vertical axis in this Figure shows the execution time required to process one $4800x3600$ pixel image using AVRA's CNN. Results for 5, 50, 100, 200, and 400 contexts are shown with error bars indicating the standard deviation for each measurement. A moving average line is added to the figure, revealing with a decreasing slope that the latency cost of adding more contexts is decreasing.

Execution Time Measurement for Google Now on Tap

GNoT delay tactics while it is processing the screenshot include waiting to activate graphics while the user is still holding down the trigger button on phone, showing an animation around the phone frame, showing an animation at the bottom of the screen, and finally, as the card is being displayed by sliding upward, the elements in the card such as images sometimes load after the card is shown onscreen.

The timing data from GNoT was captured with an external camera by recording video of a phone running GNoT and analyzing images. The phone used was a Samsung Galaxy S5 (Model SM-G900W8 running Android 6.0.1). Execution time was measured down to the millisecond by analyzing the video using Media Player Classic [157], and recording the time in milliseconds between holding down the button on the phone (marked on the video by a clicking noise), and On Tap presenting a result on the screen (the first video frame where the Google Now card is displayed and stopped moving). The average execution time required to process an image from $SMALL_IMAGES$ through GNoT and to see the results onscreen was 2.80 ± 0.47 seconds.

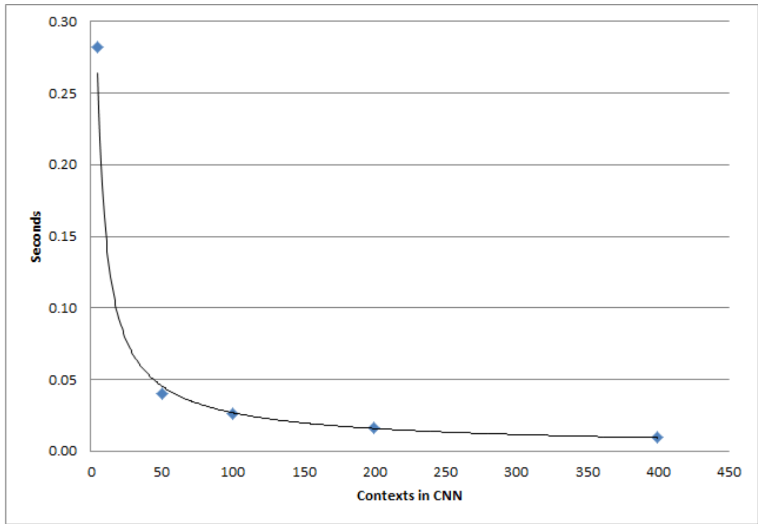


Figure 3.6: CNN latency per context: The vertical axis in this Figure shows the execution time required to process one 4800x3600 pixel image, divided by the number of contexts trained into the CNN. Results for 5, 50, 100, 200, and 400 contexts are shown.

3.7.2 Recall, Precision, Precision-Recall and ROC Curve

AVRA and GNoT are characterized below, observing the recall and precision for each in a confusion table. The procedure for testing each RS is described.

AVRA

It is interesting to observe the capabilities of AVRA regarding overlapping images because a program like AVRA must recognize onscreen items such as program windows that may overlap when observed. AVRA recognizes multiple items in one image. For example, as shown in Figure 3.8, AVRA analyzing an image containing both a terminal window and an Eclipse IDE window leads to the recognition of both by the CNN. AVRA sometimes recognizes both contexts when they are partially occluded as shown in Figure 3.8, but when overlap is high, as in Figure 3.8e, the occluded context (eclipse) was not recognized. However, in some cases such as Figure 3.8f, the occluded context (in this case a terminal window) was recognized by the CNN. When two windows appeared side by side, the CNN usually recognized the context associated with each one as show in Figure 3.7.

To observe the precision and recall for AVRA, it was tested by displaying to the system images on the full screen area one at a time. There were 50 4800x3600 pixel fullscreen desktop images which contained one of two contexts (eclipse or console) and

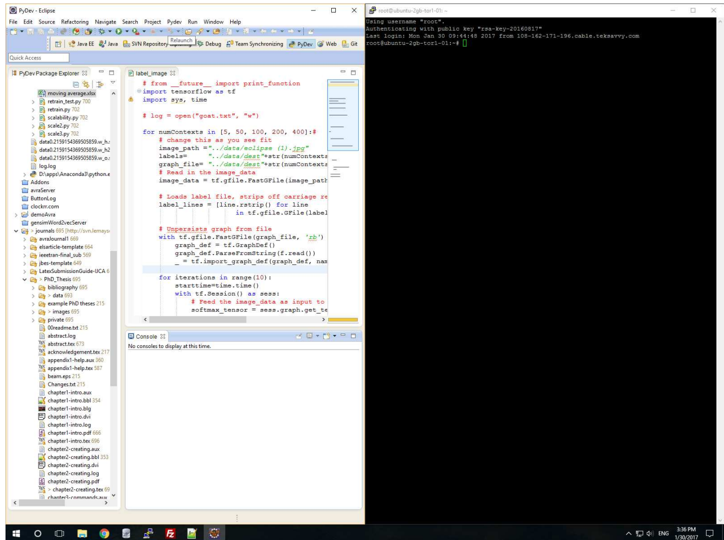
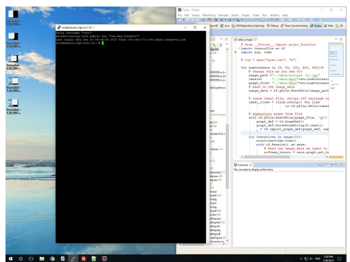


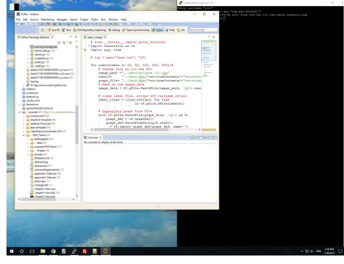
Figure 3.7: AVRA's CNN recognizing content in side by side windows. CNN output score was: eclipse (79.64%), console (12.73%), facebook (3.74%).

one specific *keyword* known to the CNN for that context. The images were collected from real examples, and so they sometimes contained several other *keywords* trained into AVRA for the context in question. For the eclipse context the *keyword* was `import`, and for the console context the *keyword* was `apt-get`. Note that in eclipse the text is blue on a white background, and in a terminal the text is white on a black background. Neither of these scenarios is the typical OCR situation of black text on a white background. These examples were selected because they are a more realistic sample of text on a desktop computer than the obvious case of black text on a white background. To test for TN results 50 additional images containing no relevant context or keyword in them were also presented to AVRA one at a time. These 50 $4800x3600$ images were collected from Google Images using the keywords "my pictures" and the photo type filter was set to "photo". No FP examples were recorded for AVRA during the experiment, as the chance that a context and keyword are incorrectly selected by AVRA at the same time is very small.

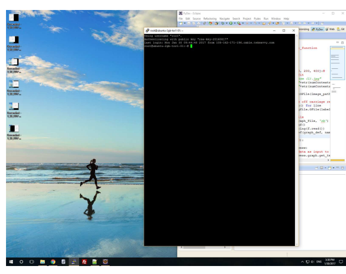
The confusion matrix for the recorded observations is presented in Table 3.4. To generate an ROC curve for the collected AVRA data, a binary classification measure was used. Any sample with a TP results was associated to the ground truth state $[1, 0]$, and any sample with no TP results was associated to the ground truth state $[0, 1]$. The classifier guess was set to $[1, 0]$ in cases where TP or FP was observed. Otherwise



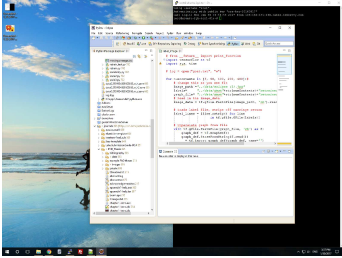
(a) Console occluding Eclipse. CNN output score was: eclipse (51.67%), console (30.51%), desktop (16.30%). The partially occluded Eclipse context was recognized.



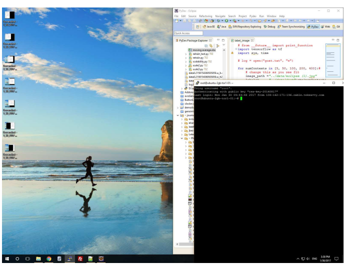
(b) Eclipse occluding Console. CNN output score was: eclipse (76.85%), console (13.26%), desktop (4.97%), gene (2.47%). The partially occluded console context was recognized.



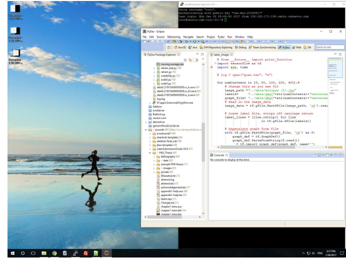
(c) Console occluding Eclipse. CNN output score was: console (13.63%), desktop (84.12%). The partially occluded Eclipse context was missed.



(d) Eclipse occluding Console. CNN output score was: eclipse (53.65%), desktop (28.68%), console (9.12%), facebook (4.84%), gene (2.93%). The partially occluded console context was recognized.



(e) Console occluding Eclipse. CNN output score was: console (6.52%), desktop (93.21%). The partially occluded Eclipse context was missed.



(f) Eclipse occluding Console. CNN output score was: eclipse (64.85%), desktop (14.19%), console (11.75%), facebook (4.94%) gene (3.20%). The partially occluded console context was recognized.

Figure 3.8: AVRA recognition of occluded program windows. The context names are further explained in the next chapter. The CNN used to produce these results was trained to recognize 11 contexts overall.

Table 3.4: Confusion matrix for assessing AVRA after it was trained on 11 classes. The testing data had 25 images per class, with 1 class per testing image, one relevant *keyword* per image, and classification threshold K=1%. Hyperparameter selection and the training procedures for the CNN and DNN are explained in later chapters. Samples either contained context C_A and keyword K_B, or they did not $(\neg C_A \vee \neg K_B)$.

		Predicted		Recall
		$C_A \wedge K_B$	$\neg C_A \vee \neg K_B$	
Actual	$C_A \wedge K_B$	36	14	0.7200
	$\neg C_A \vee \neg K_B$	0	50	
Precision		1.0000		

the classifier guess for the sample was set to the state $[0, 1]$. The ground truth and classifier guesses were used to create the ROC curve of Figure 3.9, and the Precision-Recall curve of Figure 3.10. In 5 of the 50 images containing recommendable information, the recommendation was picked up by the text filter but was eliminated by the RS and 3 or more items the RS could recommend ranked higher than the *keyword* of interest.

Google Now on Tap

GNoT recognizes multiple items in one image. For example, as shown in Figure 3.11, GNoT analyzing an image containing both former presidents Barack Obama and George W. Bush leads to cards for both men being presented. It recognizes both men separately using the images of Figure 3.12, and also when they are partially occluded as shown in Figure 3.13. When overlap is high, as in Figure 3.13e, only one of the contexts (Barack Obama) was recognized. It is interesting to observe the capabilities of GNoT regarding overlapping images because a program like AVRA must recognize onscreen items such as program windows that may overlap when observed.

To observe the precision and recall for GNoT, each image from $SMALL_IMAGES$ was presented on the screen of the phone one at a time. Each time a new image was displayed, GNoT was triggered by holding don the home button. The results were captured as follows: A result is considered TP if the suggested action recommendation was related to the image content, FP if the recommendation was unrelated to content, TN if a recommendation was not expected based upon the image content and no recommendation was provided, and FN if a recommendation was expected based upon the image content but did not appear. Table 3.5 shows where the labels belong in the confusion

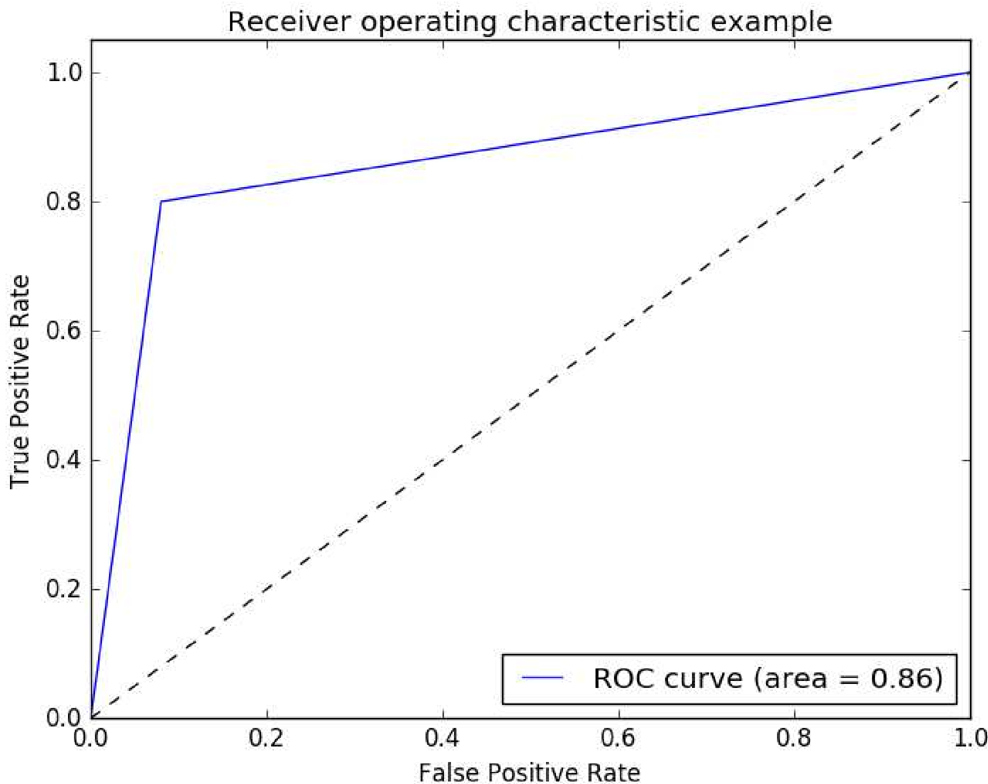


Figure 3.9: ROC curve for AVRA binary classification experiment

matrix, and Table 3.6 presents the observed results collected during the experiment.

In the confusion matrix of Table 3.6, predicted classes are shown in columns, and the actual classes presented to the classifier are rows. Unlike a typical confusion table, the number of tests for any class is not the sum of the integers in a given row. The sum for each row is the number of predictions made by the GNoT for the test images it processed. One test image can result in multiple cards being shown to the user. Also, a Google Now card offering to search for an image of the screen is ignored. An image containing no text was considered a true negative unless a relevant recommendation was produced based upon the image alone, making it a true positive. Or, if an image with no text produced an unrelated recommendation it was considered a false positive. Note that the false negative rate is quite low as only one true positive per test is required in order to not count false negatives for that test.

To generate an ROC curve for the collected GNoT data, a binary classification measure was used. Any sample with one or more TP results was associated to the ground

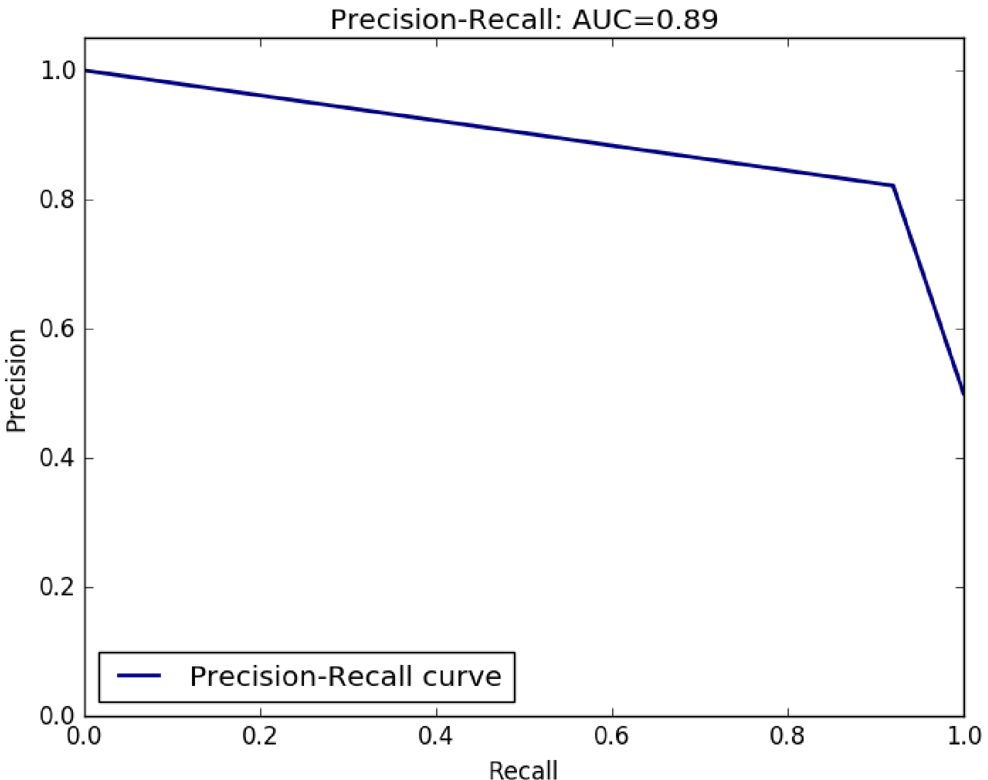


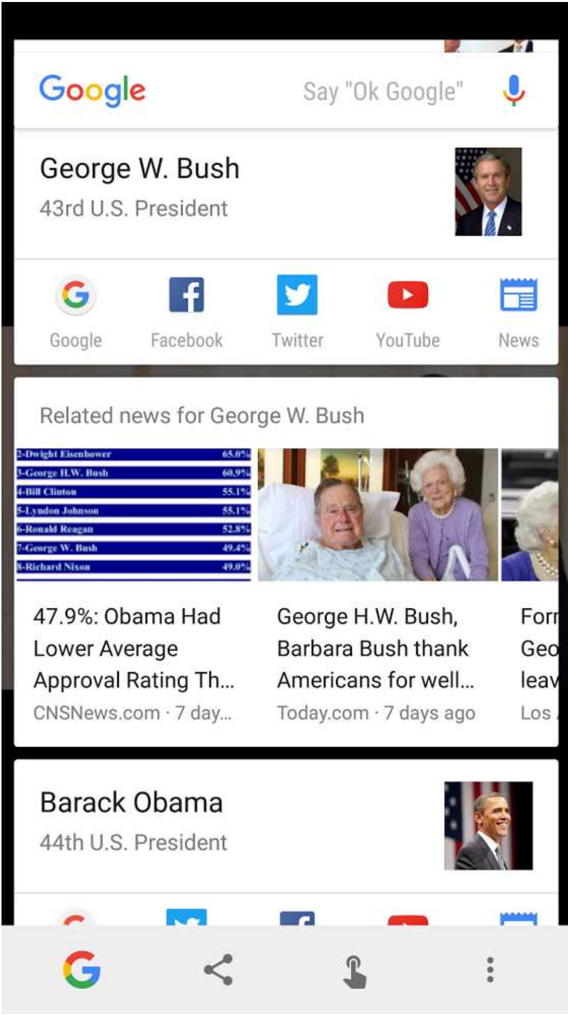
Figure 3.10: Precision-Recall curve for AVRA binary classification experiment

truth state $[1, 0]$, and any sample with no TP results was associated to the ground truth state $[0, 1]$. The classifier guess was set to $[1, 0]$ in cases where TP or FP were greater than 0. Otherwise the classifier guess for the sample was set to the state $[0, 1]$. These 50 samples of ground truth and classifier guesses were used to create the ROC curve of Figure 3.14, and the Precision-Recall curve of Figure 3.15.

3.7.3 Comparing Approaches with MOE and MOP

In comparing GNoT with AVRA, it is important to first lay out the differences between these two tools. First, GNoT runs on a RISC processor in a smartphone, while AVRA runs on a very powerful personal computer with 2 GPUs. Another difference is the input image size. AVRA processes $4800x3600$ images (over 17 million pixels) while GNoT processes $1080x1920$ pixel images (over 2 million pixels). Another important difference is that GNoT is triggered by user action, and so it stalls the user with animations while

(a) Image of former U.S. presidents Barack Obama and George W. Bush analyzed by Google Now on Tap

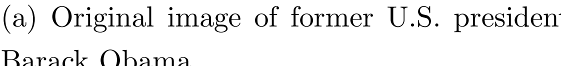


(b) Cards displayed by Google Now on Tap after processing image of former presidents

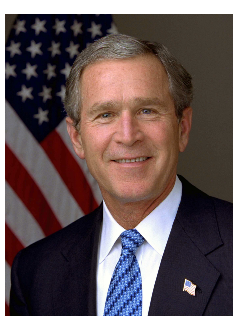
Figure 3.11: Google Now on Tap identifying multiple items in one image

Table 3.5: Illustration of category labels for confusion matrix

		Predicted		Recall
		Related	Unrelated	
Actual	Related	TP	FN	
	Unrelated	FP	TN	
Precision				



(a) Original image of former U.S. president Barack Obama



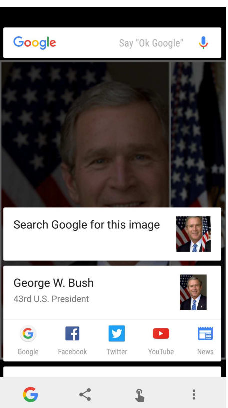
(b) Original image of former U.S. president George W. Bush

Figure 3.12: Original images of former U.S. presidents

Table 3.6: Confusion matrix for a Google Now on Tap experiment

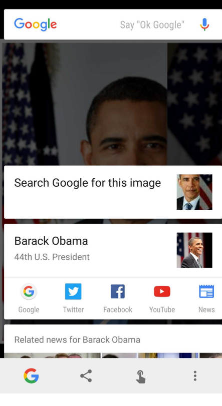
<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Predicted</td><td rowspan="2">Recall</td></tr>
<tr><td>Related</td><td>Unrelated</td></tr>
<tr><td rowspan="2">Actual</td><td>Related</td><td>66</td><td>7</td><td>0.9041</td></tr>
<tr><td>Unrelated</td><td>6</td><td>15</td><td></td></tr>
<tr><td colspan="2">Precision</td><td>0.9167</td><td></td><td></td></tr>
</table>

(a) Partially occluded image of former U.S. president George W. Bush.



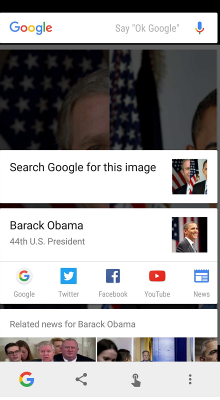
(b) Google Now on Tap produces a card for George W. Bush during partial occlusion.

(c) Partially occluded image of former U.S. president Barack Obama.



(d) Google Now on Tap produces a card for Barack Obama during partial occlusion.

(e) 50% occluded image of former U.S. presidents Barack Obama and George W. Bush.



(f) Google Now on Tap produces a card for Barack Obama but not for George W. Bush during 50% occlusion of both images.

Figure 3.13: Google Now on Tap recognition of occluded former U.S. presidents

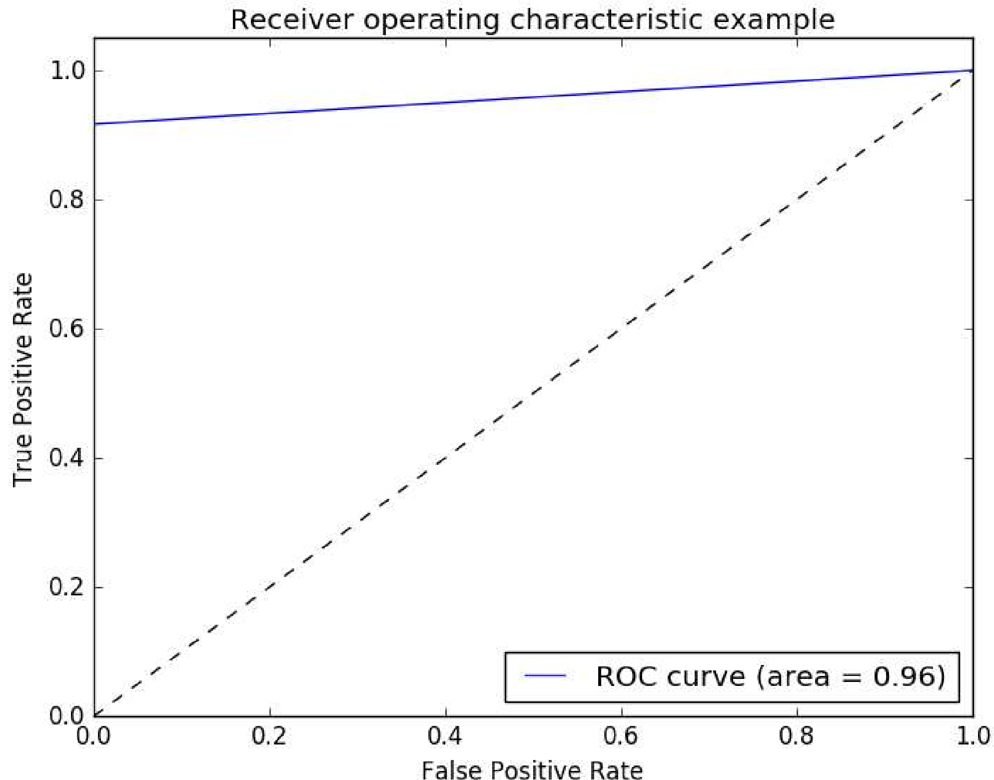


Figure 3.14: ROC curve for Google Now on Tap binary classification experiment

processing. AVRA instead runs constantly in the background, and so processing delays are somewhat hidden from the user experience. Another point to consider is that the goal for AVRA is to reduce cognitive pressure on the user by exposing suggestions in the form of action recommendations, while GNoT is an on-demand search tool that is meant to intervene only when prompted by the user. Another important difference is that AVRA is a specialized tool, while GNoT is a general purpose tool. That is to say, AVRA watches the user and offers to replay the exact actions it observed, and so it has a narrow skill set focused on onscreen words. GNoT has a competing vision for helping the user, where general purpose information such as location-based assistance (e.g. nearby restaurant) and timely information (e.g. local sports event) are combined with image recognition capabilities (e.g. label an image of a celebrity that contains no text). In stark contrast, AVRA does not understand location, world events, or acting on images alone (unless programmed to do so as in the case of the desktop triggering offers to launch frequently visited websites). Finally, GNoT was tested against a wide variety of images containing

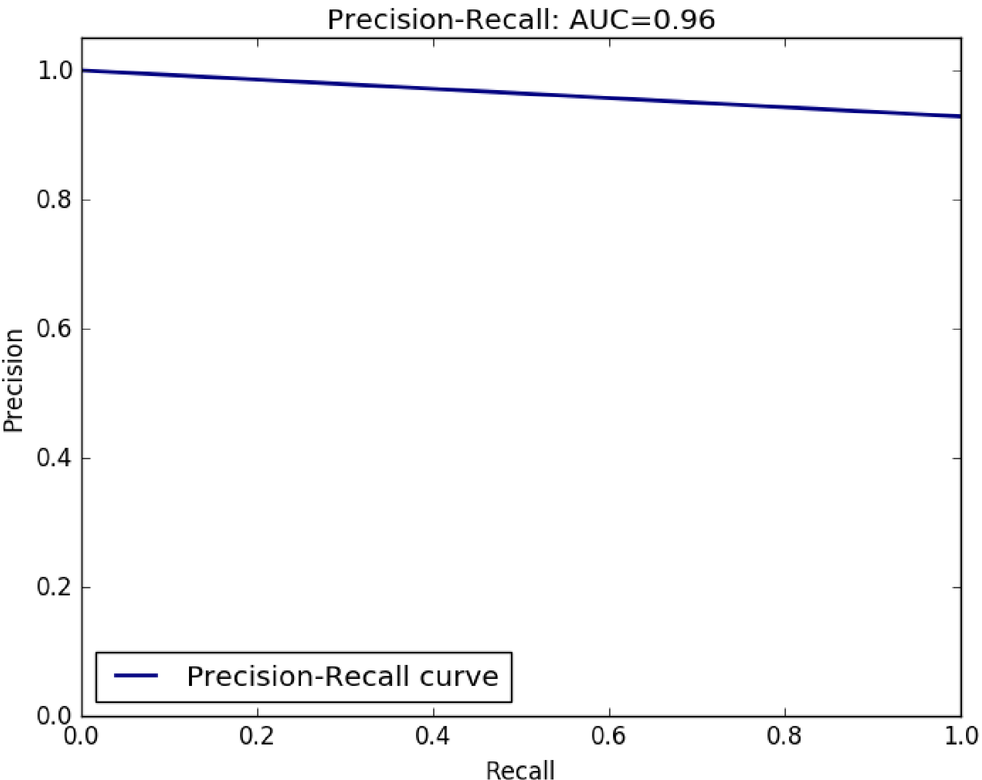


Figure 3.15: Precision-Recall curve for Google Now on Tap binary classification experiment

contexts from many topics. In contrast, AVRA was tested only against a narrow set of contexts. This difference in the difficulty of the testing should be considered when comparing these two tools. AVRA is at this stage a research tool, and is not ready to classify any and all text and image input, whereas GNoT is trained on a large number of topics, and so it is easier to test with. Finally, GNoT runs on the Android operating system, which does not show side by side windows, and therefore GNoT need not detect the graphical features and text contents of multiple application windows. AVRA faces a more complicated situation where multiple applications can appear onscreen together.

For this thesis, the utility of comparing AVRA and GNoT is captured by the MOE and MOP. Indicators for each MOE and MOP are color coded as red, yellow, or green in Table 3.7. Red indicates non-compliance, yellow indicates non-compliance that may be overcome with further effort, and green indicates compliance between the requirement and measurement.

Starting with MOE1 at the first row of Table 3.7, and assessing the ROC area under the curve with qualitative labels as described in [22], GNoT performance was excellent (Figure 3.14), and AVRA's performance was good (Figure 3.9). MOE1 stated that recommendations related to onscreen messages are produced when expected with a classification quality that is "good" or "excellent". Comparing the precision-recall curves of AVRA and GNoT, both curves had high precision and recall, other than the point where AVRA began to lose precision when recall was set above 90%. As described in Chapter 5, AVRA's CNN was designed to focus on maximizing recall rather than precision.

MOE2 stated that adding/learning new recommendations should not require new software integrations to be programmed. For MOE2, AVRA did not require integration into many programs, and new contexts trained into the system did not require new programming integration. Instead AVRA is trained (supervised or unsupervised) with training data. GNoT is closed source, and was likely developed using Google's TensorFlow which can run directly on the phone's hardware. According to GNoT product manager Aneto Okonkwo, new features were added in 2016 to enhance the user experience including selecting specific words to pass to the analysis engine, improved search by image, and searching by image using the camera [18]. GNoT likely learns to recognize text and images using labeled data with supervised learning, using the same Google code base as the Google Vision API and Photo OCR [77] [33].

MOE3 stated that for large images, recommendations should be provided within a reasonable response time. For MOE3, the latency of AVRA was found to be 2.5x longer for larger images than smaller ones. Assuming the same ratio for GNoT, one can estimate an execution time for fullscreen desktop images would take 7 seconds ($2.5 * 2.8 = 7.0$). This latency can be reduced with further effort, simply by using a newer generation GPU, and so both for AVRA and for GNoT the indicator is set to yellow. Adding some detail, the VOLTA GPU architecture performs deep learning computations dramatically faster than the older GeForce GTX 770 GPUs that were used in this work [168]. Similarly the Tensor Processing Unit (TPU) 2.0 is also expected to provide massive performance improvements over the results recorded here [72].

MOE4 stated that the user should be able to view and act on recommendations with a single mouse click. For MOE4, AVRA executes action recommendations with one click, whereas GNoT requires the user to first initiate it and then with one click (and perhaps a swipe to see additional cards) a suggestion can be opened.

MOE5 states that recommendations should be personalized to the user. For MOE5, AVRA provides personalization of recommendations based upon the user's interaction

with AVRA recommendations, while GNoT performs personalization using location and user profile information.

MOE6 stated that supervised learning should be able to train new recommendations into the system. Supervised learning is a standard approach to training image classifiers at Google, and so one can comfortably assess that GNoT meets MOE6. As described in Chapter 9.2 AVRA too can be trained using supervised learning.

MOE7 stated that unsupervised learning should be able to train new recommendations into the system. It is unclear whether GNoT contains unsupervised learning capabilities.

MOP1 stated that the response time with many contexts and keywords trained into the system should remain reasonable. For MOP1, GNoT has already been trained on many topics, and continues to deliver results quickly. It therefore passes the scalability test. AVRA's CNN latency was relatively small at 3.97 ± 0.10 for 400 contexts, and the CNN latency when adding additional contexts was observed to be decreasing.

For MOP2, both AVRA and GNoT could recognize when multiple items onscreen could lead to different recommendations, and provide several of them. AVRA can provide up to 3 recommendations, while GNoT can provide many recommendations as an ordered list or Google Now cards. One downside of AVRA's limited number of outputs is that when many recommendations are possible, only 3 are presented to the user, and the rest cannot be seen by the user.

MOP3 stated that multiple onscreen recommendation opportunities should be detected and recommended when their features overlap on the screen. Both AVRA and GNoT were observed to have the ability to sometimes detect overlapping onscreen items.

The outcome of the comparison between AVRA and GNoT is that both systems are in line with the thesis statement, other than MOE7 regarding unsupervised learning. With additional effort, both systems could be accelerated to have a latency less that 3 seconds (MOE3 and MOP1), and GNoT could be set to act passively instead of being triggered by a button, resolving MOE4.

3.8 Chapter Summary

This chapter presented the overall design of AVRA, a virtual assistant for personal computer users. Measures of performance and measures of effectiveness were defined, and the design of AVRA's Mixed Initiative qualities was discussed. The performance of AVRA and a similar closed source commercial tool were recorded, and these two tools were com-

Table 3.7: Comparing approaches with MOE and MOP

	Note	AVRA Indicator	Google Now on Tap Indicator
MOE1	*Quality* $\geq$ *good*	Good	Excellent
MOE2	*Set #Integrations*	Success	Assumed True
MOE3	*Latency* $<$ 3 *s*	3.52 ± 1.51 s for small image 8.5 ± 1.4 s for large image	2.80 ± 0.47 s for small image Estimated 7.0 s for large image
MOE4	1 *click actions*	Success	2 clicks
MOE5	*Personalized*	Based on GUI clicks and evicts	Location and events
MOE6	*Supervised learning*	Success	Assumed True
MOE7	*Unsupervised learning*	Success	Unknown
MOP1	*Latency* $<$ 3 *s at scale*	3.97 ± 0.10 s CNN latency, 400 contexts & large image	3.52 ± 1.51 s for small image Estimated 7.0 s for large image
MOP2	*Multiple context recommendation*	Success	Success
MOP3	*Detect overlapping features*	Success	Success

pared in the context of the aforementioned requirements. The thesis statement claimed that a deep learning artificial intelligence can provide action recommendations related to onscreen messages. This chapter explained at a high level how action recommendations can be provided within a reasonable response time, and how these recommendations can be acted upon with a single mouse click.

Thus far the outlines of this system have been described, but many details are yet to be elaborated. The CNN for pattern recognition within images, accelerating text processing, and the DNN for text recognition are described in the following chapters. The details behind recommendation personalization and unsupervised learning will also be covered in subsequent chapters. The next chapter takes a step back from the implementation details and provides the reader with a series of examples where AVRA can be applied.

Chapter 4

Supervised Learning Use Cases

In this chapter, several use cases are presented which motivate this work by demonstrating applications of AVRA after application-specific supervised learning. These use cases intentionally span very unrelated topics in order to demonstrate the versatility of an "seeing" desktop assistant. The first use case involves recognizing onscreen references to decisions by the Supreme Court of Canada (Section 4.1) in order to recommend further detail to the reader. The second use case involves recognizing onscreen references to gene family names (Section 4.2), with the goal to provide further detail to the user regarding the mentioned gene family. A third use case involves recognizing error messages in the Eclipse IDE (Section 4.3) in order to assist a programmer to explore solutions to these onscreen errors. The fourth use case involves detecting the presence of the computer's desktop (Section 4.4) in order to recommend programs or websites that the user may want to visit. The fifth use case involves detecting the presence of a terminal window in order to recommend additional information on command usage (Section 4.5). The final use case involves detecting that the user is browsing social media, and offering to the user to compose an email to any of her friends when the name of that friend is seen on the computer screen (Section 4.6).

In describing each of the use cases mentioned above, the description begins with an overview of the use case including a specific example, and then proceeds to list the name used as a context identifier (e.g. "legal"). Next, the image data used to train AVRA is discussed, followed by a discussion of the *keyword* information and recommendation data used to train the use-case specific neural networks. Each use case description concludes with a report on some of the problems encountered during development, and the data sources used to build up the use case.

It is important to note here that these use cases are all active in the same system (AVRA) at the same time. As discussed in previous chapters, it is the onscreen content that drives AVRA to differentiate between contexts, and recommend context-relevant recommendations.

The CNN for the use cases below was trained on 9654 images in total, covering 11 different contexts.

The two overall challenges in developing these use cases were poor classification of *keywords* with very short length (e.g. the terminal command "ls"), and sometimes low context detection confidence (e.g. 2% confidence in the correct class). These cases were rare but noticeable. Perhaps the short *keyword* recognition could be resolved by modifying the DNN input filter hyperparameters. The low confidence context detection cases may be mitigated by collecting additional image data for context training. One final problem was making the distinction between similar contexts. AVRA had trouble differentiating between legal documents and research papers, as they look similar from a distance.

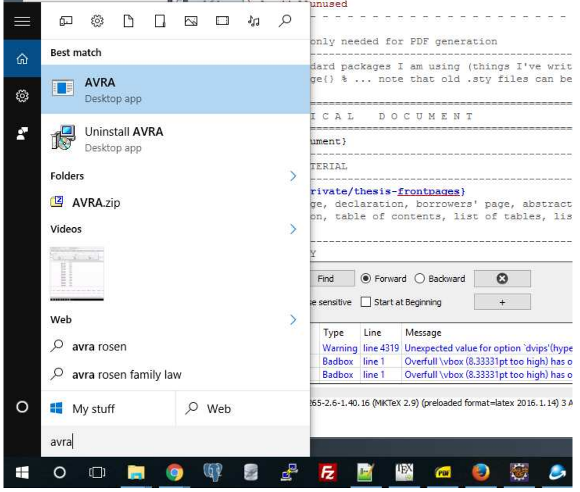
4.1 Supreme Court of Canada

The goal in this use case is to assist the user to find the full text of a particular judgment by the Supreme Court of Canada (SCC) when it is cited in another case. This action should only take place when AVRA detects a context related to reading legal articles. Tools exist to insert hyperlinks of cited cases into websites [39]. However, these tools do not recognize in within a reasonable response time, as AVRA does, when a document that could be annotated with a link appears onscreen.

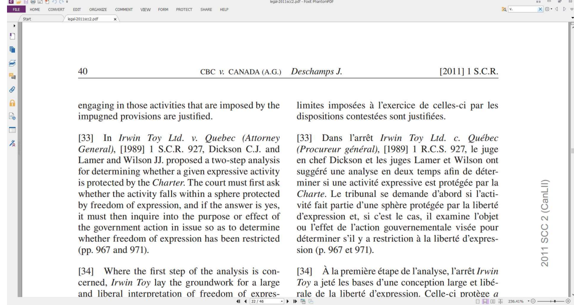
The first step in the use case is to run AVRA's desktop software as shown in Figure 4.1a. Next, the user opens a PDF document containing a decision from the SCC which cites another SCC decision as shown in Figure 4.1b. In the next step, AVRA detects the reference and recommends to the user in an AVRA button to explore the references case further as shown in Figure 4.1c. Finally, clicking on this button launches a browser window containing the full text of the cited case and links to other useful information Figure 4.1d.

Context name: legal

CNN training data: 56 decisions by the SCC chosen at random from the corpus of all SCC decisions. The PDF files for all of the selected decisions were merged into one PDF file with 2946 pages. This file was then converted into 2946 images for processing



(a) Launching AVRA



40 CBC *v.* CANADA (A.G.) *Deschamps J.* [2011] 1 S.C.R.

engaging in those activities that are imposed by the impugned provisions are justified.

[33] In *Irwin Toy Ltd. v. Quebec (Attorney General)*, [1989] 1 S.C.R. 927, Dickson C.J. and Lamer and Wilson JJ. proposed a two-step analysis for determining whether a given expressive activity is protected by the *Charter*. The court must first ask whether the activity falls within a sphere protected by freedom of expression, and if the answer is yes, it must then inquire into the purpose or effect of the government action in issue so as to determine whether freedom of expression has been restricted (pp. 967 and 971).

[34] Where the first step of the analysis is concerned, *Irwin Toy* lay the groundwork for a large and liberal interpretation of freedom of expres-

limites imposées à l'exercice de celles-ci par les dispositions contestées sont justifiées.

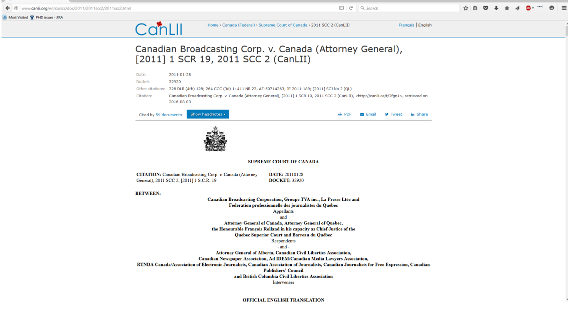
[33] Dans l'arrêt *Irwin Toy Ltd. c. Québec (Procureur général)*, [1989] 1 R.C.S. 927, le juge en chef Dickson et les juges Lamer et Wilson ont suggéré une analyse en deux temps afin de déterminer si une activité expressive est protégée par la *Charte*. Le tribunal se demande d'abord si l'activité fait partie d'une sphère protégée par la liberté d'expression et, si c'est le cas, il examine l'objet ou l'effet de l'action gouvernementale visée pour déterminer s'il y a restriction à la liberté d'expression (p. 967 et 971).

[34] À la première étape de l'analyse, l'arrêt *Irwin Toy* a jeté les bases d'une conception large et libérale de la liberté d'expression. Celle-ci protège *a*

2011 SCC 2 (CanLII)

(b) Context and *keyword* detection [226]

(c) Action recommendation



(d) Recommended action launched [40]

Figure 4.1: Use case illustration: AVRA assisting with legal document reading

by the CNN.

DNN ***keywords*****:** Specific case names in the format “Roe v. Wade”. A text processing algorithm was used to obtain *keywords* used to construct the DNN input text.

Recommendation when DNN ***keyword*** **recognized:** Open a website in a browser where the content is the full text of the cited decision and additional helpful tools and links. A text processing algorithm was used to obtain the hyperlinks required to construct the action recommendations.

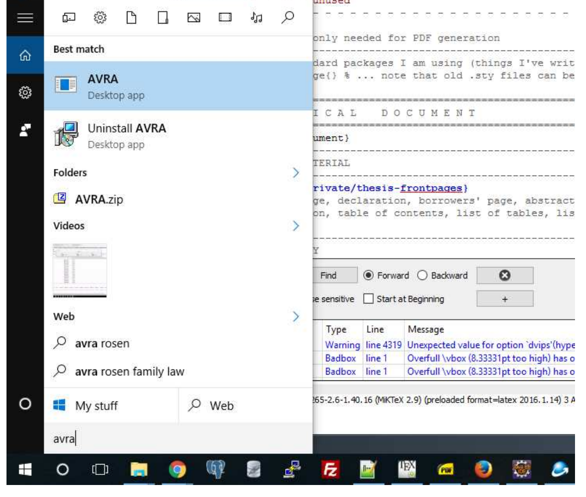
Problems encountered: An initial idea was to encode docket numbers (e.g. 31459) as the DNN *keywords* because docket numbers were easy to extract and encode. However, it emerged in further investigation of the legal literature that the docket number for SCC cases are not commonly referenced in the legal literature in a way that is useful to the reader. A more useful approach was to look for onscreen citations by name (e.g. Roe v. Wade). This caused a problem where multiple cases have the same citation, and so the RS opens to the case search page rather than a particular case. Case name split across multiple lines confuses the text processing algorithm, and therefore it misses the *keyword*. Only implemented for a small number of *keywords* as data extraction is tedious and should be further automated with CANLII OpenSearch API functionality [40]. Another problem was that the CNN recognition of SCC document images had low confidence scores. Perhaps this was because the page format of SCC documents is not consistent through time. It has changed over time as, for example, an 1877 SCC decision has single column format while other decisions from the 1980s and 1990s have orienting marks along the middle of the page, and documents from the 2000s have numbering along left and right outside margins of the text. Fonts have also changed over time, and image resolutions is lower for older scanned files than it is for modern digital files.

Datasource: [40]

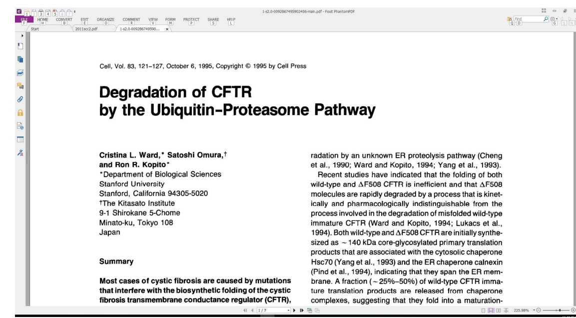
4.2 Genetic Research

In this use case the user is reading research material related to genetic research, and AVRA assists the user by offering the user to explore the relationship between a gene name recognized on the computer screen, its function, and related genes. Specifically, the cystic fibrosis gene CFTR is used in this example.

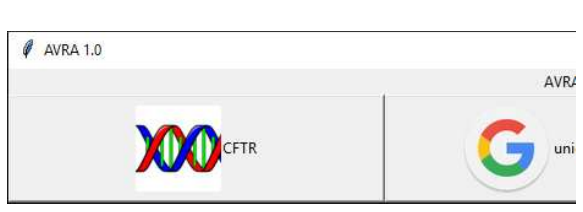
The first step in the use case is to run AVRA’s desktop software as shown in Figure 4.2a. Next, the user opens a PDF document containing the genetic research material as shown in Figure 4.2b. In the research material the CFTR gene is mentioned. AVRA



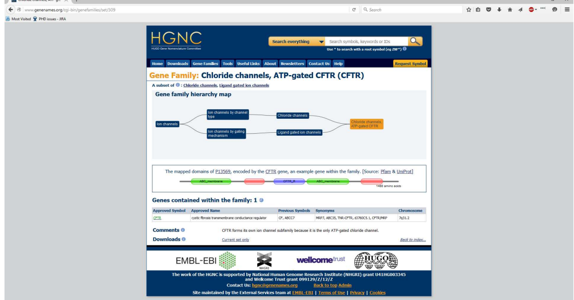
(a) Launching AVRA



(b) Context and *keyword* detection [249]



(c) Action recommendation



(d) Recommended action launched [80]

Figure 4.2: Use case illustration: AVRA assisting with genetics research

then detects the reference to the CFTR gene family and recommends to the user in an AVRA button to explore the references gene family as shown in Figure 4.2c. Finally, clicking on this button launches a browser window containing a detailed review of the Cystic Fibrosis Transmembrane conductance Regulator (CFTR) and links to other useful information as shown in Figure 4.2d.

Context name: gene

CNN training data: The CNN was trained on images of research papers related to the gene families in question. This was accomplished by following links constructed using the gene family names and the keyword "gene" to find PDF files from Google Scholar [76] where the top results with an available PDF file in a relevant field (genetics, cell biology, cancer research, etc.) was downloaded. In all, 87 research papers were collected,

and then merged into a single 980 page PDF file. This file was then converted into 980 images for training the CNN. Note that the genetic research content may be viewed in a variety of programs and browsers as AVRA has been trained on images of the research data rather than images of the PDF reader program.

DNN *keywords*: 93 gene family names (e.g. ZCCHC) and their approved names (e.g. lysocardiolipin acyltransferases). 1365 *keywords* in total.

Recommendation when DNN *keyword* recognized: When a gene family name or specific gene name is recognized, AVRA can offer to open a browser window to a website describing the related gene family.

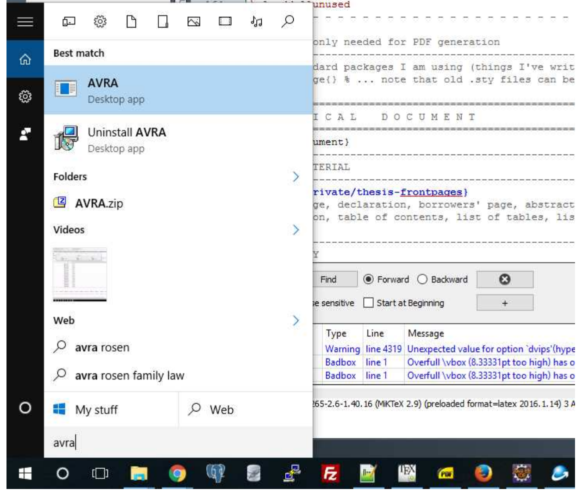
Problems encountered: *Keywords* that span across more than one line in the text of an onscreen research paper were often not recognized by the DNN. Also, some gene family names such as "MARCH" and "XP" were too generic to find meaningful results when searching for research publications in an automated search process. In collecting examples for CNN training, these cases were resolved by simple passing over those gene names that did not result in easily accessible relevant research publications. Another problem encountered was the limit on the number of recommendations. AVRA can only recommend 3 actions to the user, and often this means that many opportunities to explore are not exposed to the user. Specifically, in the case where the AVRA identifies multiple gene names on the screen, the RS narrows down which *keywords* should lead to recommended actions (based upon the confidence in the *keyword* detection, the confidence that the gene context is on the screen, and the past history of the user's interaction with AVRA).

Datasource: [80], [76]

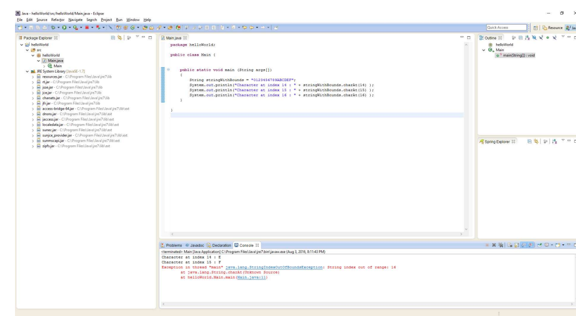
4.3 Eclipse IDE

In this use case, AVRA assists the user as she investigates solutions to error messages appearing in the Eclipse console during a programming session.

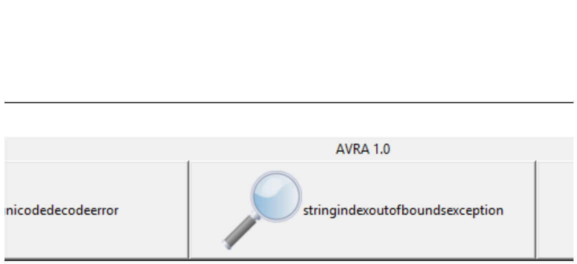
The first step in the use case is to run AVRA's desktop software as shown in Figure 4.3a. Next, the user opens the Eclipse IDE and begins writing a program. During the iterative programming, compiling, and debugging of the programming task, the console outputs an error message as shown in Figure 4.3b. AVRA then detects the error message `StringIndexOutOfBoundsException` and recommends to the user in an AVRA button to explore solutions to this error as show in Figure 4.3c. Finally, clicking on this button launches a browser window containing information relevant to the detected error message



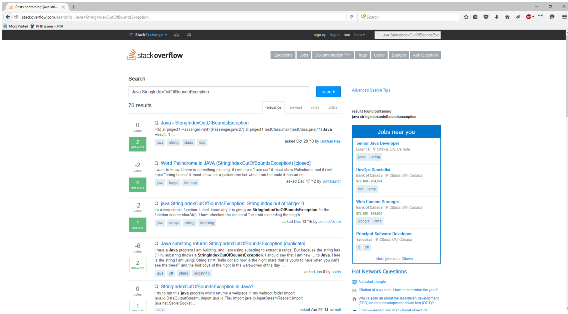
(a) Launching AVRA



(b) Context and *keyword* detection



(c) Action recommendation



(d) Recommended action launched [223]

Figure 4.3: Use case illustration: AVRA assisting with a programming task

and links to other useful information as show in Figure 4.3d.

Context name: eclipse

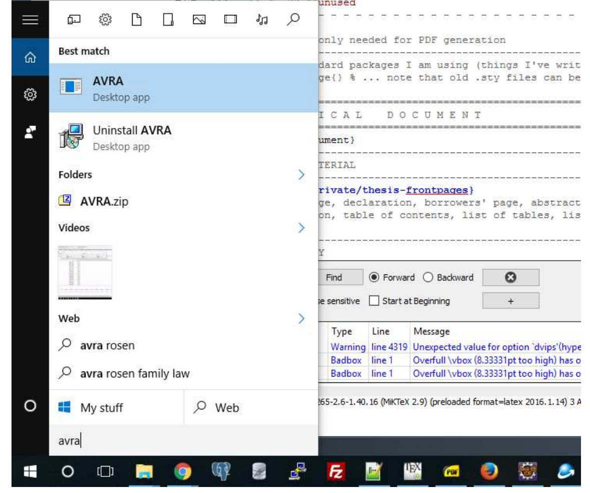
CNN training data: 190 images of eclipse programming.

DNN ***keywords*****:** 299 common Java and Python error messages

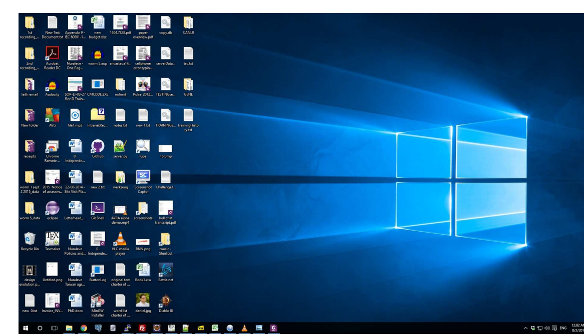
Recommendation when DNN ***keyword*** **recognized:** stackoverflow.com search results for the detected *keyword* + language e.g. "Python Exception"

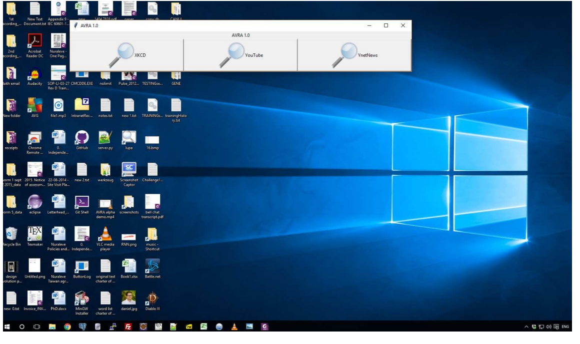
Problems encountered: Short error messages confuse the pre-DNN filtering algorithm. Also, the `exception` exception is a substring of most other exceptions such as `baseexception`, and so it shows up when it should not. Not resolved in this work but using ontologies in the RS should resolve this later on.

Datasources: [222], [223], [81]



(a) Launching AVRA


(b) Context detection


(c) Action recommendation

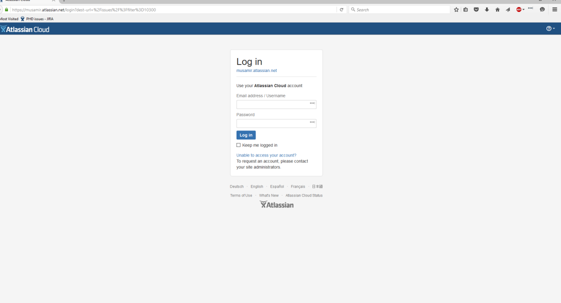

(d) Recommended action launched

Figure 4.4: Use case illustration: AVRA assisting the user to start a new task

4.4 Computer Desktop

In this use case, AVRA assists the user as she decides to start a new task. The first step in the use case is to run AVRA's desktop software as shown in Figure 4.4a. Next, the shows the desktop and waits for AVRA to detect that the desktop is presented as shown in Figure 4.4b. Having detected the desktop context, AVRA proceeds to recommend to the user in an AVRA button to launch websites she frequently visits as shown in Figure 4.4c. Finally, clicking on one of the three AVRA GUI buttons launches a browser window containing the website of the clicked recommendation 4.4d.

Context name: desktop

CNN training data: 300 images of Microsoft Windows Desktop (various back-

grounds, icons, OS versions)

DNN *keywords***:** None. This is a use case where AVRA can proceed directly to the recommendation without looking for *keywords*.

Recommendation when context recognized: Top 3 most frequently visited websites

Problems encountered: The recommendations are currently based upon parsing the chrome browser history. This is a violation of the shallow integration approach that should be avoided in future versions of AVRA. In an ideal scenario, AVRA would keep track of the user's browser history through a less invasive interaction such as keystroke logging and/or parsing URL text.

Datasource: [141]

4.5 Console Programming

In this use case presence of a command in a terminal window triggers AVRA to recommend additional information on the usage of that command.

The first step in the use case is to run AVRA's desktop software as shown in Figure 4.5a. Next, the user opens a terminal window, logs into a server, and types a command with incorrect parameters (`ls -lj`). An error message is printed to the terminal as a result of this incorrect command as shown in Figure 4.5b. AVRA then detects the error message and recommends to the user in an AVRA button to explore solutions to this error by learning more about the specific command as shown in Figure 4.5c. Finally, clicking on this button launches a browser window containing information relevant to the command that triggered the detected error message and links to other useful information as shown in Figure 4.5d.

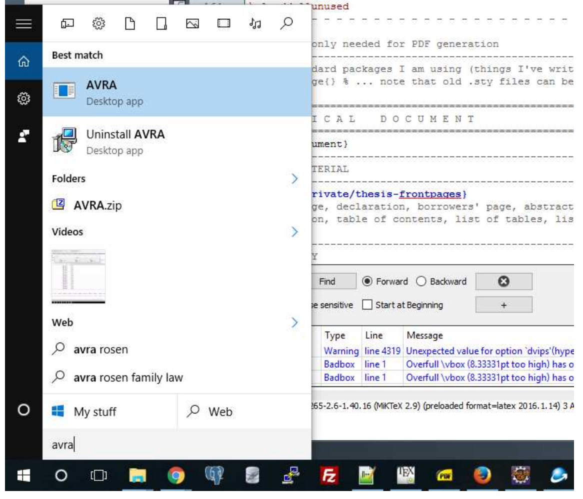
Context name: console

CNN training data: 656 images of command line interfaces

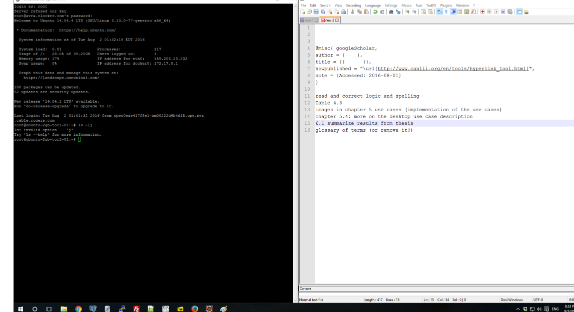
DNN *keywords***:** 258 commom bash terminal commands

Recommendation when DNN *keyword* **recognized:** Launch browser window containing recipe for how to use that command, and an explanation of what the command does (e.g. MAN pages)

Problems encountered: Consoles configured with white backgrounds or dramatic color schemes reduces the confidence of the CNN. Ignored these cases. The DNN does not understand that lower lines in a console are more important than higher lines. Not addressed in this work.

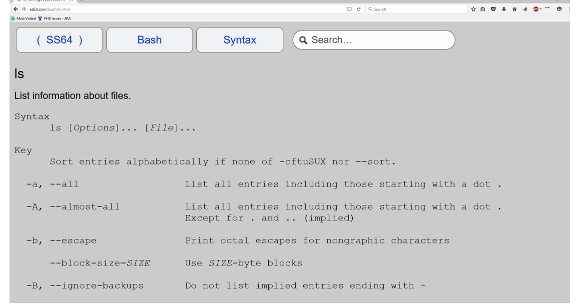


(a) Launching AVRA



(b) Context and *keyword* detection

(c) Action recommendation



(d) Recommended action launched

Figure 4.5: Use case illustration: AVRA assisting the user with console commands

Datasource: [5]

4.6 Browsing Social Media

In this social media use case the user is browsing social media website Facebook. AVRA assists the user by offering to the user to compose an email to any of her friends when the name of that friend is seen on the computer screen.

The first step in the use case is to run AVRA's desktop software as shown in Figure 4.6a. Next, the user opens a browser window to facebook.com and logs into her account. The user's facebook friend Liora appears in an item in the user's facebook feed as shown in Figure 4.6b. AVRA then detects the presence of this friend's name and recommends to the user in an AVRA button to compose an email draft to Liora as shown in Figure 4.6c. Finally, clicking on this button launches a browser window containing a draft email to Liora in her native email client which the user can then customize and send as shown in Figure 4.6d.

Context name: facebook

CNN training data: 885 images of facebook pages and profiles

DNN *keywords*: 366 names of facebook friends

Recommendation when DNN *keyword* recognized: Compose and email to the detected friend (e.g. Email friend.name@mail.com). Each friend name (DNN *keyword*) was matched with the email address of that contact to create the action recommendations.

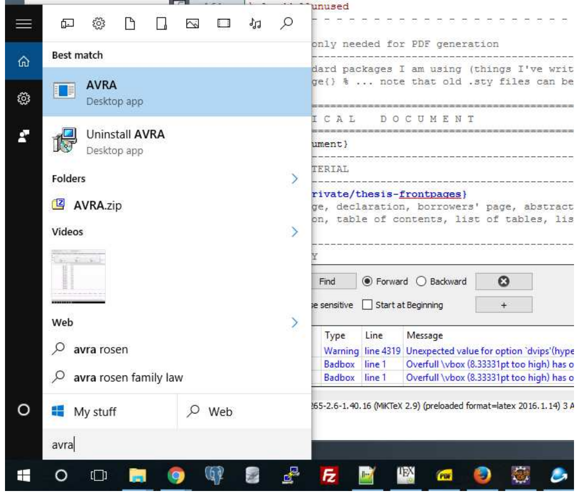
Problems encountered: For CNN training, filtered out images of login page, mobile views of facebook pages and profiles. Even though many of the training images were annotated with handwritten text, arrows, and circles, the CNN classifier had no trouble learning what facebook looks like.

Datasources: [60], [141]

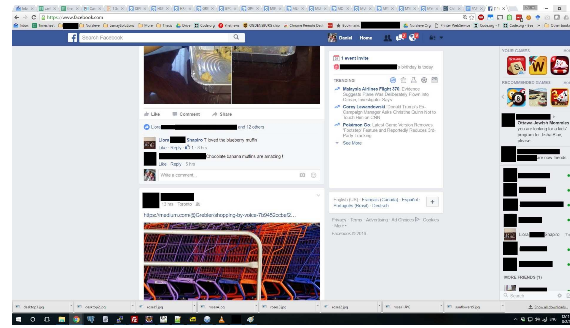
4.7 Chapter Summary

This chapter presented several examples where AVRA can be applied.

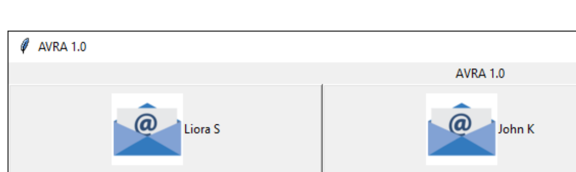
Some use cases were more useful than others, and several use cases exposed technical challenges such as text printed across more than one line. These examples for how the AVRA can benefit the user were limited in utility. They were a demonstration of what AVRA can do, rather than what it should do. For example, names appearing



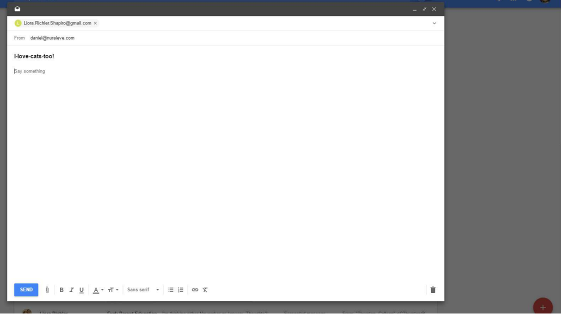
(a) Launching AVRA



(b) Context and *keyword* detection. Names of all contacts but one have been redacted to protect privacy.



(c) Action recommendation



(d) Recommended action launched

Figure 4.6: Use case illustration: AVRA assisting the user when browsing social media

onscreen should not pester the user with suggestions to compose an email to each of them. Similarly, gene names appearing onscreen should only appear when the user wants to look them up. The examples in this chapter support the thesis statement that a deep learning artificial intelligence can provide action recommendations related to onscreen messages.

Chapter 5

Context Recognition with a Convolutional Neural Network

This chapter describes the CNN used to process fullscreen images of the computer screen into one or more image classes called contexts. Recognition of the onscreen contexts enables AVRA to reason about the meaning of onscreen text when it looks for actions to recommend to the user. Each context detected by the CNN with a high enough confidence triggers the activation of a context-specific process to analyze the onscreen text. Issues surrounding the design of the CNN are discussed, including execution time, recall, precision, and supervised learning.

In AVRA, context is obtained using a trained CNN classifier to extract features from a screen capture and then classifying these features onto a set of stereotypes at the softmax output of the CNN. For example, the trained CNN may see a screen print of the Eclipse Java IDE as 80.95% eclipse-like and 18.79% desktop-like. These two contexts result from the CNN identifying features in the image that look like the desktop of a computer including the task bar, clock, and start button. Similarly, the look and feel of the Eclipse IDE for Java Developers may result in the CNN identifying image features implying the presence of that context as well. AVRA can interpret one image in more than one context. This is important for the very common use case where the user places two application windows side by side on the computer screen. Any context that is detected by the CNN with a confidence higher than a predefined threshold K (e.g. K=10%) will signal to the *keyword* recognition DNN for that context that it should process the onscreen text in search of *keywords* that the RS has been taught to associate with action recommendations. The CNN uses the predicted relevant contexts

to steer the OCR text through a filtering stage described in Section 6 and then on to the corresponding DNNs, restricting the search space for action recommendations. Limiting the RS processing to specific contexts reduces the execution time of the RS. Continuing with the Eclipse IDE example, the "eclipse" DNN activated by the CNN detection of the IDE programming context will proceed to recognize Java error messages on the screen as described in Section 7.

Diving deeper into the features a CNN observes in a computer screen image, the pre-trained Inception CNN [7] [230] was used explore the features common in computer screen images. The CNN was used to classify 10 images of the eclipse IDE and 10 images of console windows. Examples of these images are presented in Figure 5.1. For each image, the 10 most activated features in the classifier output were recorded, and Inception's label related to each of these features was also recorded. For each image type (eclipse and console) the activation level for each feature for all ten images was summed. This result, presented in Table 5.1, gives a good overall description of which features in Inception are activated in a given image type. The 32 features listed in the table were responsible for 81.6% of the activations for images of console, and 90.7% of the activations for images of the Eclipse IDE. The feature "web site, website, internet site, site" was responsible for most of the activation related to both images. Clearly, there are fewer features strongly activated by images of the Eclipse IDE. This result indicates that perhaps a classifier can be trained to discriminate between these images based upon the features of these different image classes (i.e. CNN context).

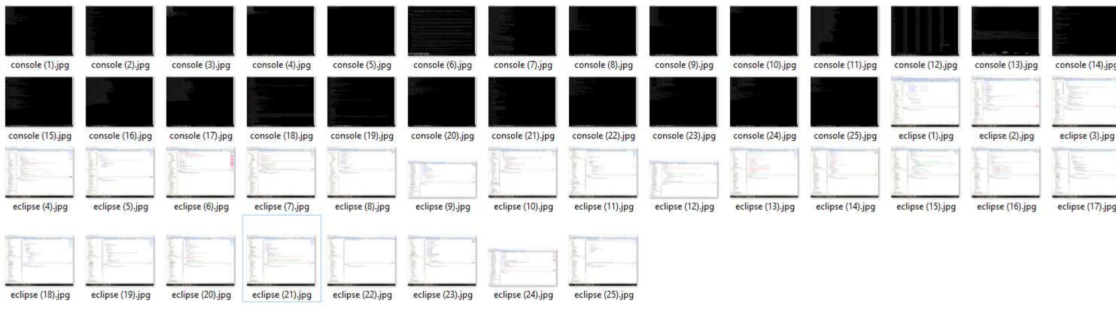


Figure 5.1: Examples of console and Eclipse IDE screenshots

Each CNN context represents a knowledge domain of related *keywords* and actions learned by one of AVRA's DNNs. The deep learning CNN in AVRA is used to interpret onscreen graphical hints to predict the contexts that are relevant to the current screen image. This approach of isolating knowledge into contexts is a way of isolating specialized

Table 5.1: CNN features activated by console and/or Eclipse IDE screenshots

Feature name from Inception	Eclipse IDE	Console
analog clock	0	0.00921
binder, ring-binder	0.00257	0.10522
book jacket, dust cover, dust jacket, dust wrapper	0	0.24755
brass, memorial tablet, plaque	0	0.00641
chain mail, ring mail, mail, chain armor, chain armour, ring armor, ring armour	0	0.00837
cloak	0	0.00811
comic book	0	0.00377
computer keyboard, keypad	0.01275	0
desk	0.00809	0
desktop computer	0.08155	0.1177
digital clock	0	0.01278
envelope	0.00263	0.06244
hand-held computer, hand-held microcomputer	0.00071	0
laptop, laptop computer	0.0853	0.02972
loudspeaker, speaker, speaker unit, loudspeaker system, speaker system	0	0.16193
menu	0.00661	0.00845
modem	0	0.00681
monitor	0.73201	0.51427
mouse, computer mouse	0.00673	0
notebook, notebook computer	0.1118	0.07426
oscilloscope, scope, cathode-ray oscilloscope, CRO	0.01249	0.01372
packet	0	0.00119
perfume, essence	0	0.00738
pole	0	0.00658
radiator	0	0.00725
radio, wireless	0	0.04437
screen, CRT screen	0.17703	0.44099
spotlight, spot	0	0.01351
syringe	0	0.00671
television, television system	0.00162	0.09622
web site, website, internet site, site	7.83109	5.95115
window screen	0	0.19693
SUM	9.07298	8.163
% of activation related to the features above	90.7 %	81.6 %

sets of information so that they can be more easily curated by AVRA's developers.

AVRA contexts need not be programs on the computer. They can just as easily be the contents of those programs. For example, a CNN trained to recognize the Eclipse IDE may recognize Java error messages and stack traces as distinct features in the training set. The trained CNN classifier will detect the stacktrace regardless of the program presenting it on the computer screen (e.g. putty, Eclipse, Notepad, Stack Overflow, and others). Another interesting point is that these CNN-derived contexts are time-invariant, and therefore easy to test with. With LSTM and RNN approaches, context is built up based upon past states as with a state machine, whereas with the approach in this work, the CNN processes a still image to generate a context without considering the past. Making the Markovian assumption here is important, because computer users tend to change contexts very quickly, for example with the "ALT+TAB" keyboard shortcut, and so assuming that a causal relationship exists between frames displayed to the computer screen dramatically complicates the model required to interpret the data.

Some further examples help to illustrate AVRA's image processing capabilities. Consider a computer screen image containing a view of the Eclipse IDE, and the features observed in a similar image of the NetBeans IDE. The CNN sees different features in these two IDEs as their texture, color, fonts, and other low level features are not similar to each other. It is the human mind that associates these two relatively equivalent tools because of their similar function, rather than their visual similarities. From the perspective of AVRA's CNN, these two programs would look entirely different. Consider another example where an options window inside the Eclipse IDE is contrasted with a view of the IDE that does not have this options window open. These screenshots may have different features, but the image dataset used to train the CNN contains images of open options windows and the IDE without open options windows. The result is that both images activate the same context. The CNN training dataset is therefore key in setting the boundary between context domains within the supervised learning realm. It is therefore interesting to ask what features of the input image are observed by the CNN.

Our understanding of how the neural codes inside a deep neural network represent information about the world around us is still developing. For example, what does a CNN really think a computer screen looks like? One way to interpret neural network models is by finding inputs that force one or a set of neurons to strongly activate, or by observing the activation of one or more neurons for some training data [109]. Starting with a randomly initialized image (noise), and iteratively maximizing the activation only the neurons that are related to a specified image type by adjusting the image through

backpropagation. Eventually we obtain a different image representing how our network visualizes the specified image type. The resulting images for each type of computer screen image represent the features of that type of image. In this work, to visualize the CNN model, style transfer was used to expose the features of images instead [205]. Style transfer is the process of applying the style of one image to the content of another [67].

Style transfer of layer 4 was implemented using the vgg16 CNN [214] and [94]. Style transfer from a desktop image to a photo of the author, and Eclipse IDE to a photo of the author are present in Figures 5.2a and 5.2b. It is clear from the style transfer that the features encoded into the neural network are sometimes useful (e.g., Figure 5.2a shows an image of the author decorated with desktop icons, revealing that the CNN likes to see desktop icons for the desktop context), and sometimes arbitrary (e.g. the author's image in Figure 5.2a is tinted blue as the style of the desktop image was arbitrarily a blue color). Mixing the style and content of images within the same class reveals the commonality between them. For example, in Figure 5.2d the file menu features and code features in the mixed image are apparent, whereas they were not revealed in the mixed image of Figure 5.2b which did not involve style transfer on images of the same class.

The CNN in AVRA was implemented using tensorflow [6] and inception v3 [230]. The training of the CNN is described in Section 5.1.

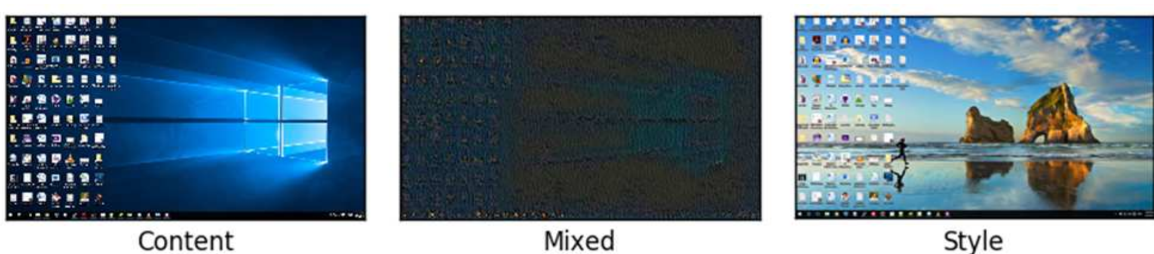
5.1 Supervised Context Learning

Consider first the training of a CNN to recognize the context from a computer screen image. Transfer learning is the process of training a neural network for one application, and using the final weights from that network after training as the starting point for a different but related application [263]. In AVRA, transfer learning is used to improve CNN training time and classification accuracy [263]. The transfer learning was implemented using the inception v3 CNN [230], taking advantage of feature detection capabilities of image recognition software trained on large sets of images. The more classes included in a classification problem, the more complex the model must be to differentiate between classes. This leads to a requirement for more training data. One advantage of transfer learning is that only the final layers become more complex in this case, and these final layers are only a small portion of the overall CNN model parameters which must be learned. This explains the relative insensitivity of the CNN to the number of classes added to AVRA.

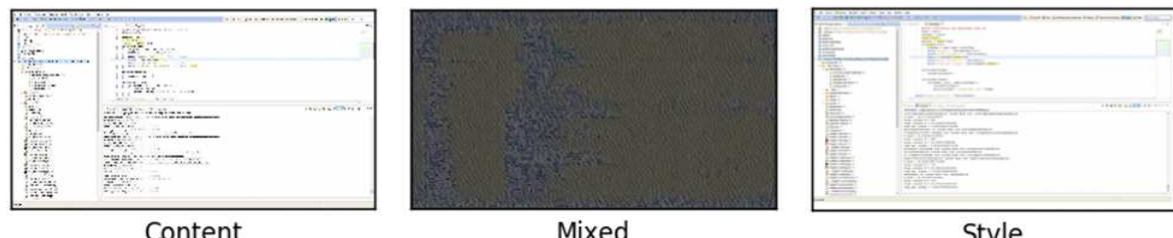
One problem with training a new class into the CNN is insufficient high-quality

(a) Example style transfer between image of the desktop and image of the author

(b) Example style transfer between image of the Eclipse IDE and image of the author



(c) Example style transfer between two images of the desktop



(d) Example style transfer between two images of the Eclipse IDE

Figure 5.2: CNN style transfer examples

testing and training image data available on hand. These image datasets are required in order to teach the CNN through supervised learning in a general enough way that the CNN will recognize testing data that was not part of the training set with reasonable accuracy. Typically several hundred images of a class are required in order to effectively train a CNN. The CNN has many output classes into which it can classify an image. For supervised context learning in AVRA we use the same dataset for testing and training, and rely on cross-validation to measure the model fitness. The challenge is acquiring images that look like a particular domain, for example the Eclipse IDE.

One solution to this lack of readily available training data was to extract representative images from the Internet to form datasets. To train AVRA, this data scraping activity was performed for several contexts including desktop screen captures of Windows, and the Eclipse IDE for Java Developers. To collect the images, a nodejs program relying on the images-scraper library [179] cycled through a hand-crafted list of keywords relating to the desired context e.g. "eclipse IDE java programming" and submitted these keywords to various search engines. The search engine submissions return lists of URLs for images and additional information about these images such as image type and size. The image search was narrowed to include only JPEG formatted images with a minimum with of 1024 pixels. The next step involved manual data validation where non-representative images were deleted. A further step of duplicate image deletion was accomplished with an automated tool. Further information on how the collected images were applied is provided in Chapter 4.

A second solution to the lack of readily available training data was to generate the training images locally. For example, image captures from a local machine that are representative of the problem were collected. The image capture program was executed for several days to generate sufficient data to browse and extract relevant images.

5.2 Precision and Recall of Context Recognition with CNN

Four examples of training images are presented in Figure 5.3. The training images were segmented into 11 classes. The number of training images per class was as follows: A(2946), B(980), C(190), D(300), E(656), F(884), G(800), H(634), I(899), J(700), K(642). After training, the CNN's performance was measured in terms of the precision and recall and sample ROC curves were examined. The testing images (5 images per

class for 11 classes) were not included in the training dataset. The results are presented in a confusion table as Table 5.3.

A graph of the training and cross-validation accuracy at each epoch during the CNN training is presented in Figure 5.4, produced with inception's retraining code [230]. The retraining configuration included 4000 training steps. Figure 5.4 shows that validation accuracy lags training accuracy, as expected. The shape of the learning graph is also as expected, with an initial high rate of learning followed by slower incremental improvements in accuracy.

For the CNN characterization, let TN be the number of true negative classifications. These are instances where an image was classified correctly as not being in a certain class. Let TP be the number of true positive classifications. These are instances where an image was classified correctly as being in a certain class. Let FN be the number of false negative classifications. These are instances where an image was classified incorrectly as not being in a certain class. Let FP be the number of false positive classifications. These are instances where an image was classified incorrectly as being in the wrong class.

In AVRA, the classifier can make more than one prediction per image. Multiple different contexts can be present on the computer screen at the same time, and therefore consider that one image of the computer screen containing side-by-side windows of the Eclipse IDE and a console window could trigger two predictions which may both be accurate. For simplicity, the testing dataset used to produce the confusion matrix of Table 5.3 only contained images with one "actual" class per image. The threshold K for the CNN making a class prediction (one of many AVRA hyperparameters) is a key parameter in tuning AVRA to be more conservative (higher threshold) or more open to evaluating hypotheses (lower threshold). In the confusion matrix of Table 5.3, predicted classes are columns, and actual classes are rows. Cells are counts of classifications or misclassifications. On the diagonal where a column intersects a row with the same label (e.g. column "A" and row "A") is the count of true positive results. Integers in all other cells represent classification errors. Unlike a typical confusion table, the number of tests for any class is not the sum of the integers in a given row. The sum for each row is the number of predictions made by the CNN for the 5 test images it processed. To calculate precision for a column, the number of true positive classifications TP (where row and column label match) is divided by the sum of the integers in the column (representing all TP and FP). Recall is calculated as TP divided by the sum of the integers in the row (corresponding to all TP and FP). For example, the first results row ("A") of Table 5.3 was generated by processing 5 images containing only context "A" through

doi:10.1006/geno.2002.6698, available online at http://www.idealibrary.com on IDEAL **Article**

Evolution of the Regulators of G-Protein Signaling Multigene Family in Mouse and Human

David A. Sierra,[1] Debra J. Gilbert,[2] Deborah Householder,[2] Nick V. Grishin,[3] Kan Yu,[1] Pallavi Ukidwe,[1] Sheryll A. Barker,[1] Wei He,[4] Theodore G. Wensel,[4] Glen Otero,[5] Greg Brown,[5] Neal G. Copeland,[2] Nancy A. Jenkins,[2] and Thomas M. Wilkie[1,*]

[1]Pharmacology Department, UT Southwestern, Dallas, Texas 75390-9041, USA
[2]Mouse Cancer Genetics Program, National Cancer Institute-Frederick, Frederick, Maryland 21702, USA
[3]Howard Hughes Medical Institute, Biochemistry Department, UT Southwestern, Dallas, Texas 75390-9152, USA
[4]Baylor College of Medicine, Molecular Biophysics, Houston, Texas 77030, USA
[5]DoubleTwist, Oakland, California 94612, USA

[*]To whom correspondence and reprint requests should be addressed. Fax: (214) 648-8626. E-mail: thomas.wilkie@utsouthwestern.edu

The regulators of G-protein signaling (RGS) proteins are important regulatory and structural components of G-protein coupled receptor complexes. RGS proteins are GTPase activating proteins (GAPs) of Gi- and Gq-class Gα proteins, and thereby accelerate signaling kinetics and termination. Here, we mapped the chromosomal positions of all 21 *Rgs* genes in mouse, and determined human *RGS* gene structures using genomic sequence from partially assembled bacterial artificial chromosomes (BACs) and Celera fragments. In mice and humans, 18 of 21 *RGS* genes are either tandemly duplicated or tightly linked to genes encoding other components of G-protein signaling pathways, including Gα, Gγ, receptors (GPCR), and receptor kinases (GPRK). A phylogenetic tree revealed seven *RGS* gene subfamilies in the yeast and metazoan genomes that have been sequenced. We propose that similar systematic analyses of all multigene families from human and other mammalian genomes will help complete the assembly and annotation of the human genome sequence.

Key words: GAP, GTPase accelerating protein (GAP), Gγ-like (GGL), G-protein coupled receptor (GPCR), multigene family, phylogenetic tree

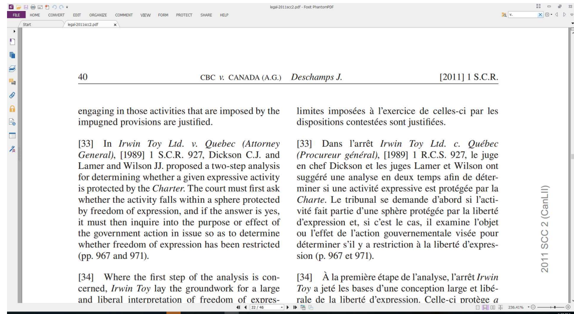
INTRODUCTION

G-protein signaling mediates intercellular communication in a diverse group of eukaryotes. Fungi, *Dictyostelium discoideum*, and animals express all components of the signal transduction cascade, including ligands, receptors, effectors, heterotrimeric Gαβγ proteins, and the regulators of G-protein signaling (RGS) proteins. Among the higher eukaryotes, only plants have not yet been shown to have RGS proteins, but they express all other G-protein signaling components. This broad phylogenetic distribution implies that G-protein signal transduction evolved with ancestral eukaryotes over one and a half billion years ago and that RGS proteins have a fundamental regulatory role. Indeed, mutations in RGS proteins block recovery from mating-pheromone-induced cell cycle arrest in yeast, alter motility and egg laying in worms, cause duplications of vertebrate head structures, and delay recovery from a light flash in photoreceptor cells [1–5]. Other members of the G-protein-coupled signaling pathway have been linked to human disease, such as GNAS1 in pseudohypoparathyroidism and McCune-Albright syndrome, transducin in congenital night blindness, and RPE retinal G-protein-coupled receptor (RGR) in autosomal recessive retinitis pigmentosa (see the OMIM database, http://www.ncbi.nlm.nih.gov).

RGS proteins regulate G-protein signaling by accelerating transit through the cycle of GTP binding and hydrolysis. G-protein signaling is activated when Gα binds GTP to release both Gα-GTP and Gβγ. Signaling terminates when GTP hydrolysis allows reassociation of the heterotrimer Gα–GDPβγ. RGS proteins accelerate signal termination because they are GTPase accelerating proteins (GAPs) for Gα subunits [6]. Most RGS proteins are relatively nonspecific Gi- and Gq-GAPs, but some RGS proteins, such as RGSZ1 and RGS2, show Gα substrate specificity [7,8]. GAP activity is conveyed by an approximately 130-amino-acid motif termed the RGS domain [9]. Distantly related RGS-like (RGL) domains in p115RhoGEF-related proteins (rgRGS) are GAPs for α-subunits of the G12 class [10], whereas no dedicated GAPs have yet been identified for α-subunits of the Gs class. (see note added in proof).

GENOMICS Vol. 79, Number 2, February 2002 177
Copyright © 2002 by Academic Press. All rights of reproduction in any form reserved.
0888-7543/02 $35.00

(a) Example scientific research article view [211]


40 CBC *v.* CANADA (A.G.) *Deschamps J.* [2011] 1 S.C.R.

engaging in those activities that are imposed by the impugned provisions are justified.

[33] In *Irwin Toy Ltd. v. Quebec (Attorney General)*, [1989] 1 S.C.R. 927, Dickson C.J. and Lamer and Wilson JJ. proposed a two-step analysis for determining whether a given expressive activity is protected by the *Charter*. The court must first ask whether the activity falls within a sphere protected by freedom of expression, and if the answer is yes, it must then inquire into the purpose or effect of the government action in issue so as to determine whether freedom of expression has been restricted (pp. 967 and 971).

[34] Where the first step of the analysis is concerned, *Irwin Toy* lay the groundwork for a large and liberal interpretation of freedom of expres-

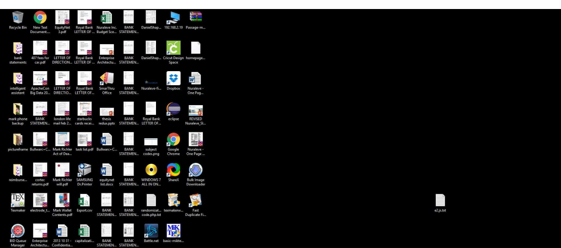
limites imposées à l'exercice de celles-ci par les dispositions contestées sont justifiées.

[33] Dans l'arrêt *Irwin Toy Ltd. c. Québec (Procureur général)*, [1989] 1 R.C.S. 927, le juge en chef Dickson et les juges Lamer et Wilson ont suggéré une analyse en deux temps afin de déterminer si une activité expressive est protégée par la *Charte*. Le tribunal se demande d'abord si l'activité fait partie d'une sphère protégée par la liberté d'expression et, si c'est le cas, il examine l'objet ou l'effet de l'action gouvernementale visée pour déterminer s'il y a restriction à la liberté d'expression (p. 967 et 971).

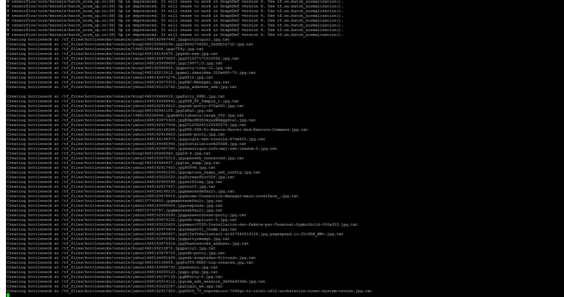
[34] À la première étape de l'analyse, l'arrêt *Irwin Toy* a jeté les bases d'une conception large et libérale de la liberté d'expression. Celle-ci protège *a*

2011 SCC 2 (CanLII)

(b) Example legal document view [226]



(c) Example desktop view



(d) Example console programming view

Figure 5.3: CNN example training images

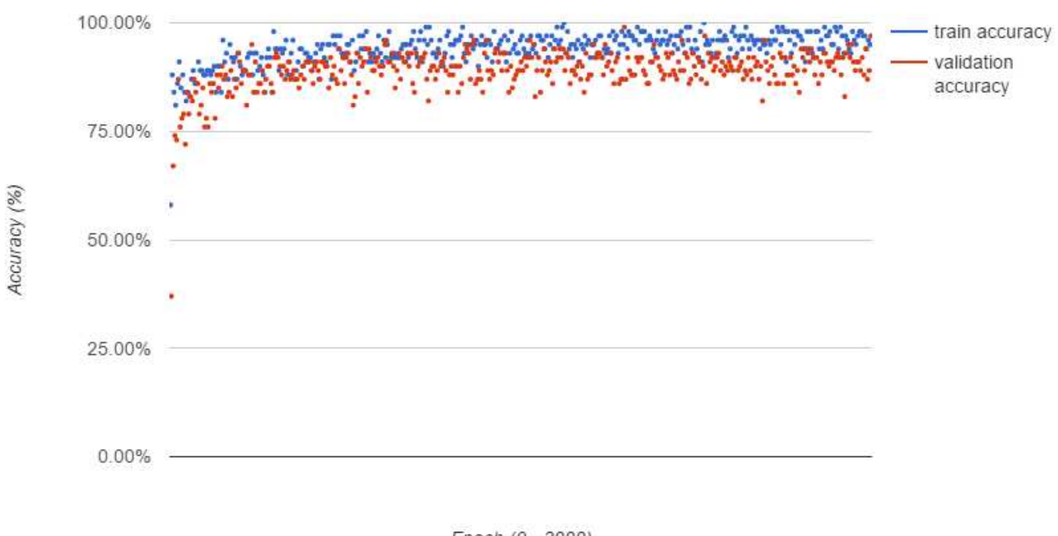


Figure 5.4: Training and validation classification accuracy during the 4000 epochs of the CNN training process. The final test accuracy was 92.20%

Table 5.2: Interpretation of the CNN's ROC area under the curve with qualitative labels

Fig.	Curve #	Area	Quality Label	Fig.	Curve #	Area	Quality Label
5.3	1	0.18	worse than chance	5.4	1	0.14	worse than chance
5.3	2	0.12	worse than chance	5.4	2	0.2	worse than chance
5.3	3	0.26	worse than chance	5.4	3	0.15	worse than chance
5.3	4	0.28	worse than chance	5.4	4	0.3	worse than chance
5.3	5	0.46	worse than chance	5.4	5	0.43	worse than chance
5.3	6	0.56	fail	5.4	6	0.51	fail
5.3	7	0.63	poor	5.4	7	0.65	poor
5.3	8	0.73	fair	5.4	8	0.71	fair
5.3	9	0.82	good	5.4	9	0.83	good
5.3	10	0.76	fair	5.4	10	0.76	fair
5.3	macro-average	0.53	fail	5.4	macro-average	0.5	fail
Fig.	Curve #	Area	Quality Label	Fig.	Curve #	Area	Quality Label
5.5	1	0.14	worse than chance	5.6	1	0.08	worse than chance
5.5	2	0.18	worse than chance	5.6	2	0.16	worse than chance
5.5	3	0.25	worse than chance	5.6	3	0.3	worse than chance
5.5	4	0.31	worse than chance	5.6	4	0.39	worse than chance
5.5	5	0.41	worse than chance	5.6	5	0.43	worse than chance
5.5	6	0.47	worse than chance	5.6	6	0.57	fail
5.5	7	0.58	fail	5.6	7	0.69	poor
5.5	8	0.64	poor	5.6	8	0.81	good
5.5	9	0.77	fair	5.6	9	0.85	good
5.5	10	1	excellent	5.6	10	1	excellent
5.5	macro-average	0.5	fail	5.6	macro-average	0.56	fail

the CNN. During each classification, the CNN predicted the correct class “A” as well as some other classes (“B” and “C” every time, and “E” and “F” four out of 5 times). The recall for this situation is 1.0, because the presence of an image containing context “A” always resulted in the activation of context “A”. However, the precision (bottom of column “A”) which asks how often context “A” is activated by information other than the presence of context “A” is also relatively high at 0.71. Only the presence of context “B” caused the CNN to include “A” in its context guesses.

The recall observed in Table 5.3 is high at the expense of precision. The process followed to identify a suitable value for K was to try various values until the recall was maximized at 100%. There is a fundamental trade-off between recall and precision determined by the threshold K. This is demonstrated with the confusion table in Table 5.4, where K=95%. With such a high requirement for certainty that a context has been detected, there is a higher precision, but only when the context is detected. The cost of increasing K is missing the context completely, costing the algorithm on both the precision and the recall metrics. False positive context detection wastes CPU time but does not lower the recognition rate, whereas false negative context detection causes a decrease in recall. Therefore, AVRA is configured with a low K threshold as in Table 5.3, rather than a high value as in Table 5.4. It is worth emphasizing that false positive context detection is a small price to pay for high true positive detection. When the recommendations are ranked by subsequesnt modules in AVRA, a false context detection, even if falsely detected with high confidence, will be ranked lower if the keywords related to that context are not detected from within the onscreen image. It is therefore acceptable to maximize recall at the expense of precision.

Delving deeper into the trade-off between the true positive and false positive rates, four example ROC curves are presented in Figures 5.5, 5.6, 5.7, and 5.8. Converting these results into tabular form using the scoring method described in [22] produces Table 5.2. The qualitative results of Table 5.2 seem at first to contradict the excellent recall and average precision results of Table 5.3, and the real world experience that AVRA does “work” with K=1%. What the ROC curves are in fact revealing is that specific contexts require a high false positive rate in order to be detected frequently. The number of classes trained into the CNN did not seem to be the key variable. Training 5 classes instead of 11 yielded a similar overall ROC curve. Rather than the number of classes being the problem, it is the amount of data per class. There is not enough training data available to the model to demand better than chance prediction of the onscreen contexts without also making mistakes. One solution to lower the false positive rate is to increase the amount

of data collected per context. Contexts with a low area under the curve corresponded to smaller numbers of training images. However, the downside of requiring many hundreds of images per context is that AVRA then becomes difficult to train because the required training data is difficult to obtain without significant human intervention or waiting a long time for the algorithm to gain enough data to learn a new context. These problems of insufficient data and extracting data for training is discussed further in Chapter 9 on unsupervised learning. What was identified thus far in this chapter is that the CNN can detect a number of contexts, but only under the condition that it will also make many false-positive guesses.

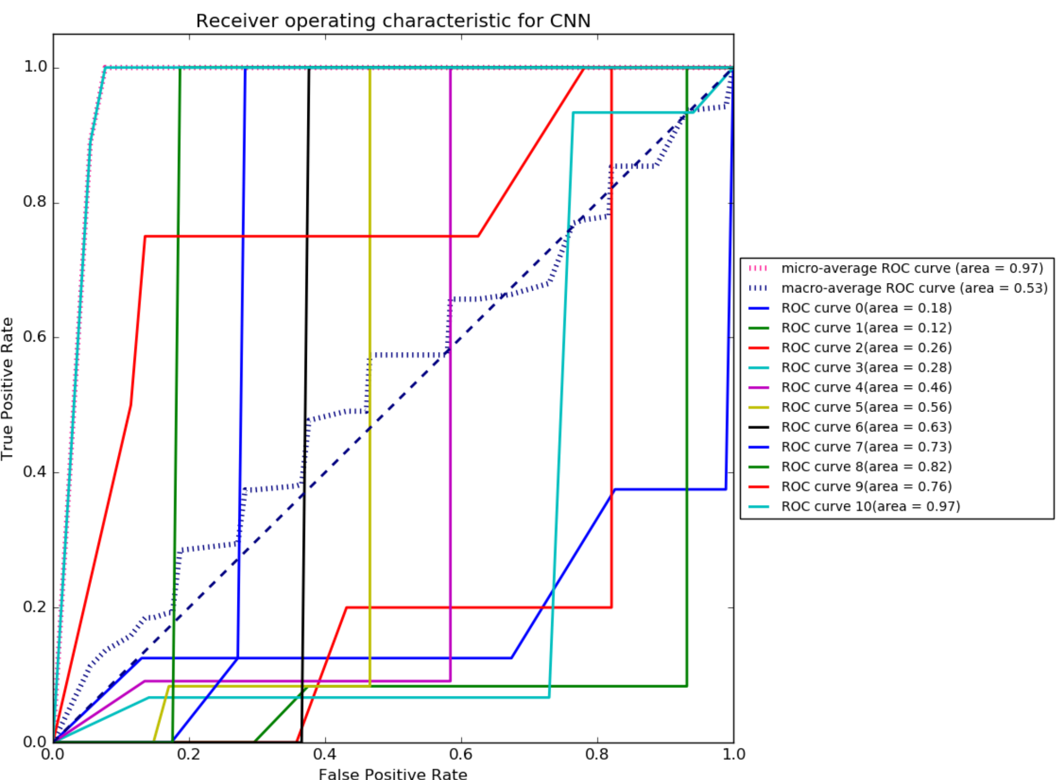


Figure 5.5: The first of 4 ROC curve examples. The CNN was trained on 10 classes of images resulting in a macro-average ROC curve with area under the curve of 0.53 and the class with the lowest area under the curve had area 0.12

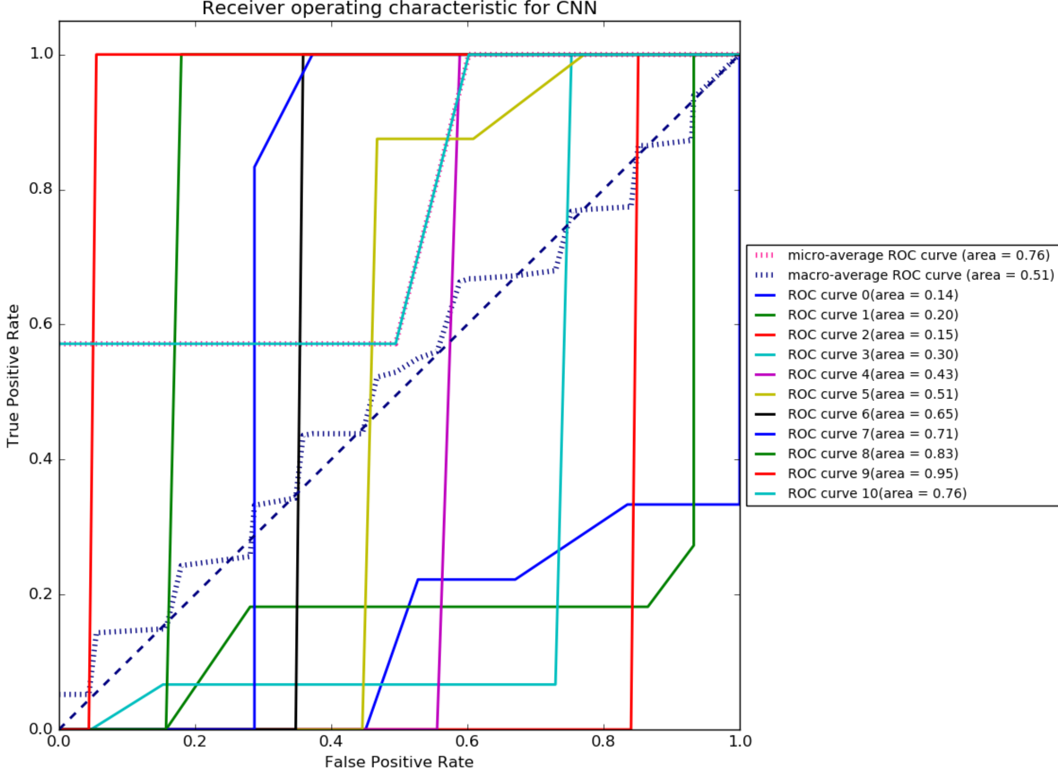


Figure 5.6: The second of 4 ROC curve examples. The CNN was trained on 10 classes of images resulting in a macro-average ROC curve with area under the curve of 0.51 and the class with the lowest area under the curve had area 0.14

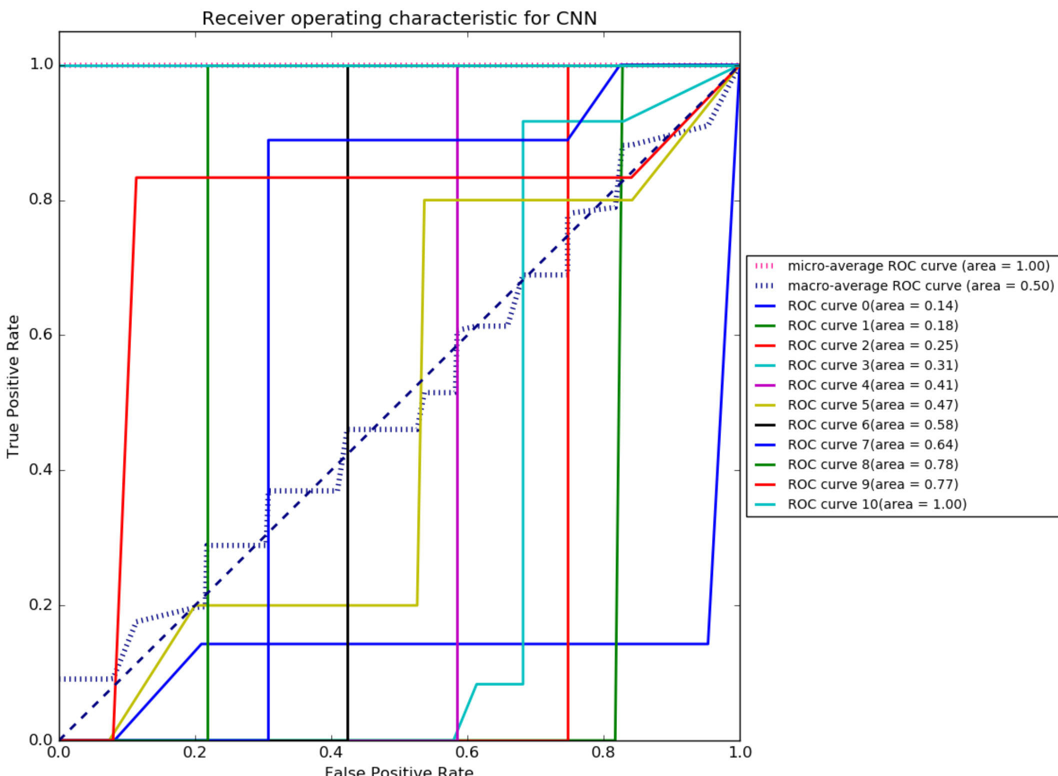


Figure 5.7: The third of 4 ROC curve examples. The CNN was trained on 10 classes of images resulting in a macro-average ROC curve with area under the curve of 0.50 and the class with the lowest area under the curve had area 0.14

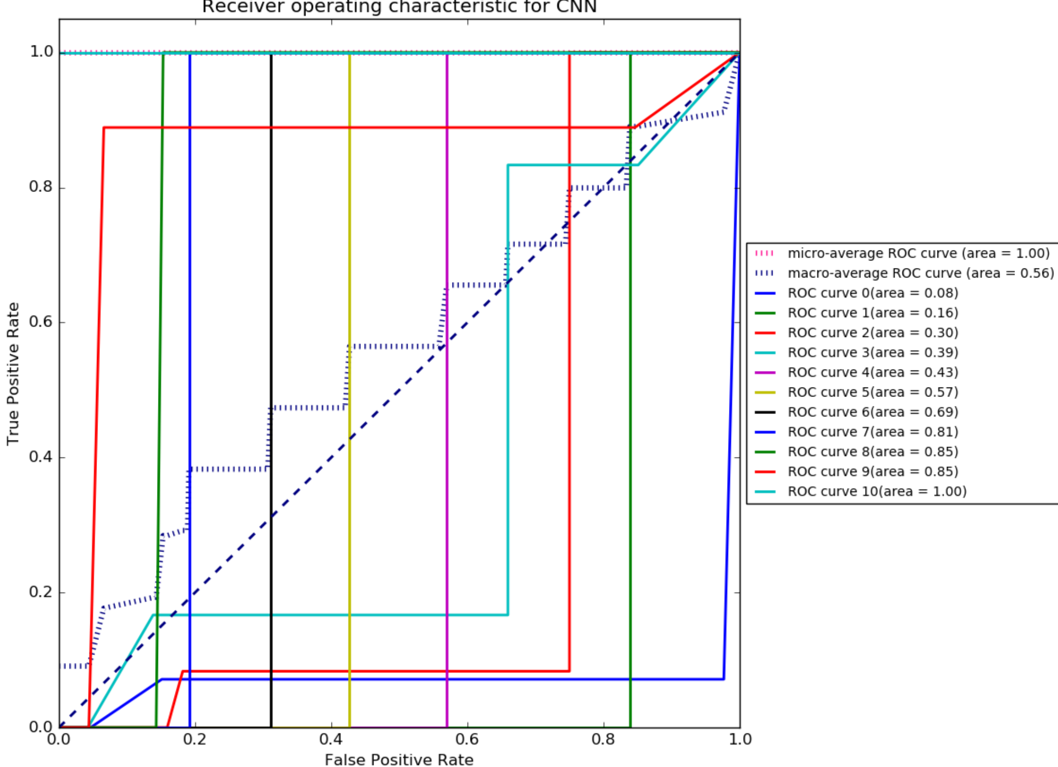


Figure 5.8: The fourth of 4 ROC curve examples. The CNN was trained on 10 classes of images resulting in a macro-average ROC curve with area under the curve of 0.56 and the class with the lowest area under the curve had area 0.08

Table 5.3: Confusion matrix for assessing AVRA's CNN after it was trained on 11 classes. The testing data had 5 images per class, with 1 class per image, and **classification threshold K=1%.**

		Predicted											Recall
Actual		A	B	C	D	E	F	G	H	I	J	K	
	A	5	5	5	0	4	4	0	0	0	0	0	1
	B	2	5	5	2	5	5	0	0	0	0	0	1
	C	0	3	5	0	5	5	0	0	0	0	0	1
	D	0	0	0	5	2	0	1	0	1	0	1	1
	E	0	4	1	2	5	0	0	0	0	0	0	1
	F	0	0	5	5	5	5	0	0	0	0	0	1
	G	0	0	0	0	0	0	5	0	0	0	4	1
	H	0	0	0	0	0	0	0	5	0	1	0	1
	I	0	0	0	0	0	0	3	1	5	4	0	1
	J	0	0	0	0	0	0	2	3	2	5	0	1
	K	0	0	0	0	0	0	3	0	0	0	5	1
Precision		0.71	0.29	0.24	0.36	0.19	0.26	0.36	0.56	0.63	0.50	0.50	

Table 5.4: Confusion matrix for assessing AVRA's CNN after it was trained on 11 classes. The testing data had 5 images per class, with 1 class per image, and **classification threshold K=95%.**

	Predicted												Recall
Actual		A	B	C	D	E	F	G	H	I	J	K	
	A	0	0	0	0	0	0	0	0	0	0	0	0
	B	0	0	0	0	0	0	0	0	0	0	0	0
	C	0	0	0	0	0	0	0	0	0	0	0	0
	D	0	0	0	3	0	0	0	0	0	0	0	0.6
	E	0	0	0	0	1	0	0	0	0	0	0	0.2
	F	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	2	0	0	0	0	0.4
	H	0	0	0	0	0	0	0	5	0	0	0	1
	I	0	0	0	0	0	0	0	0	1	0	0	0.2
	J	0	0	0	0	0	0	0	0	0	2	0	0.4
	K	0	0	0	0	0	0	0	0	0	0	3	0.6
Precision		0	0	0	1	1	0	1	1	1	1	1	

5.3 Chapter Summary

This chapter described the CNN used recognize contexts in images of the computer screen, and how to train the CNN with supervised learning. This was essential to the thesis statement that action recommendations can be provided without integration into each individual program executing on the computer. Context recognition enables AVRA to recommend appropriate actions. Context detected by the CNN includes a confidence score, which enables AVRA weigh action recommendations. Also, the CNN performance was characterized in this chapter in terms of execution time, recall, and precision.

Chapter 6

Fast Context-specific Text Filtering

Interpreting all of the onscreen text for each context detected in a computer screen image is unacceptably slow. This Chapter describes a text filtering module (labeled F in Figure 3.1) which accelerates the text analysis by discarding text that is not similar to the text trained into AVRA. This filtering task reduces the amount of text to be classified, thereby reducing the execution time. Several text similarity algorithms were candidates for implementing this filtering task, and this Chapter describes the observations and analysis carried out to determine which algorithm should be used to implement the text filter.

In AVRA, the recognition of onscreen *keywords* such as computer error messages is achieved through the extraction and classification of textual information from the image data in a screen capture. This text is extracted using Optical Character Recognition (OCR), and then chopped into many small segments using a windowing approach described below. These text segments are filtered to remove text dissimilar from the *keywords* recognized by AVRA for the context in question. The surviving text segments are classified and ranked as described in Section 7.

Neural networks are not always the best solution when latency minimization is crucial. The input text filtering algorithm used here is optimized for speed, and relies on loop optimization, dictionary memorization (called memoization), and binary operations to compare strings and compute a similarity score between candidate text and the text known to a particular DNN. Rather than classifying all text into a class, this filter discards *keywords* too dissimilar to the text on which a given DNN has been trained. The filter is not exhaustive in its comparison of candidate text with DNN *keywords*. Rather, once candidate text is similar to any *keyword* in the targeted DNN, the candidate

is considered fit to be classified by the DNN, and the filter moves to evaluate the next candidate *keyword*. This filtering algorithm typically discards the vast majority of input text (e.g. 98% for a screen capture of a program in the eclipse IDE). This is exactly as expected, and much more efficient than passing many *keywords* to a DNN which will clearly return with some nonsensical classification of most candidate *keywords* along with a very low confidence score.

The windowing approach used on the OCR text output involves tokenizing the text with spaces as separators, and then the *keyword* array is processed one index at a time, and then two at a time, and then three at a time. For example, if the OCR text is "the cat is black", then the resulting four candidate texts in the first step would be ("the", "cat", "is", "black"). In the second step, the three candidate texts would be ("the cat", "cat is", "is black"). Finally, in the third step, the two candidate texts would be ("the cat is", "cat is black"). This windowing approach allow *keywords* broken apart by one or two injected spaces in the OCR process to be joined back together and recognized during classification. Typically this windowing approach breaks the OCR output text into thousands of short text snippets of 1 to 3 words.

The windowing approach causes a large increase in the number of candidate texts to be classified. Very often the vast majority of candidate text is not to be classified as it is not meaningful to the DNN classifier. Each image of the screen may be processed into thousands of text snippets *ocr* for evaluation by multiple DNNs. AVRA therefore includes an input text filtering algorithm described in Algorithm 2 which narrows down which text should be evaluated by each DNN.

This work is concerned with dropping text that is too dissimilar from anything trained into a DNN. The similarity between one text string and many keywords is therefore at the center of this problem. The algorithm (hereafter "the filter") which filters the candidate input text (hereafter *ocr*) checks that the similarity between *ocr* and any keyword (hereafter *keyword*) is sufficiently high to justify DNN classification. Once the filter has identified that the *ocr* is similar to any *keyword*, it stops analyzing and proceeds to pass the text to be classified by the DNN. Generally, the vast majority of *ocr* input to the filter is dropped.

Tables 6.1 and 6.2 list the filter implementation approaches considered. The approaches listed in Table 6.1 were implemented in a similar way, shown in Algorithm 1, which loops through the *keywords* with each *ocr* looking for the first exact match (line 4) or sufficiently similar *keyword* (line 8). The function $FUNC$ configured by hyper-parameters $PARAMS$ compares *keyword* and *ocr* where the result is then compared

using operation OP in relation to the hyperparameter $LIMIT$. The $LIMIT$ defines the similarity threshold past which the filter should let an *ocr* pass to the DNN.

Consider an example of how Algorithm 1 is used in this work to implement a specific approach. The package [231] is an implementation of the edit distance algorithm described in [96]. Its minimum edit distance function *editdistance.eval* is used as $FUNC$ with no additional hyperparameters $PARAMS$. The resulting minimum edit distance must be less edits than some constant defined by hyperparameter $LIMIT$, and therefore OP is the operation $<=$.

The text filtering approaches in Table 6.1 contain hyperparameters that can take on a range of values, each affecting the execution time, specificity, precision, and recall of the implementation. Table 6.1 lists these hyperparameters and their value ranges for each algorithm.

ALGORITHM 1: A Generic Filtering Algorithm

Input: $keywords, ocrs$; Classification threshold $LIMIT$; 0 or more additional hyperparameters $PARAMS$; Function reference $FUNC$; Binary comparison operation in the set $\{<=, >=\}$ OP; OCR output text snippets $ocrs$

Output: List of text strings to be classified by a neural network $DNNinput$

```
1  ed(keywords, ocrs):
2  for each ocr ∈ ocrs do
3      for each keyword ∈ keywords do
4          if keyword == ocr then
5              DNNinput.append(ocr)
6              break
7          end
8          else if OP(FUNC(keyword, ocr, PARAMS), LIMIT) then
9              DNNinput.append(ocr)
10             break
11         end
12     end
13 end
```

The L1, L2, and substring approaches were implemented using variations of Algorithm 1. L1 and L2 implementations followed a character embedding approach for encoding

Table 6.1: Hyperparameter configurations for approaches implemented using Algorithm 1

ID	**FUNC**	**FUNC Implementation**	**PARAMS (range, step size)**	**OP**	**LIMIT (range, step size)**
1	Edit Distance [96]	[231]	-	$\leq$	(0-10, 1)
2	Edit Distance [124]	[164]	-	$\leq$	(0-100, 1)
3	Ratio	[164]	-	$\geq$	(0-1, 0.1)
4	Jaro	[164]	-	$\geq$	(0-1, 0.1)
5	Jaro	[240]	-	$\geq$	(0-1, 0.1)
6	Jaro-Winkler	[164]	prefix_weight (0-1, 0.1)	$\geq$	(0-1, 0.1)
7	Jaro-Winkler	[240]	long_tolerance (0-1,1)	$\geq$	(0-1, 0.1)
8	Needleman-Wunsch	[90]	gap_cost (0.5-3,0.5)	$\geq$	(5-15, 1)
11	Cosine	[90]	-	$\geq$	(0.5-1,0.01)

Table 6.2: Hyperparameter configurations for approaches implemented using variations of Algorithm 1

ID	**Name**	**Implementation**	**PARAMS (range, step size)**	**LIMIT (range, step size)**
9	L1	[105]	windowSize(1-10,1)	(0.0001-0.1,0.01)
10	L2	[105]	windowSize(1-10,1)	(0.0001-0.1,0.01)
12	Substring	-	-	-
13	LazyJaroWinkler	Algorithm 2	swb (0-3,1), wshift(0-4,1), minsim(0-0.9,0.1), mindist(0-9,1)	(0-9,1)
14	SimHash	[123]	width(1-5,1), k (1-30,1)	-

text into feature vectors [187]. The model for encoding text was designed with a feature vector of length 100, and trained on the sequence of characters in each of the keywords. Word vectors were composed of the sum of the character vectors, divided by the number of characters in the word. The *keyword* vectors were stored in a dictionary for fast retrieval during string comparison.

The substring approach checks if any *keyword* is a substring of the input text *ocr*. The advantage of this approach is high speed and the ability to identify keywords in an *ocr* with characters added to the start or end of the *keyword*, while the disadvantage is a an inability to spot *ocr* with containing *keyword* text with one or more spelling errors. While Algorithm 1 requires at least one hyperparameter $LIMIT$, this substring approach contains no hyperparameters.

An LSH SimHash implementation that varies the length of the hash string is described in [123]. Conveniently, the comparison function compares one input text string (*ocr*) with all of the encoded text (*keywords*) with one function call to an index ($get_near_dups()$).

6.1 LazyJaroWinkler: A Fast Text Filter

LazyJaroWinkler (Algorithm 2) is an extension of Jaro-Winkler. LazyJaroWinkler was optimized for speed using loop optimization, memoization, and low latency binary operations to compare strings and compute a similarity score between *ocr* and *keywords*. The filter should improve execution time when an exact match is very unlikely. Therefore, checking for an exact match in Algorithm 2 is deeper into the loop structure. Under the most common condition when *ocr* is not similar to any *keyword*, the match failure should occur quickly in a low latency outer loop, rather than computing a more expensive metric such as edit distance during every iteration through the *keywords*. Dictionaries can be employed to store the results of common operations using memoization, further accelerating the most common case where *ocr* and *keyword* are not similar.

In Algorithm 2, for each candidate *ocr*, if it is an exact match for a keyword recognized by the DNN (*keyword*), then the candidate *ocr* is allowed to pass the filter and the loop skips to evaluate the next candidate *ocr* (lines 12 to 15). The loop optimization on lines 1 to 4 memorizes a binary-based frequency representation $binFreq()$ of each keyword known to the DNN and the number of high bits in that binary representation. These values are then available in all of the following loops without incurring a computation penalty. Loop optimization by precomputation is performed on lines 6, 7, 9, and 10. After removing non-alphanumeric characters, if the candidate *ocr* is more than *swb*

characters shorter than a given *keyword*, or if it is 2^{wshift} times longer, then it likely will not match. Shorter texts are penalized much more heavily because information in the candidate *ocr* was lost in the OCR process, whereas longer strings can result from the merging of several words, which happens on occasion and does not imply a loss of information. The similarity between the candidate *ocr* and *keyword* is computed at line 16 using Algorithm 4. Algorithm 4 computes the number of bits in common between the $binFreq()$ representations of *keyword* and the candidate *ocr*, and then divides this by the number of high bits in the *keyword*. If the two are sufficiently similar, a second check measures how different the candidate *ocr* and *keyword* are at line 17. At this point in Algorithm 2 on line 18, the filtering algorithm has established that the candidate *ocr* is similar enough to *keyword* that it is worth spending the time on a relatively slow and more accurate comparison. A candidate text string *ocr* passes the filter if the minimum edit distance is less than $LIMIT$ (line 19).

$binFreq()$ (Algorithm 3) creates an integer for a word containing one 3-bit frequency bin for each of the 26 letters in the alphabet. Each bin inside the integer saturates after 3 occurrences of the corresponding letter within the word. For example, the $binFreq()$ binary representation of "a" is 001 while "aa" 011, and abab is 011 011. A word with many of the same letter quickly saturates the corresponding bin. For example, "baaaaab" becomes 011 111, the same as "baaaab" and "baaab". The number of bits in the XOR of two $binFreq()$ results gives a rough score for the distance between words (Algorithm 2 line 17). Also, the number of bits in the AND of two $binFreq()$ results gives a rough score for the similarity as it keeps only the bits in common (Algorithm 4 line 6). One drawback of this approach is that numbers and other characters are not represented in the letter frequency analysis, and so numerical strings are not seen as similar. For example, $binFreq('a123456')$ AND $binFreq('b123456')$ becomes 000 001 AND 001 000, which resolves to 0. One would expect these 2 similar strings to have a higher similarity score than 0. Testing with realistic data, increasing the number of bins per character to more than 3 did not significantly improve accuracy but did increase the execution time. Less than 3 bins decreased accuracy. Similarly representing all characters with their own bin had a negative impact on execution time while not significantly improving accuracy.

Regarding the design of Algorithm 2, it is important to consider the 5 hyperparameters that affect execution time as well as the specificity, precision and recall of the filter. The hyperparameters for Algorithm 2 are *minsim* (line 16), *mindist* (line 18), *swb* (set on line 7 and applied on line 12 to skip over *ocr* too short in length compared to the DNN *keyword*), $wshift$ (line 10 to skip over *ocr* too long in comparison to the

DNN *keyword*), and $LIMIT$. $LIMIT$ performs the same threshold role as it does for Jaro-Winkler (IDs 6 and 7). A Jaro-Winkler score of 0 means that the 2 values are not similar, while larger values imply increased similarity. Therefore, a lower $LIMIT$ causes the algorithm to allow more results to pass the filter, possibly increasing recall, but likely decreasing precision. As more results pass the filter and are processed by the DNN, execution time increases. *minsim* filters out *ocr* which have a large edit distance from the DNN *keywords*. It is compared against an approximation of the percentage of bits in common between two binary letter frequency encoded words. Increasing the similarity threshold causes the execution time of Algorithm 2 to decrease and the number of *ocr* allowed to pass the filter decreases. Also, increasing the similarity threshold generally tends to decrease recall while increasing precision. *mindist* is used to skip *ocr* that contain too many differences from the *keyword* in question. It is compared to a count of the number of bits that are different between the binary letter frequency encoded *ocr* and *keyword*. Increasing *mindist* causes the execution time to increase and the number of *ocr* allowed to pass the filter increases. Also, increasing *mindist* generally increases recall while decreasing the precision. Increasing *swb* allows *ocr* much shorter that the *keyword* and causes the execution time to increase while the number of *ocr* allowed to pass the filter increases. This is particularly problematic when *ocr* is much shorter than *keyword*, indicating that *ocr* is likely irrelevant text, and so increasing *swb* generally increases recall while decreasing precision. $wshift$ filters out *ocr* with a length much longer than the length of the keyword in question. Increasing $wshift$ allows longer and longer *ocr* to pass the filter, causing the execution time to increase. Also, increasing $wshift$ generally increases recall while decreasing the precision.

6.2 Trade-Off Between Filter Recall and Execution Time

In this section, design space exploration for the filter implementation is documented. The objective is to discover which algorithms and hyperparameter configurations result in low latency, pass simple tests, and result in high recall, precision and specificity. Four datasets were constructed that correspond to interesting extreme cases within the space of all possible inputs to the filter from the OCR software. These corner cases are sanity checks to make sure that the algorithm for each approach handles an easy to verify task correctly and in reasonable time. A fifth dataset containing real world data was

created to model realistic conditions. These 5 datasets were used to evaluate the many hyperparameter settings for each of the approaches to implementing the filter. Dataset 1 contained 300 *keyword* and 3,000 *ocr*, with 0 *ocr* similar to any *keyword*. In this case the filter should reject all 3,000 *ocr*. Dataset 2 contained 300 *keyword* and 3,000 *ocr*, with each *ocr* randomly assigned a *keyword*. In this case the filter should accept all 3,000 *ocr*. Dataset 3 contained 300 *keyword* and 3,000 *ocr*, where all *ocr* randomly assigned a *keyword* with one added spelling error (e.g. 'NullPointerException' became 'NullPointerExc3ption'). In this case, the filter should accept all 3,000 *ocr*. Datasets 1, 2, and 3 were constructed such that *keyword* terms and *ocr* candidate text strings were 15 characters in length (lowercase letters or numbers), with the goal of avoiding the problem with editdistance mentioned earlier, where the algorithm replaces one string with the other when comparing two short strings. Allowing the text strings to contain numbers models realistic data more closely. Dataset 4 contained 300 *keyword* and 3,000 *ocr*, where all *ocr* and *keyword* were composed of 5 randomly selected lowercase letters. In this case the filter should reject all 3,000 *ocr*. Finally, Dataset 5 contained 299 *keyword* and 3,000 *ocr*, with 496 of the *ocr* similar to any *keyword*.

Each dataset was processed 10 times using each filter implementation in each of the hyperparameter configurations listed in Tables 6.1 and 6.2. However, NeedlemanWunsch (ID 8) is not listed, as high execution times such as 634 seconds per iteration rendered the algorithm too slow to be useful as a filter. Across the 5 datasets ≈1 million observations were recorded for LazyJaroWinkler.

Tables 6.3 and 6.4 present the hyperparameter configurations for each filter implementation, selecting the observations where the number of results was closest to 3,000 for datasets 1 and 2, and the number of results was closest to zero for datasets 3 and 4. These top results were then sorted by execution time, and the lowest execution time observation for dataset 5 was selected. Each observation involved executing the algorithm ten times for each scenario (Datasets 1, 2, 3, 4, and 5). Table 6.5 contains the analysis of the observations, reporting the specificity, precision and recall for each dataset. Precision and recall were not calculated for datasets 3 and 4, as they cannot result correct matches ($TP + FP = 0$, and $TP + FN = 0$) and so the equations for these metrics result in a denominator of 0. The observations were collected on 10 Virtual Machines (VMs) within a cloud computing infrastructure and using task to VM assignment based on the identifier of the VM. Each VM was configured with SSD storage, 512MB RAM, and a 2GHz 64-bit Xeon® CPU. Results were stored by the worker thread on each VM into a centralized database which could then be analyzed further using SQL commands.

The complete dataset characterizing the solution space including the views extracting the best hyperparameter configurations is available for further study at [207].

Two broad groups emerge from the data: the first group with IDs {1, 2, 9, 10, 11, 12, 14} generated lower quality results, while the second group with IDs {3, 4, 5, 6, 7, 13} generated more promising results. In the first group, serious deficiencies in execution time, specificity, and/or recall are immediately apparent. Specifically, for IDs 9, 10, 11, and 14 the average execution times for Dataset 5 were above 5 seconds (See Table 6.3). Word embedding was slow due to ad-hoc model generation each time a string list arrives from the OCR. Also, the word embedding approach does not tolerate one-off spelling mistakes and so required a large corpus that well-characterizes the words to be compared. For IDs 1 and 2 the specificity for Dataset 4 was lower as a result of the weakness of the edit distance approach discussed in the prior art regarding short length strings (See Table 6.4). For ID 12 the recall on Dataset 2 was 0 as even small spelling errors were missed by the substring matching approach (See Table 6.5).

The second group scored well on execution time, specificity, and accuracy for Datasets 1 to 4, and so they should be compared based upon their execution time, specificity, and recall for the realistic Dataset 5. ID 13 (LazyJaroWinler) had the best execution time in the second group, while ID 7 (Jaro-Winkler) had the highest recall in the second group. LazyJaroWinler's aggressive approach in the outer loops resulted in the false rejection of a small amount of data from Dataset 2. Jaro-Winkler provided the highest recall (≈65%) with a low execution time (≈1.35s), while LazyJaroWinkler provided half the execution time (≈0.67s) at the expense of five sixths the recall (≈11%). By simply discarding randomly half of the input data, Jaro-Winkler can achieve a similar execution time to LazyJaroWinkler with a recall of 0% for the discarded half of the data and ≈65% for the second half, averaging to ≈33% recall. The advantage of LazyJaroWinkler over discarding data is that all data is processed and therefore the closest matches to the known classes of data are returned, whereas discarding data may miss strong matches. Randomly discarding samples from the input data mitigates this problem somewhat, as the classification task executes every second.

LazyJaroWinkler is a new approach that breaks from loops as soon as a similarity match is identified. Memoization limits the overall execution time, and when there is no match, most computation involves low level binary operations on integers. Memoization limits the overall execution time of Algorithms 2, 3, and 4. The takeaway advice to the designer of a filter for a text classifier is to first select an algorithm having high recall and specificity for extreme cases, and then to consider the trade-off between recall and

execution time when selecting a particular algorithm.

ALGORITHM 2: LazyJaroWinkler

Input: $LIMIT$; swb; $wshift$; $minsim$; $mindist$; Keywords that neural network was trained to classify are stored in $keywords.values()$, while the indexes of these entries are stored in $keywords.keys()$; OCR output text $ocrs$

Output: List of text strings to be classified by a neural network $DNNinput$

```
1  for each keyword ∈ keywords.values() do
2  |   binDict[keyword] ← binFreq(keyword)
3  |   freqDict[keyword] ← bin(binDict[keyword]).count(1)
4  end
5  for each ocr ∈ ocrs do
6  |   sw ← ocr.keepOnlyLetters()
7  |   swLen ← swb + Legth(sw)
8  |   for each keyword ∈ keywords.values() do
9  |   |   kLen ← Legth(keyword)
10 |   |   kLenS ← kLen << wshift
11 |   |   if NOT (kLen > swLen OR kLenS < swLen) then
12 |   |   |   if keyword == ocr then
13 |   |   |   |   DNNinput.append(ocr)
14 |   |   |   |   break
15 |   |   |   end
16 |   |   |   if similarity(keyword, ocr) > minsim then
17 |   |   |   |   distance ← binary(binDict[keyword] ⊕ binFreq(ocr)).count(1)
18 |   |   |   |   if distance <= mindist then
19 |   |   |   |   |   if jaro_winkler(keyword, ocr) >= LIMIT then
20 |   |   |   |   |   |   DNNinput.append(ocr)
21 |   |   |   |   |   |   break
22 |   |   |   |   |   end
23 |   |   |   |   end
24 |   |   |   end
25 |   |   end
26 |   end
27 end
```

ALGORITHM 3: Binary Letter Frequency Encoding. $0b111$, $0b011$, and $0b001$ are binary bit masks.

Input: Any text string for which a similarity will later be computed: *word*
Output: Integer representation of a binary field representing the frequency of each letter in *word*: *intermediate*

```
1  binFreq(word):
2      if word ∈ binDict.keys() then
3          return binDict[word]
4      end
5      letters = { a, b, ..., z }
6      intermediate ← 0, iteration ← 0, bits ← 3
7      for each letter ∈ letters do
8          count ← word.count(letter)
9          if count >= 3 then
10             intermediate ← intermediate|(0b111 << (iteration ∗ bits))
11         else if count == 2 then
12             intermediate ← intermediate|(0b011 << (iteration ∗ bits))
13         else if count == 1 then
14             intermediate ← intermediate|(0b001 << (iteration ∗ bits))
15         iteration ← iteration + 1
16     end
17     binDict[word] ← intermediate
18     return intermediate
```

ALGORITHM 4: Similarity. Computes the bit-level similarity between input text and a word from a neural network's lexicon.

Input: A word in a neural network's lexicon: *keyword*; A substring from the OCR text *ocr*

Output: Similarity score between 0 and 1

```
1  similarity(keyword, ocr):
2      if ocr ∈ binDict.keys() then
3          freqOCR ← binDict[ocr]
4      else
5          freqOCR ← binFreq(ocr)
6      same ← (binDict[keyword] & freqOCR)
7      if same ∈ cntDict.keys() then
8          sameCount ← cntDict[same]
9      else
10         sameCount ← binary(same).count(1)
11         cntDict[same] ← sameCount
12     return sameCount/freqDict[keyword]
```

Table 6.3: Mean ($\bar{x}$) and standard deviation (σ) of 10 execution time samples for each filter's most effective hyperparameter configuration.

		Dataset 1 (Exact matches) Exec. Time (s)		**Dataset 2** (One letter changed) Exec. Time (s)		**Dataset 3** (No matches) Exec. Time (s)		**Dataset 4** (No matches & short strings) Exec. Time (s)		**Dataset 5** (Screen capture) Exec. Time (s)	
ID	**Hyperparameters**	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
1	LIMIT=2	2.37	0.17	2.08	0.22	4.18	0.28	1.47	0.12	3.71	0.35
2	LIMIT=2	0.67	0.07	0.61	0.07	1.10	0.12	0.41	0.04	1.62	0.12
9	LIMIT=0.049; windowSize=1	0.05	0.01	9.87	1.45	11.83	1.67	13.08	3.03	11.71	1.11
10	LIMIT=0.007; windowSize=1	0.04	0.00	14.74	1.32	19.39	1.41	20.77	1.62	18.97	1.46
11	LIMIT=0.84	4.20	0.22	3.99	0.20	8.12	0.84	5.99	0.21	7.59	0.44
12	-	0.13	0.01	0.07	0.01	0.12	0.02	0.10	0.02	0.11	0.02
14	width=3; k=17	19.81	2.02	21.49	1.44	25.21	2.33	24.04	2.10	24.14	2.30
3	LIMIT=0.6	0.61	0.06	0.54	0.03	1.06	0.10	0.49	0.03	1.29	0.13
4	LIMIT=0.8	0.47	0.05	0.39	0.03	0.85	0.19	0.46	0.04	0.82	0.09
5	LIMIT=0.8	0.73	0.06	0.66	0.05	1.32	0.22	0.63	0.06	1.47	0.16
6	LIMIT=1; prefix_weight=0.2	0.47	0.05	0.46	0.15	0.83	0.12	0.48	0.04	0.81	0.09
7	LIMIT=0.8; long_tolerance=0	0.75	0.06	0.72	0.21	1.32	0.27	0.65	0.06	1.35	0.09
13	LIMIT=9; swb=1; wshift=1; minsim=0.7; mindist=6	0.79	0.09	0.70	0.06	1.55	0.26	1.27	0.12	0.67	0.14

Table 6.4: Mean ($\bar{x}$) and standard deviation (σ) for 10 samles counting the number of *ocr* that pass the filter for each filter's most effective hyperparameter configuration. The the same hyperparameter settings were used in Table 6.3.

	Dataset 1 (Exact matches)		**Dataset 2** (One letter changed)		**Dataset 3** (No matches)		**Dataset 4** (No matches & short strings)		**Dataset 5** (Screen capture)	
ID	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
1	3000.00	0.00	3000.00	0.00	0.00	0.00	465.90	21.47	65.00	0.00
2	3000.00	0.00	3000.00	0.00	0.00	0.00	465.90	21.47	65.00	0.00
9	3000.00	0.00	1551.50	326.21	490.80	98.62	27.30	7.32	331.20	68.05
10	3000.00	0.00	1661.70	348.77	390.50	44.12	13.90	8.56	329.60	11.82
11	3000.00	0.00	2997.00	6.75	2.40	1.65	154.20	11.68	237.00	0.00
12	3000.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	805.00	0.00
14	3000.00	0.00	2086.60	84.23	104.00	9.31	122.20	11.04	534.00	0.00
3	3000.00	0.00	3000.00	0.00	0.00	0.00	44.10	5.22	359.00	0.00
4	3000.00	0.00	3000.00	0.00	0.00	0.00	43.80	4.59	238.00	0.00
5	3000.00	0.00	3000.00	0.00	0.00	0.00	43.40	5.17	251.00	0.00
6	3000.00	0.00	3000.00	0.00	0.00	0.00	0.00	0.00	232.00	0.00
7	3000.00	0.00	3000.00	0.00	0.00	0.00	105.20	7.81	378.00	0.00
13	3000.00	0.00	2995.00	12.69	0.00	0.00	0.00	0.00	62.00	0.00

Table 6.5: Analysis of observations: The average specificity, precision and recall were calculated and are presented here. Specificity, precision and recall was 100% for all filter implementations for Dataset 1 and so those results are omitted.

	Dataset 2			**Dataset 3**			**Dataset 4**			**Dataset 5**		
ID	**Spec.** (%)	**Prec.** (%)	**Rec.** (%)	**Spec.** (%)	**Prec.** (%)	**Rec.** (%)	**Spec.** (%)	**Prec.** (%)	**Rec.** (%)	**Spec.** (%)	**Prec.** (%)	**Rec.** (%)
1	100.00%	100.00%	100.00%	100.00%	N/A	N/A	84.47%	N/A	N/A	98.56%	40.00%	8.75%
2	100.00%	100.00%	100.00%	100.00%	N/A	N/A	84.47%	N/A	N/A	98.56%	40.00%	8.75%
9	100.00%	100.00%	51.72%	83.64%	N/A	N/A	99.09%	N/A	N/A	93.45%	48.55%	56.23%
10	100.00%	100.00%	55.39%	86.98%	N/A	N/A	99.54%	N/A	N/A	94.34%	50.32%	52.19%
11	100.00%	100.00%	99.90%	99.92%	N/A	N/A	94.86%	N/A	N/A	95.56%	49.37%	39.39%
12	0.00%	0.00%	0.00%	100.00%	N/A	N/A	100.00%	N/A	N/A	94.52%	66.59%	99.33%
14	100.00%	100.00%	69.55%	96.53%	N/A	N/A	95.93%	N/A	N/A	89.49%	46.82%	84.18%
3	100.00%	100.00%	100.00%	100.00%	N/A	N/A	98.53%	N/A	N/A	92.75%	47.45%	59.60%
4	100.00%	100.00%	100.00%	100.00%	N/A	N/A	98.54%	N/A	N/A	96.00%	54.62%	43.77%
5	100.00%	100.00%	100.00%	100.00%	N/A	N/A	98.55%	N/A	N/A	95.78%	54.03%	45.12%
6	100.00%	100.00%	100.00%	100.00%	N/A	N/A	100.00%	N/A	N/A	97.30%	68.53%	53.54%
7	100.00%	100.00%	100.00%	100.00%	N/A	N/A	96.49%	N/A	N/A	94.15%	57.64%	72.39%
13	100.00%	100.00%	99.83%	100.00%	N/A	N/A	100.00%	N/A	N/A	**99.37%**	**72.58%**	**15.15%**

Filtering out text that is dissimilar to the detected contexts is essential to lowering the execution time of AVRA, specifically the time between a client's request to process an image, and the response from the server. In this section the execution time of the filtering algorithm is considered in comparison to a state of the art approximate edit distance algorithm programmed to perform the same text filtering task.

In one example with only 8 true positive keywords, LazyJaroWinkler executing on a corei7 assessed 2957 candidate text snippets in 0.47 seconds and kept 8 matches in total, or %0.3 of the input text. The algorithm compared each candidate with up to 299 DNN *keywords*. As shown in Table 6.6, an implementation of the edit distance algorithm described in [96] was used for comparison to the state of the art in edit distance calculation [231]. Comparing each of the 2957 candidate text snippets to find at least one close similarity to one of the 299 DNN *keywords* completed executing in 4.4 seconds. The incremental improvement in execution time seen in Table 6.6 indicates that the speedup achieved by LazyJaroWinkler is not a result of only one algorithm modification from Jaro-Winkler. Rather, several unrelated optimization techniques combined resulted in the observed speedup of almost 10 times compared to the implementation using state of the art techniques. LazyJaroWinkler's low latency is a product of the fact that the vast majority of the computation when there is no match involves memorized results from dictionaries, and low level binary operations on integers. These low level operations such as AND, OR, SHIFT, and XOR are executed extremely quickly. The algorithm requires no sorting and breaks from loops as soon as a similarity match is identified. Memoization for partial results with dictionaries reduced the execution time of LazyJaroWinkler and its subcomponents Algorithm 3 and Algorithm 4.

6.2.1 Filter Sensitivity to Hyperparameter Changes

It is interesting to consider how sensitive the text filtering algorithms are to the hyperparameter settings. For example, how important is it that LazyJaroWinkler have a minimum similarity setting of 0.7 rather than 0.8 or 0.6? For algorithms with one hyperparameter such as Jaro (ID 5), an objective function can be defined $Obj = a+b+6000-c-d$, where a is the number of results that pass the filter for Dataset 1, b is the number of results that pass the filter for Dataset 2, c is the number of results that pass the filter for Dataset 3, and d is the number of results that pass the filter for Dataset 4. A score of 12,000 for the objective function indicates that the filter accepted all of the exact or near matches in Datasets 1 and 2, while rejecting Datasets 3 and 4 where there are no

Table 6.6: Comparing the binary operation focused text filtering from Chapter 6 with [231], an implementation of the edit distance algorithm described in [96]. These data were collected using 2,957 candidate text snippets filtered based upon 299 DNN *keywords*.

Match (%)	**Match (#)**	**Correct match (#)**	**AVRA filter execution time (s)**	**Most recent optimization**	**Speedup relative to editdistance (4.4 s)**
1.3%	40	8	28.83	Added dictionary binDict	0.15
1.3%	40	8	15.43	Dictionary improvement (binary operations)	0.29
1.3%	40	8	9.50	Penalize candidate text if too short	0.46
1.3%	40	8	9.35	Loop optimization	0.47
1.3%	40	8	1.79	2nd dictionary memorizes binFreq inputs	2.46
1.3%	40	8	1.02	Increase penalty for text if too short	4.31
1.3%	40	8	0.88	Penalize candidate text if too long	5.00
1.3%	40	8	0.78	Further optimize similarity function vars	5.64
0.3%	8	8	0.46	Filter if longest common substring too small	9.57

matches. The relationship for Jaro between $LIMIT$ and the objective function is show in Figure 6.1. The fitness of the filter is easy to see because of the two dimensional nature of the data. There is clearly a maximum point at 0.8 where Jaro is configured best. The execution time impact of the hyperparameter on Jaro is observed in Figure 6.2.

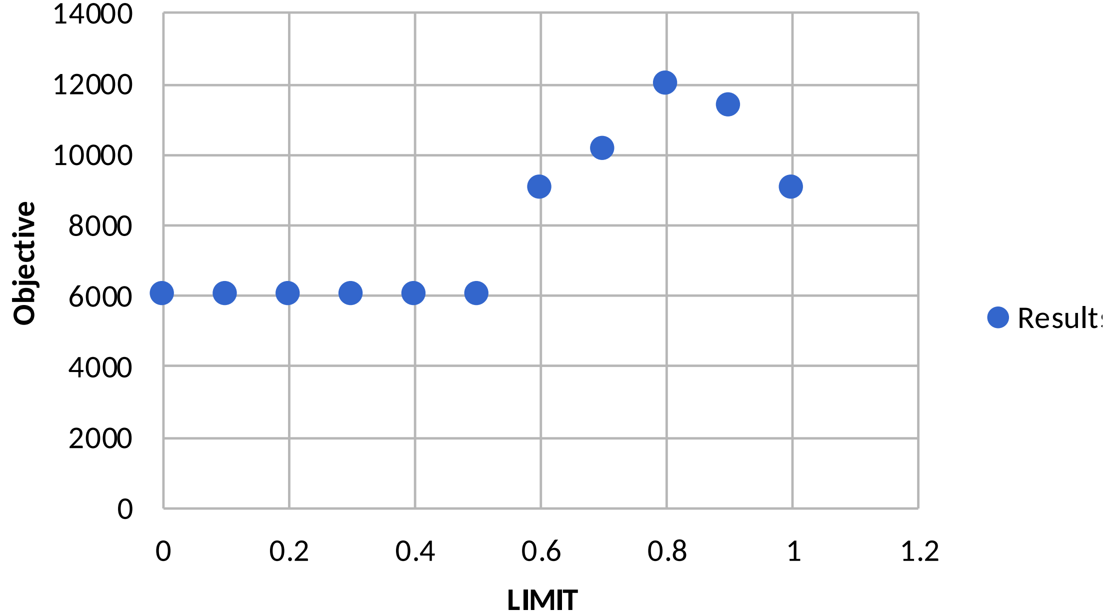


Figure 6.1: Jaro hyperparameter impact: Objective vs. LIMIT

The trade-off between execution time and filter quality can be decided using the data in Figures 6.1 and 6.2 for Jaro, but can this approach extend to LazyJaroWinkler which contains many hyperparameters? Imagine that for each hyperparameter a graph is plotted to visualize the relationship of the hyperparameter with the execution time and the overall filter quality. These parameters affect each other, and so the trend for one hyperparameter indicates only general first order trends, and does not capture multivariable relationships. A correlation matrix is a better measure of the strength of the observed correlations. The heat map of Figure 6.3 presents the Pearson correlation coefficient for the recorded LazyJaroWinkler data.

LazyJaroWinkler contains hyperparameters LIMIT, swb, wshift, minsim, and mindist. The bottom right quadrant of the heat map reveals that the hyperparameters do not strongly affect each other. As expected, the top left quadrant reveals that the data-source (which dataset is processed by the filter) is correlated with the number of results

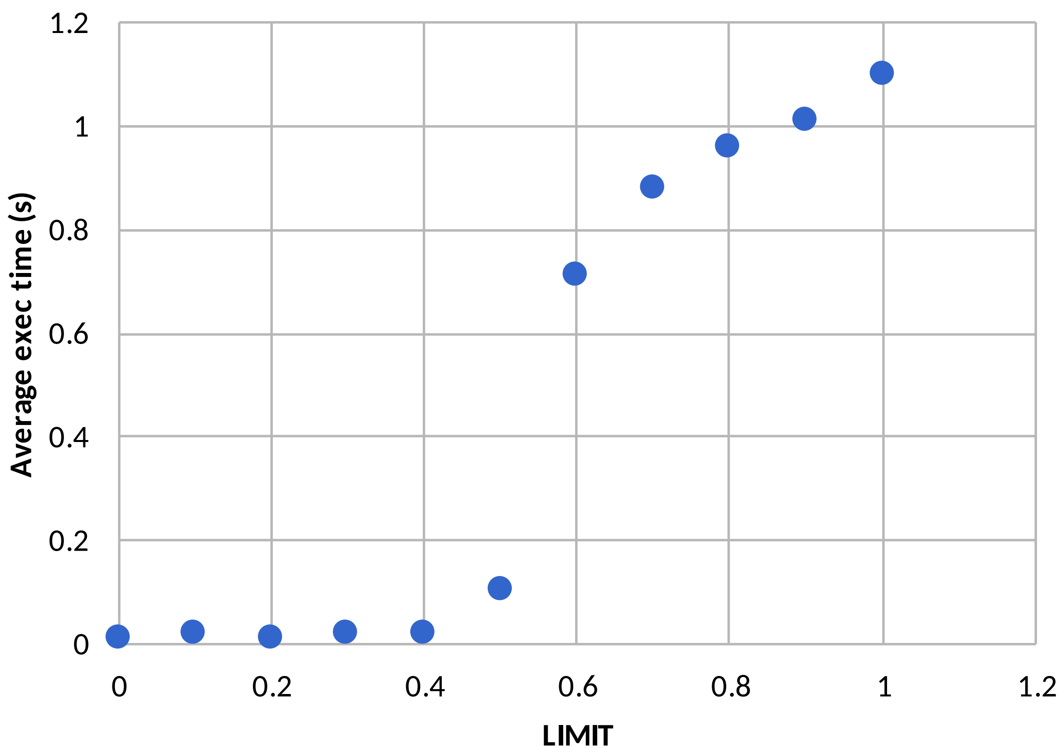


Figure 6.2: Jaro hyperparameter impact: Execution Time vs. LIMIT

and the execution time. The bottom left quadrant reveals that wshift and minSim have a strong impact on execution time. It also shows that swb, wshift, and minDist have the strongest impact on the number of results of the 5 hyperparameters. It is interesting to note that the least sensitive hyperparameter is LIMIT. Perhaps because it is only used in the innermost loop of the filter it has less impact on the execution time.

To get a sense for how the points in the LazyJaroWinkler design space cluster, t-SNE 2D dimensionality reduction ([202] [130]) was applied to a 1% random sample of the collected data for 100,000 iterations with a perplexity of 30. 10% of the data were used as a testing dataset with the remaining 90% used as the training dataset. As shown in Figure 6.4, 10 classes were learned from the data, and clusters appeared to form.

6.3 Chapter Summary

This chapter described a text filtering approach to accelerate text classification. This acceleration was achieved by discarding text that was not similar to a set of known *keywords*. Several implementation options were modeled in order to identify a filter that understands short strings, small spelling errors, matches and mismatches, and has low latency with high specificity. This push for low latency arises from the thesis statement which claims that action recommendations can be provided within a reasonable response time.

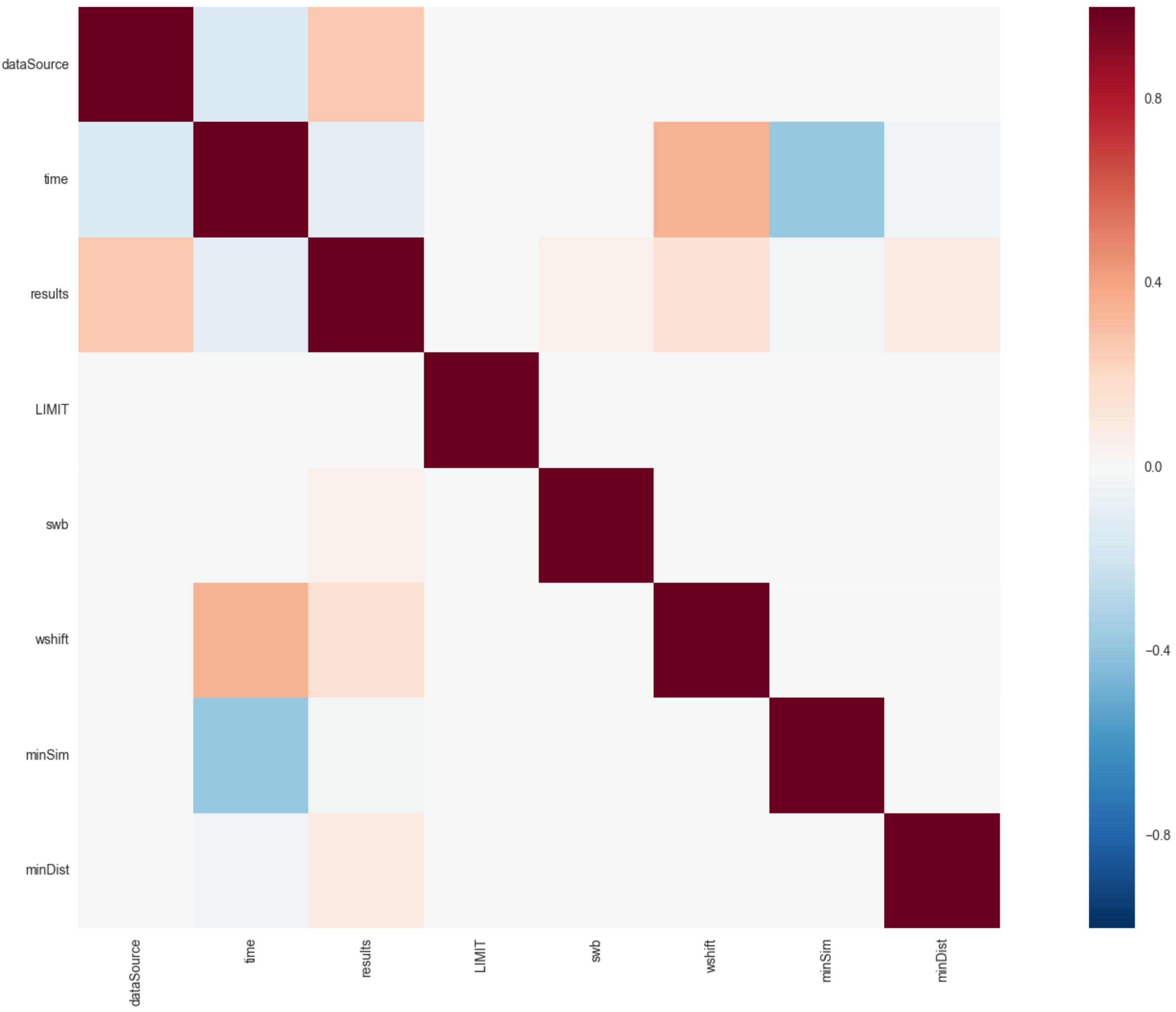


Figure 6.3: Heat map of the Prearson coefficient correlation matrix for LazyJaroWinkler

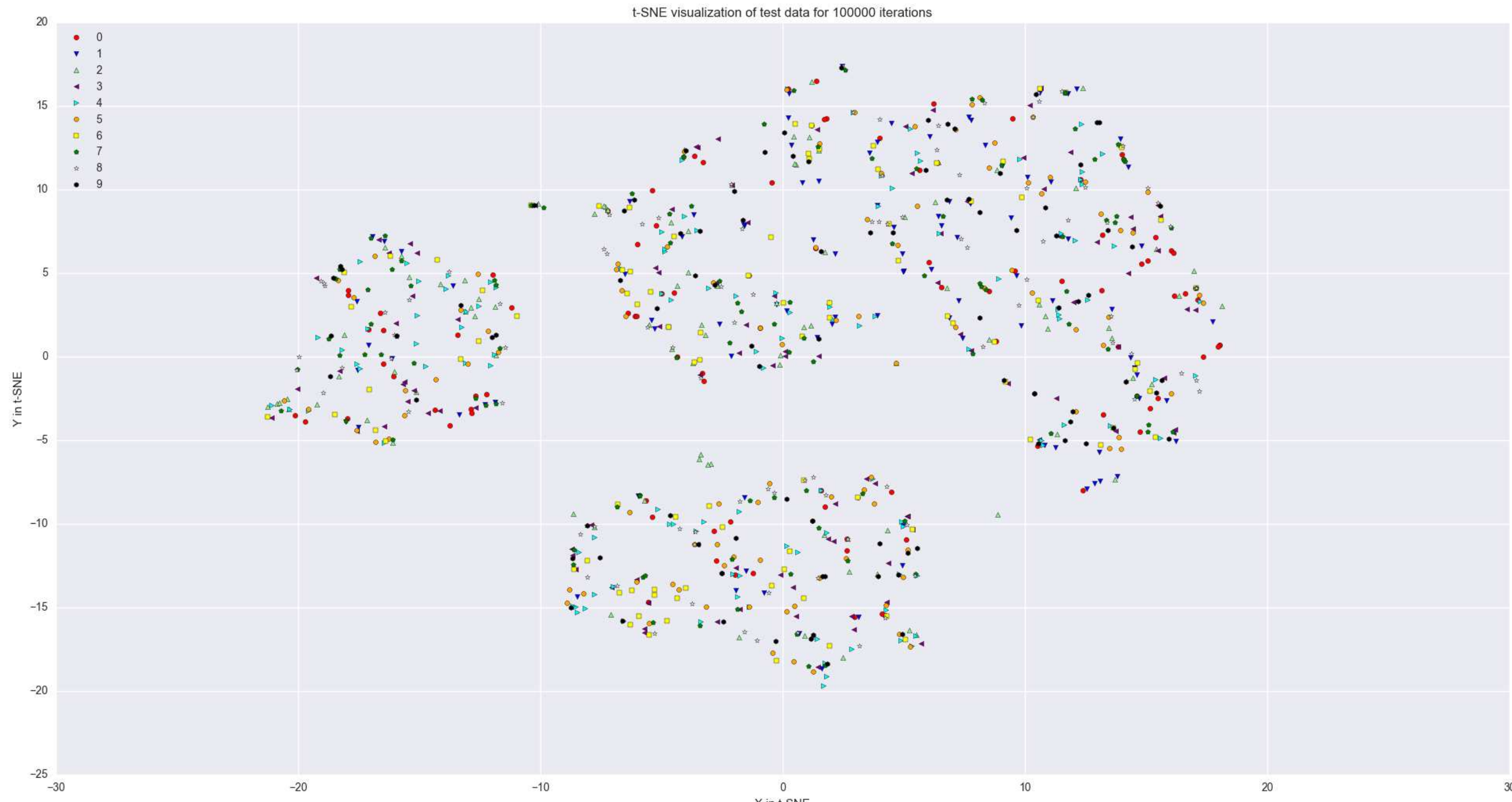


Figure 6.4: Clusters in a t-SNE 2D dimensionality reduction of the LazyJaroWinkler design space

Chapter 7

Classification of OCR Output

This Chapter describes the recognition of *keywords* in noisy text. OCR introduces spelling errors into the onscreen text analyzed by AVRA, and so a classifier is trained to recognize mistakes made by the OCR software for the keywords it is meant to detect. This chapter compares several options for implementing an OCR output classification system, and settles upon a particular DNN implementation. Using the recognition of error messages within a set of generated images as an example, a baseline measurement of OCR output recognition accuracy is presented in Section 7.1.1, where an exact match between the OCR input and output text is counted as a success and any other text produced is a failure. Dictionary and spell-check-based approaches achieved up to 88% classification accuracy (Sections 7.1.2, 7.1.3 and 7.1.4), while various deep learning approaches achieved up to 96% accuracy, and a combination of deep learning with a dictionary achieved 97% accuracy (Section 7.1.5). The chapter concludes with a description of the supervised learning process for the DNN (Section 7.2).

Recognizing variants of terms (for example using stemming and/or lemmatization [199]) can reduce the number of exactly equivalent *keywords* learned by AVRA. Although this is a useful idea, it was not implemented for this work. Other standard NLP approaches used to facilitate word embedding such as removing common words from the text (removing stop words [199]) do not address this particular recognition problem as the noise in the text is at the character level rather than the word level, and understanding the sequence of tokens as an intent is not the goal here. The text for two adjacent windows can appear jumbled together into a mixed up paragraph including artifacts from the image converted into text that is not really there. What emerges from the OCR is a long string of text that may contain specific keywords. Rather than maxi-

mizing the accuracy of OCR output recognition for all onscreen text, or understanding what the text means, the goal here is to maximize the accuracy for specific *keywords*. OCR misspellings are obtained by generating many images containing a given *keyword* (e.g. `nullpointerexception`), extracting text from the generated images using OCR, and then comparing the extracted text with the original error message text to learn the misspellings one might expect to see in the OCR output. Normalization of keywords using equivalence classes requires explicit knowledge of the related terms [199], which are misspelled versions of the correctly spelled keyword. In this work a neural network is used so that it can intuit spelling corrections from data outside of the training data, whereas normalization can only associate terms it knows about.

7.1 Supervised Keyword Learning

An error message is "text that is displayed to describe a problem that has occurred that is preventing the user or the system from completing a task." [143]. A simple approach to detecting onscreen error messages is extracting text from a screen capture using OCR, and then parsing the OCR output text to detect error message text. This approach works in principle, but does not work well in practice with error messages because the OCR conversion from image to text often includes misspellings. Unfortunately, OCR processing of error message text is more prone to these translation errors than standard English text. It is demonstrated here that OCR accuracy on error messages is lower than general English text, and it is shown that even in a state of the art OCR system this misspelling of error message terms takes place.

For example, OCR of natural language text shown in Figure 7.1 (A) resulted in the output text of Figure 7.1 (B). Clearly the onscreen text was captured correctly. However, consider the OCR of error message text from a terminal window shown in Figure 7.2 (A) resulting in the output text of Figure 7.2 (B). The OCR system extracted the error message text. However, the OCR text contains a spelling error in the most important *keyword* `nullpointer3xception` used by the virtual agent to detect the presence of the specific error message. This seems at first glance like a small problem, but in fact this misspelling of `nullpointerexception` causes the virtual agent to miss that there is an onscreen error.

The goal in this work is to learn a mapping from incorrect OCR output words (e.g. `nullpointer3xception`) back to the correct words (e.g. `nullpointerexception`) that was present on the computer screen, thereby improving error message recognition. Es-

Deep learning

From Wikipedia, the free encyclopedia

For deep versus shallow learning in educational psychology, see Student approaches to learning.

Deep learning (also known as **deep structured learning**, **hierarchical learning** or **deep machine learning**) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.[1][2][3][4][5][6][7][8]

Deep learning is part of a broader family of machine learning methods based on learning representations of data. An observation (e.g., an image) can be represented in many ways

(a) A cropped image of a webpage containing natural language [257]

```
deep learning from wikipedia, the free encyclopedia for deep versus
shallow learning in educational psychology, see student approaches
to learning. deep learning (also known as deep structured learning,
hierarchical learning or deep machine learning) is a branch of
machine learning based on a set of algorithms that attempt to model
high-level abstractions in data by using multiple processing layers,
with complex structures or otherwise, composed of multiple non-linear
transformations.[1h2h3h4h5h6h7h8] representations of data. an observation
(e.g., an image) can be represented in many ways.
```

(b) OCR output

Figure 7.1: Example of OCR correctly processing natural language

sentially, this system must learn to detect and correct mistakes in OCR output. This is accomplished by comparing OCR input and output for specific *keywords* (in this case error message text) to characterize many of the possible misspellings that may be commonly produced by OCR for the *keyword* in question. This information can then be used as a basis for correcting OCR output.

```
Exception in thread "main" java.lang.NullPointerException
    at makeSomeErrors.makeerror.doSomething(makeerror.java:11)
    at makeSomeErrors.makeerror.main(makeerror.java:7)
```

(a) Error message sent to OCR system

```
exception in thread "main" java.lang.nullpointer3xception
at makesomeerrors.makeerror.dosomething(makeerror.java:11) at
makesomeerrors.makeerror.main(makeerror.java:7)
```

(b) OCR output

Figure 7.2: Example of OCR incorrectly processing an error message

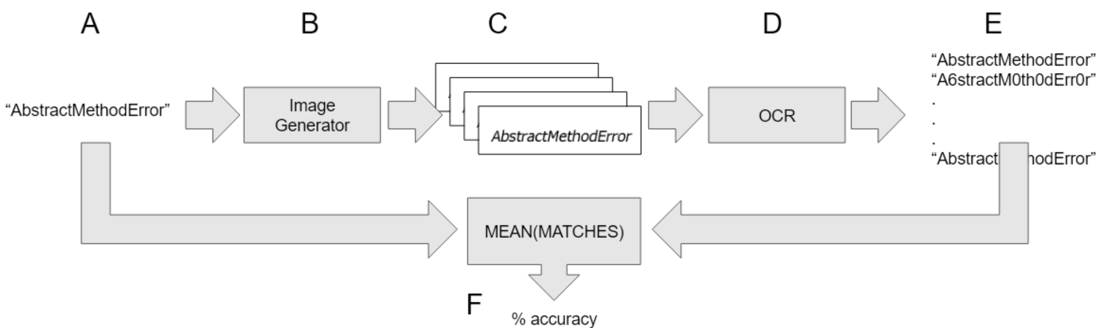


Figure 7.3: System for recording and correcting OCR mistakes.

Recording OCR Mistakes

Consider the block diagram in Figure 7.3 for recording OCR mistakes. An image generator (B) generates many images of various font settings (C) for one specific term (A). The system then submits the images for OCR processing (D), optionally performs corrective operations on the OCR output (E), and then compares the accuracy of the output with the original text input (F). OCR at (D) uses the pytesseract wrapper [91] for the tesseract-OCR engine [216] [111] to convert the image back into a string of text (E). In this system the OCR is treated as a black box (D), and the behavior of this black box is characterized through the recording of its input (A) and output (E).

The image generation subroutine depicted in Figure 7.3 (B) takes in text (e.g. `AbstractMethodError`) and outputs a grayscale PNG image containing that text. The system selects a random vertical and random horizontal position on the screen to display the text, ensuring that it is visible and therefore not cut off. The image size is always 1024 x 786 pixels with a white background and black text. The word style is selected randomly as either bold, italic, or plain. The font size is selected randomly as a value between 10pt and 40pt. The font is selected randomly from the following set of fonts: Arial, Verdana, Arial Narrow, Century, Courier, Courier New, Dialog, DialogInput, Futura, Garamond, Helvetica, Impact, Lucida Sans, SansSerif, Serif, Tahoma, Times, and Times New Roman.

OCR Accuracy Decreases for Error Message Text

The system of Figure 7.3 was used to compare tesseract-OCR accuracy recovering natural English language words within images compared to processing error message text [216]. Two sets of words were created to represent the two domains to be evaluated:

Charter Word List (CWL) representing natural English text, and Java Error List (JEL) representing error messages. CWL contains the set of each word in the Canadian Charter of Rights and Freedoms [9] cast to lowercase and excluding numbering, indices, braces, and special characters. CWL contains 541 natural English language words such as "imprisonment". JEL contains a set of error messages produced by the Java Virtual Machine (JVM). JEL contains 248 terms cast to lowercase such as "indexoutofboundsexception".

The average word length in CWL (representing common English words) is 7.27 characters, while the average word length in JEL (representing specialist terms) is 21.02 characters.

10 images were generated for each term in CWL and JEL using the image generation subroutine described above. Multiple images were generated for each term in order to sample the OCR output under various conditions (font, font size, etc.), resulting in a variety of OCR output texts. pytesseract was used to convert each image back into text using OCR. Processing of JEL resulted in 1,485 exact matches out of 2,480 tests, while processing CWL resulted in 4,567 exact matches out of 5,410 tests. These tests reveal that OCR processing of error message text (JEL) had lower accuracy (59.9%) than OCR processing of natural text (CWL) which had higher accuracy (84.4%).

Testing and Training Datasets

Using the term list JEL described above, testing and training datasets were created using the image generation subroutine. The training dataset TRAINING contained 100 images per JEL term, for a total of 24,800 images and corresponding error message terms. The dataset TESTING contained 10 images per JEL term, for a total of 2,480 images and corresponding terms. Each dataset contained the original text that each picture was based upon, as well as the text produced by the OCR process.

Regarding the dataset content, TRAINING contained 5,269 distinct OCR results associated to the 248 distinct terms in JEL. There were therefore, on average, 21.2 variants of each term produced by the OCR output. Some common substrings in JEL were "illegal" (10 terms), "error" (23 terms) and "exception" (189 terms). Note that 14 data points (0.56%) in TESTING had blank OCR text as a result of the OCR software detecting no text in the image. The theoretical maximum possible classification accuracy is therefore 99.44%. When the deep learning system described in Section 7.1.5 was trained and tested using the TESTING dataset, the classification accuracy was in fact 99.35%.

One factor impacting the time required to extract text from an image is the amount of text in the image. The more text an image contains, the longer the OCR module

will require to process the image. However, there appears to be additional information contained within the execution time of the OCR module. When processing TRAINING images, the average time required to process an image was 0.34 seconds. Incorrect OCR output required on average 0.39 seconds to execute, while correct output from the OCR required only 0.31 seconds to execute. One can therefore conclude that when text length is taken into account, the OCR output accuracy is likely higher if the OCR processing time was shorter than the running mean of the OCR processing time, and likely lower if it is less than the mean processing time.

7.1.1 Baseline Keyword Recognition

A baseline for OCR accuracy on the detection of error message words is to measure exact matches between OCR input and output text, and to calculate the resulting accuracy. As recorded in Table 7.1, the default OCR configuration had 61% accuracy processing the TESTING dataset. This result is close to the observation for JEL, where processing 2,400 error message images resulted in a 59.9% accuracy.

The tesseract-OCR software configuration can be tuned in various ways to improve OCR accuracy on non-dictionary terms. This approach is recommended in the OCR documentation [156]. Accuracy when disabling the dictionaries by setting the load_system_dawg and load_freq_dawg variables to false is reported in Table 7.1 as "Disable Dict". Disabling the dictionaries did not improve the OCR accuracy. In another attempt, additional dictionaries were disabled (load_punc_dawg, load_number_dawg, load_unambig_dawg, load_bigram_dawg, load_fixed_length_dawgs) with results reported in Table 7.1 as "Disable All Dict". Disabling the OCR dictionaries was not an effective strategy for improving OCR accuracy when processing error message text from TESTING. These results leave significant room for other approaches to provide improvements in accuracy.

7.1.2 Supervised Keyword Learning with Memoization

Memoization can be applied to improve error message recognition in the OCR output. The memoization dictionary first records a set of OCR output correction rules based upon OCR of a dataset of terms, and then it is tested against a second dataset to measure the effectiveness of the approach. This approach learns which spelling mistakes the OCR module is prone to make, and stores them in relation to the correct spelling for use after the training phase.

Table 7.1: Classification accuracy for the baseline OCR system on the TESTING dataset.

	CONFIGURATION	**FAIL**	**PASS**	**TOTAL**
Exact match	Default	965	1,515	2,480
Exact match %	Default	39%	61%	61%
Exact match	Disable Dict	996	1,484	2,480
Exact match %	Disable Dict	40%	60%	60%
Exact match	Disable All Dict	1,000	1,480	2,480
Exact match %	Disable All Dict	40%	60%	60%

Using the TRAINING dataset, a dictionary D was trained to store outputs from the OCR process as keys, and text inputs to the image generation process as values. These key/value pairs map incorrect OCR outputs back to the correct output word. The benefit of this approach is that the learning algorithm executes very quickly (seconds) compared to neural network training (hours or days), whereas the drawback of this approach is that it cannot generalize to identify new misspellings that were not observed during the training phase.

To test the accuracy of this dictionary-based approach the dataset TESTING was sent into the OCR tool and the dictionary trained on the TRAINING dataset corrected the output of the OCR tool in cases where the OCR output matched a key in the dictionary. When there was a key match, the corresponding dictionary value was substituted for the OCR output.

The results of this test are presented in Table 7.3. The 88% accuracy is a good improvement over the 61% baseline established in Section 7.1.1. This high accuracy makes it clear that most OCR output errors are repeatable as a given term is most likely to be misinterpreted in a particular way. Furthermore it is clear that more than 1 in 10 results from the OCR module is generated by rare or unpredictable circumstances that a dictionary-based approach cannot hope to solve. This type of OCR error can be generated when the text is touching the edge of the frame, is distorted, is in an unusual font, or a variety of other situations.

Another interesting result from this test was the presence of dictionary key collisions. Even with TESTING's small 248 term lexicon of values, there were several cases where two input terms resulted in the same OCR results in the training dataset, causing a key conflict between two terms. Both terms expect to use the same key, but keys cannot be shared between terms. Specific examples of these collisions are presented in Table 7.2.

Table 7.2: Examples of collisions in the dictionary keyspace resulting from OCR producing the same output for two or more different terms.

OCR Output Text	Generated by Term	Also Generated by Term
nexceptiun	unsupportedoperationexception	indirectionexception
texceededexception	sizelimitexceededexception	limitexceededexception
mum	awterror	security violation

Table 7.3: Classification accuracy for the dictionary-based approach to correcting OCR output text. The dictionary was trained on the TRAINING dataset and tested on the TESTING dataset.

	FAIL	PASS	TOTAL
exact or regex match	296	2,184	2,480
exact or regex match %	12%	88%	88%

Key collisions were not important to the accuracy on the TESTING dataset, as only one key collision was present among the 2,480 terms.

7.1.3 First Spell-check Approach

An OCR output correction system was developed in this work based upon the ideas in [26] to correct OCR output text using the Google search engine's built-in spell-check correction. The OCR output text was submitted to Google's search engine, and if the 'Showing results for' field appeared in the results page, the term Google expected replaced the OCR output text. This system correctly output 1,721 terms out of 2,480 (69.40% accuracy) when processing the TESTING dataset.

7.1.4 Second Spell-check Approach

A second OCR output correction system was implemented to further evaluate the effectiveness of correcting spelling mistakes in OCR output text. The spell-check module [138] was employed to select the best candidate correction from a variety of ranked options including known misspellings of words, dictionaries of words, word lists generated by parsing natural language documents, and word snippets. The ranking was based upon word use frequency in the reference document, preferring more common terms over less

common ones.

Without modification, this autocorrect implementation works well on correcting misspelled English words. For example, 'Exc6ption' is corrected by autocorrect to 'exception', and 'speling' is corrected to 'spelling'. However, error message text is not corrected, as it is not included in the built-in library of words and texts. For example, 'NullPointerExc6ption' is returned unmodified as 'NullPointerExc6ption' rather than the correct answer 'NullPointerException'. The system could not correct any malformed OCR output from the 24,800 examples in the TRAINING dataset, performing equally in terms of accuracy as the baseline case from Section 7.1.1. To improve these results, the autocorrect dictionary was updated to include the corpus of terms from TRAINING into its database. After this upgrade the system correctly output 1,993 terms out of 2,480 (80.36% accuracy) when processing the TESTING dataset.

Some examples of successful autocorrect modifications of the OCR output were:

- `userexceplion` contains one letter substitution mistake and was corrected to `userexception`
- `i/legalsta teexception` contains two mistakes (one letter substitution and one added space) and was corrected to `illegalstateexception`
- `abstractmethodenor` contains a mistake where two characters were subsumed into one incorrect character, and was corrected to `abstractmethoderror`

The downside of this spell-check approach using autocorrect is the time required to correct words is very long. Correcting each of the examples in the TESTING dataset required 0.79 seconds per spelling correction. For just five onscreen keywords the latency would no longer provide a reasonable response time.

7.1.5 Supervised Keyword Learning with Deep Learning

This section continues with a brief discussion on the DNN classifier developed for this work, followed by an explanation of the input and output vector encoding, and the classification accuracy results for each encoding scheme. Several possible input vector encoding schemes considered.

The string output from the OCR system must be converted into a representation that is well suited to classification by a deep learning neural network. Identifying good features involved the development of incremental improvements sometimes referred to as "feature

engineering". Specifically, binary ASCII and letter frequency encoding was investigated as well as Morse encoding and memoization. These features, when combined, are quite powerful at correcting OCR output errors. With further augmentation by a dictionary, the classifier detected the vast majority of onscreen error messages, failing only in cases where the message information was lost during the image to text conversion process.

The DNN is trained by processing an image of the *keyword* text through OCR software (generated using a text-to-image generator), and then encoding the results into vectors that include letter frequency information. This approach of *keyword* recognition with DNNs robustly detects *keywords* even in the presence of OCR output spelling errors and frame-shifts in the text.

The deep learning software package used in this work is called theano [31] [27]. The specific deep learning algorithm used in this work is a customized branch of image processing code [183], originally designed to recognize handwritten characters in the MNSIT dataset [122]. This algorithm included numerically stable softmax for the output layer, gradient scaling, improved training with dropout, improved training with noise injection, and more.

In this work a 3 layer deep learning neural network was implemented in theano to improve the classification of text produced by an OCR tool [183]. The original image processing deep learning system was modified for this work to increase the number of input neurons, and to accept text rather than image data. The output layer contains 5000 neurons. Each output neuron encodes for a specific term or is unused. Output vectors in TESTING and TRAINING are one-hot vectors. This gives the network the ability to map many millions of possible input vectors (representing OCR text) onto up to 5000 specific terms. These terms can be added dynamically as the system learns new terms. The input layer contains 784 neurons, and the hidden layer contains 625 neurons. The number of inputs, 784, is an arbitrary number left over from the original purpose of the code, which was processing image data (28 pixels by 28 pixels). Training of the network is accomplished over 150 epochs. Many experiments in this work were performed at three different levels of TRAINING dataset uniform random sampling: either 10%, 20%, or 100%. Using less of the dataset in training helps the model to complete the training phase faster at the cost of predictive performance after the training. Using too much training causes the model overfit to the training data and then underperform when classifying the TESTING dataset.

The input vector encoding for a deep learning system is crucial to the predictive success of the network. In this Section input vector encoding is discussed. Characters

in the output of the OCR system are encoded into an array representing the characters of the OCR output as inputs to the deep learning neural network. The neural network is trained in a similar way to the dictionary discussed in Section 7.1.2. Namely, the independently generated datasets TRAINING and TESTING are used to train and then test the system.

Each distinct term in the datasets (TRAINING, TESTING) was encoded as a 1-hot vector unique to that *keyword*. These 1-hot vectors are the classes that the neural network is trained to recognize. The images based upon these terms are the inputs to the OCR, and the outputs of the deep learning system are the 1-hot output vectors. Because of the sparse nature of 1-hot vectors, they are stored as integers signifying the integer index into the 1-hot vector where the only '1' in the vector is located. Input vectors in TRAINING are created using the encoding of the OCR output. During each epoch, the system is trained on TRAINING and then tested against TESTING.

Each element in both datasets contains vector representations of terms using 1-hot output vectors, and encoded OCR text as input vectors. These mappings between input vectors and output vectors translate from the plaintext of OCR to a representation that the neural network is trained to understand.

Input Vector Encoding

Deep learning input vectors can be encoded as pixels in a grayscale image. Data in the array representation of a term is left-aligned onto the input vector. At this stage various alignment schemes for the input vectors were attempted with poor results, including random alignment and random amounts of 0-padding both sides of any term less than 784 elements long. The initial approach of left alignment of the data produced better results and so it was retained. If the data is less than 784 character long, then it is right-padded with 0s to become length 784. If the data is longer than 784 characters then the data is truncated to fit into the array.

Deep learning input vectors can be encoded to increase the number of array elements/cells/pixels encoded by a character (increased number of neurons activated) and to increase the contrast of the input vectors during training (array pixel elements contain only the values 0 or 255).

The input vector representations considered here are: Naive ASCII encoding (Ascii), Binary ASCII encoding (BinAscii), Morse encoding (Morse), Morse with letter frequency encoding (MorseFreq described in Section 7.1.5), and Binary ASCII with letter frequency encoding (BinAsciiFreq). Finally, the dictionary approach from Section 7.1.2 was com-

bined with the vector encoding from BinAsciiFreq with results reported as BinAsciiFreqDict.

Naive ASCII encoding (Ascii)

This encoding involves assigning each pixel in an array of length 784 and height 1 with the ASCII value for a character. The encoding at each element maps the ASCII number of the corresponding character to the cell in the array at the same index. Figure 7.4 shows an example of this encoding scheme. The following is an example of an ASCII input vector for the term "socketexception". The text string is broken down into a vector containing the characters ['s', 'o', 'c', 'k', 'e', 't', 'e', 'x', 'c', 'e', 'p', 't', 'i', 'o', 'n']: [115, 111, 99, 107, 101, 116, 101, 120, 99, 101, 112, 116, 105, 111, 110, 0, 0, 0, 0, ..., 0]

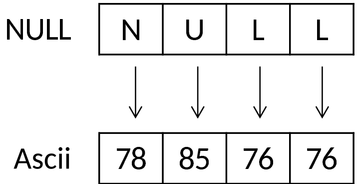


Figure 7.4: Conversion from text into input vectors using naive ASCII encoding.

As reported in Table 7.4 the naive ASCII encoding scheme was formatted in a way that the deep learning neural network was not able to learn well. The input vectors of the training data were difficult for the system to differentiate. The TRAINING and TESTING datasets included 248 classes, and so a complete failure to learn to differentiate between classes would register as (1/248), or 0.4% accuracy. A random output from the 5000 output neural network would result in a 1/5000 chance of correctly identifying a particular class. The results for the naive ASCII encoding were poor (14%) and not usable as a real-world classifier, but clearly some learning did take place.

Binary ASCII encoding (BinAscii):

To improve the perceptron processing of the information contained in the ASCII input to the neural network, the data in each ASCII character was converted into binary and then into pixels. Figure 7.5 shows an example of this encoding scheme. For example, the letter 'a' has ASCII code 55, which is '110111' in binary. This binary string is then converted into the following pixel string: "255, 255, 0, 255, 255, 255". These pixel strings

were then concatenated into a vector representing the original text coming from the OCR module. The following is an example of a Binary ASCII input vector representing the term "socketexception": [255, 255, 255, 0, 0, 255, 255, 255, 255, 0, 255, 255, 255, 255, 255, 255, 0, 0, 0, 255, 255, 255, 255, 0, 255, 0, 255, 255, 255, 255, 0, 0, 255, 0, 255, 255, 255, 255, 0, 255, 0, 0, 255, 255, 0, 0, 255, 0, 255, 255, 255, 255, 255, 0, 0, 0, 255, 255, 0, 0, 0, 255, 255, 255, 255, 0, 0, 255, 0, 255, 255, 255, 255, 0, 0, 0, 0, 255, 255, 255, 0, 255, 0, 0, 255, 255, 0, 255, 0, 0, 255, 255, 255, 0, 255, 255, 255, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 0, ..., 0, 0, 0]

This binary data format is easier for the neurons to learn because each letter that reaches the neural network input is encoded as a series of ON/OFF signals, rather than discrete or continuous normalized signals containing level information. Neural networks are well suited to transform binary input data into binary output data, as the activation function outputs a binary signal at the output of any neuron. The DNN should learn to recognize letters with a multi-neuron pattern at the input to the DNN. When training a DNN, each level of the neural network is tasked with encoding features based upon the previous layer's feature representations. Because crisp inputs in the training data (ON or OFF) are easier to learn features from in the earliest training layer (the input layer), binary input data trains the DNN better than continuous inputs. Similarly during backpropagation, it is easier to converge to crisp ON/OFF signals at the input than to learn to reconstruct the inputs at some specific level for each neuron. The information density of the binary representation is lower, but the signal is much clearer to learn from.

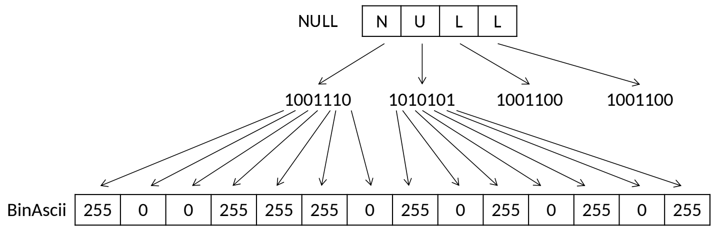


Figure 7.5: Conversion from text into input vectors using binary ASCII encoding.

The classification accuracy of binary ASCII encoding is reported in Table 7.5 and Figure 7.7 under the labels BinAscii10, BinAscii20, and BinAscii100. The binary ASCII encoding scheme was very good at training the deep learning neural network to classify OCR output text. The maximum classification accuracy achieved with binary ASCII encoding was 92%.

Morse encoding (Morse):

The 784 element Morse input vectors are populated by converting characters from letters and numbers in the OCR input/output into the dots and dashes of the International Morse Code (Morse) [4]. Although Morse standards require special characters to be mapped to specific patterns, that component of the standard was not followed for this work [4]. In Morse, a dot is occupied by one element with value 255 followed by a one element with the value 0, a dash occupies three elements with value 255 followed by one element with the value 0, the space between letters occupies 3 elements with value 0, and the space character is represented by 7 consecutive elements with value 0. The dot and dash representation of each letter and number is specified in the Morse system, and this work maps special characters to existing Morse codes to retain the information they contain. For example, the double quote is treated as a U character, and the comma is treated as the number 4.

The following is an example of a Morse input vector representing the term "socketexception": [255, 0, 255, 0, 255, 0, 0, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 255, 255, 0, 0, 0, 255, 255, 255, 0, 255, 0, 255, 255, 255, 0, 255, 0, 0, 0, 255, 255, 255, 0, 255, 0, 255, 255, 255, 0, 0, 0, 255, 0, 0, 0, 255, 255, 255, 0, 0, 0, 255, 0, 0, 0, 255, 255, 255, 0, 255, 0, 255, 0, 255, 255, 255, 0, 0, 0, 255, 255, 255, 0, 255, 0, 255, 255, 255, 0, 255, 0, 0, 0, 255, 0, 0, 0, 255, 0, 255, 255, 255, 0, ..., 0, 0, 0]

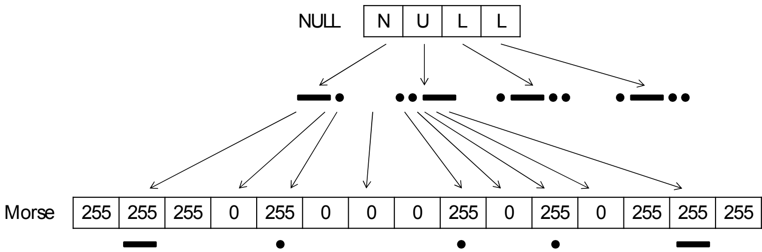


Figure 7.6: Conversion from text into input vectors using Morse encoding.

The classification accuracy of Morse encoding is reported in Table 7.5 and Figure 7.7 under the labels Morse10, Morse20, and Morse100. The Morse encoding scheme was very good at training the deep learning neural network to classify OCR output text. The maximum classification accuracy achieved with Morse encoding was 91%.

Morse with letter frequency encoding (MorseFreq):

Morse encoding of input vectors resulted in 91% classification accuracy. The 9% of TESTING that was incorrectly classified was analyzed to uncover the possible reasons for learning failures. It became clear from looking through the data that one major problem is frame shifts. Frame shifting in the OCR output text (e.g. the 'M' character interpreted as two consecutive lowercase 'l' characters) throws off the neural network by shifting all subsequent letters from their usual position. These shifts can occur multiple times in the same text. Two such shifts confuses the network too much to be resolved correctly. One solution is to encode letter frequency into the final elements of the input vector to preserve the information contained in a term when characters are added and the information in the term is shifted. This encoding region gives the network hints about the term in a frame shift invariant way using letter frequency encoding. As shown in Algorithm 5, 52 elements are used in a 2 bin letter frequency vector. Only the letters 'a' through 'z' (26 characters) are represented in the encoding, and each letter is represented by 2 elements in the vector. The encoding is simple: for each letter, one input represents the presence of a character (e.g. 'a' in the word 'apple' is present and therefore '255'), while the adjacent input represents the detection of more than one of a letter being detected. And so, the two elements encoding the frequency of 'a' in 'apple' will contain '255','0', while the elements for 'p' contain '255','255', and the elements for the letter 'z' will contain '0','0'.

Adding a letter frequency feature to the input vector creates a second avenue for the DNN to learn what each *keyword* looks like. With this second set of features, the DNN can guess at the meaning of a misspelled or shifted word by weighing the strength of the letter and letter frequency features.

The classification accuracy of Morse with 2-bin frequency encoding is reported in Table 7.5 and Figure 7.7 under the labels Morse2Freq10, Morse2Freq20, and Morse2Freq100. The maximum classification accuracy achieved with Morse with 2-bin frequency encoding was 96%.

Binary ASCII with letter frequency encoding (BinAsciiFreq):

Binary ASCII character encoding of BinAscii was combined with letter frequency encoding of MorseFreq to produce another input vector encoding scheme.

The classification accuracy of binary ASCII with 2-bin frequency encoding is reported in Table 7.5 and Figure 7.7 under the labels BinAscii2Freq10, BinAscii2Freq20, and

ALGORITHM 5: Encoding letter frequency of OCR text output into a vector

Input: Array of 26 letters [a-z]: $letter$; text output from OCR module: $ocrText$; generator of 0-filled array: $zeroes(length)$; extractor of occurrences of $letter$ in $text$: $count(letter, text)$; Number of elements used to represent the frequency of each letter: $numBuckets$

Output: Array representation of letter frequency $frequencyArr$

$frequencyArr = zeroes(26 * numBuckets)$

for i *in* $range(0, 26)$ **do**
    $occurrences = count(letter[i], ocrText)$
    for j *in* $range(0, numBuckets)$ **do**
        if $occurrences > j$ **then**
            $frequencyArr[numBuckets * i + j] = 255$
        end
    end
end

BinAscii2Freq100. The maximum classification accuracy achieved with binary ASCII with 2-bin frequency encoding was 96%. Results for binary ASCII with 10-bin frequency encoding are reported as BinAscii10Freq20, where the highest classification accuracy was also 96%.

Binary ASCII with letter frequency encoding and dictionary-based output correction (BinAsciiFreqDict):

To further increase the classification accuracy, the dictionary approach from Section 7.1.2 was combined with the input vector encoding from BinAscii10Freq20 of Section 7.1.5. The dictionary was programmed to replace the output from the neural network when it identified an exact match, and otherwise to defer to the result from the neural network. As depicted in Figure 7.8, if the memoization dictionary does not contain the spelling correction (or exact match) for the input, the DNN input vector is composed with one section encoding ASCII letters in binary, and the other encoding the frequency of each character. An entry recognized by the DNN results in exactly one neuron in the output being activated, as the DNN output employs one-hot encoding. The activated neuron corresponds to a particular globally unique ID (the index), and this ID can be used to look up in a dictionary the keyword corresponding to the activated neuron's index. This

Table 7.4: Classification accuracy for deep learning neural network using ASCII encoding of input vectors when classifying the TESTING dataset.

Training Effort	**Classification Accuracy**
Training on 10% sample of TRAINING	13.72%

approach (BinAscii10Freq20Dict) resulted in 97% as the highest classification accuracy. More detailed results are presented in Figure 7.7 and Table 7.5.

This approach resulted in accurate identification of 2403 out of 2480 images in TESTING. The remaining 77 images from TESTING that failed to be correctly classified were: 13 images where the OCR output was blank, 48 images where key collisions occurred such that the OCR result was text contained in TRAINING but for which TRAINING contained a different answer than the one contained in TESTING, and the remaining 16 images contained OCR output that TRAINING did not contain. One example of these rare OCR output mistakes is the OCR output for an image containing the term `cannotundoexception` where the OCR output was `mammogram`.

7.2 Supervised Keyword Learning Accuracy

The approaches evaluated in this work to improve error message detection accuracy by correcting the OCR output are dictionary (Section 7.1.2), spell-check (Sections 7.1.4 and 7.1.3), and deep learning (Section 7.1.5). The overall results for classification accuracy are discussed in Section 7.2.1.

7.2.1 Supervised Keyword Learning Results

Results for OCR output classification approaches described in this work are presented in Table 7.5, and Figure 7.7, and the full set of results is summarized in Table 7.6 and 7.9.

The deep learning system appears to have generalized more than the fixed dictionary map discussed in Section 7.1.2. OCR tuning, dictionary-based classification, and spell-check were outperformed by a deep learning system. Deep learning has a long initial training phase which becomes a problem only when retraining the whole network to remove (unlearn) a term. 5000 terms in the 1-hot output vector are initially available to be added to the network's lexicon without incurring a heavy training penalty. However, removing terms required the network to be completely retrained. In the dictionary

Table 7.5: Classification accuracy for deep learning neural network using various input vector encoding schemes and classifying the TESTING dataset.

Training Effort	**Classification Accuracy**	**Learning Graph Label in Figure 7.7**
None	61.09%	Baseline (not in Fig 7.7)
Training on 10% sample of TRAINING	88.26%	Morse10
Training on 20% sample of TRAINING	90.84%	Morse20
Training on 100% of TRAINING	88.91%	Morse100
Training on 10% sample of TRAINING	90.28%	BinAscii10
Training on 20% sample of TRAINING	92.01%	BinAscii20
Training on 100% of TRAINING	89.43%	BinAscii100
Training on 10% sample of TRAINING	95.97%	Morse2Freq10
Training on 20% sample of TRAINING	95.85%	Morse2Freq20
Training on 100% of TRAINING	94.03%	Morse2Freq100
Training on 10% sample of TRAINING	96.09%	BinAscii2Freq10
Training on 20% sample of TRAINING	95.97%	BinAscii2Freq20
Training on 100% of TRAINING	94.03%	BinAscii2Freq100
Training on 20% sample of TRAINING	96.29%	BinAscii10Freq20
Training on 20% sample of TRAINING	96.97%	BinAscii10Freq20Dict

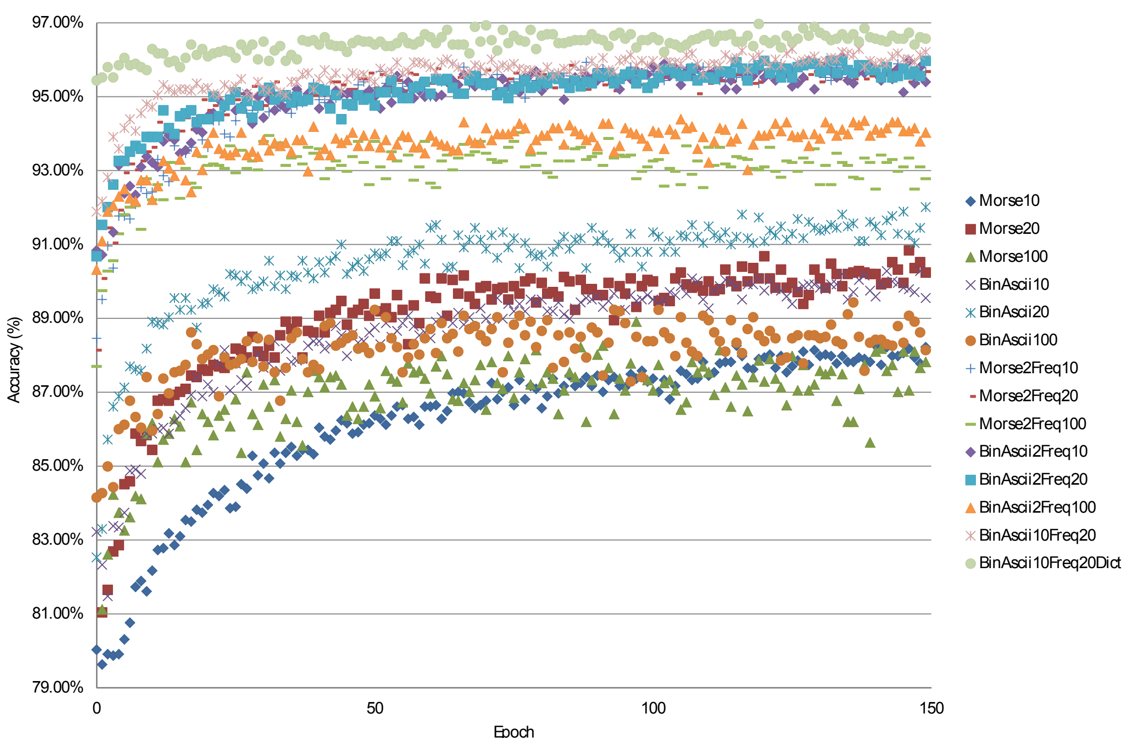


Figure 7.7: Classification accuracy (%) for various deep learning input vector encoding schemes at each training iteration (epoch) during the training phase.

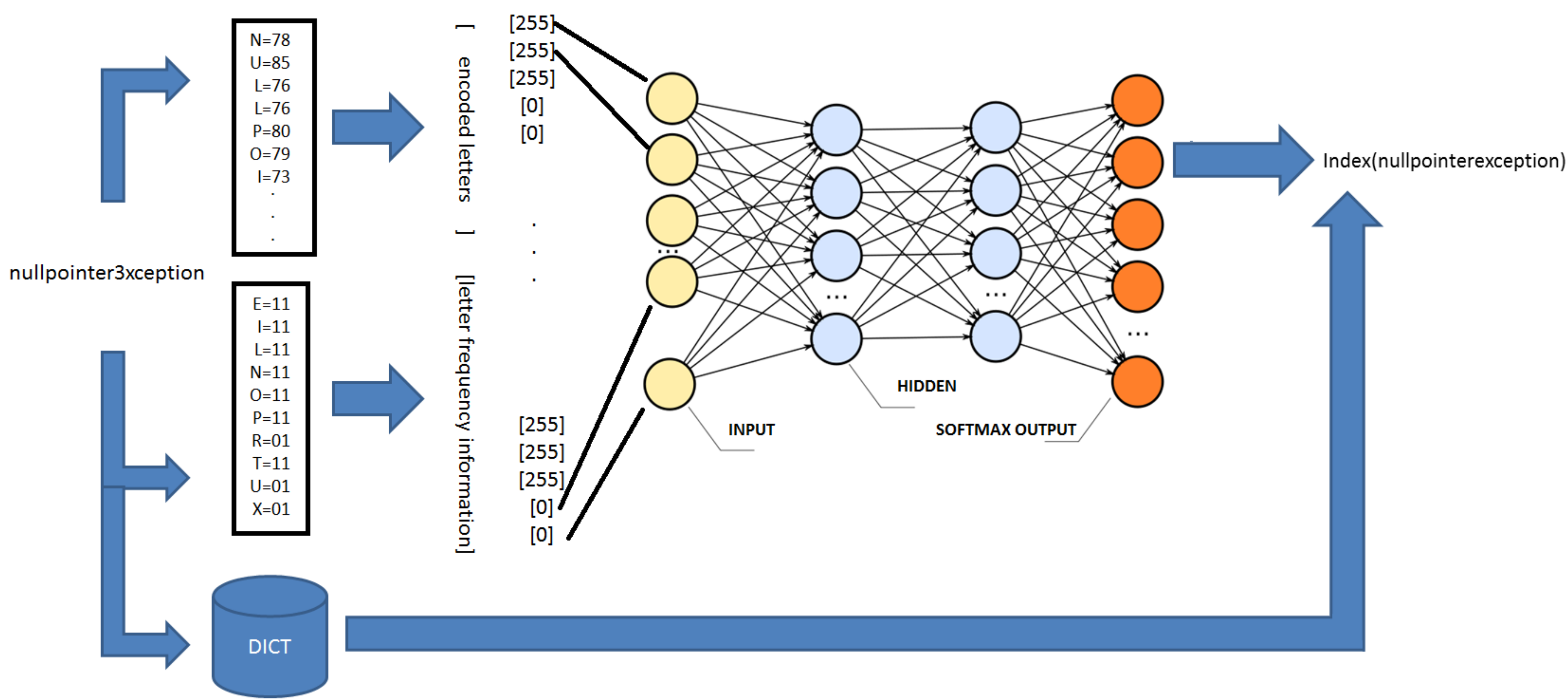


Figure 7.8: Graphical overview of AVRA's DNN (BinAscii10Freq20Dict) implementation encoding text into vectors of binary ASCII and letter frequency information.

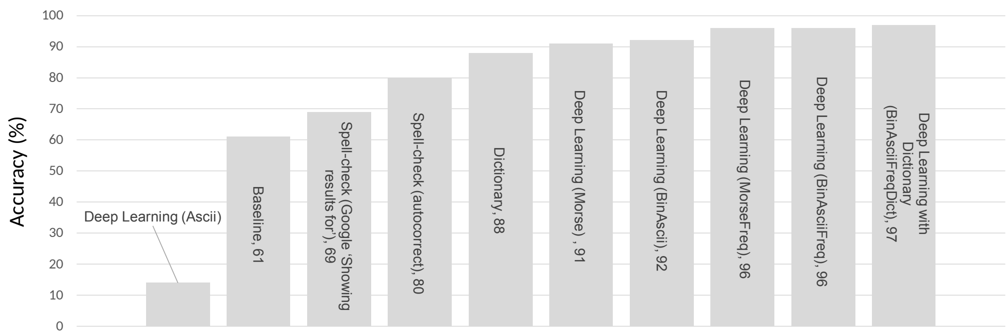


Figure 7.9: Classification accuracy on TESTING dataset for OCR approaches.

approach, adding and removing terms is comparatively very cheap, computationally.

Using a 20% sample of the Morse data trained a better classifier than using 100%. The model is likely overfitting to the training data and then getting stuck on the testing data. This is a common problem with deep learning systems that is mostly solved by dropout, which was used in this work [221]. Specifically, during training there is a 20% chance of dropping out any given input and a 50% chance of dropping hidden layer outputs. Dropout forces the DNN to learn from incomplete data so that when incomplete data arrives at the input, such as misspelled text, it can still recognize the text.

A gradual increase in accuracy from 88% to 97% can be observed in Figure 7.7 as the input vector encoding scheme was tuned with feature engineering to maximize the performance of the neural network.

Comparing ASCII and Morse input vector encoding schemes, the binary encoding of ASCII characters performed equally or slightly better than Morse encoding, even though the length of the data in Morse vectors is typically longer than ASCII vectors. Perhaps these two encoding schemes express equivalent amounts of pattern information for the neural network to learn from.

Dictionary and spell-check-based approaches achieved less than 90% accuracy, while deep learning achieved 96% accuracy, and a combination of deep learning with the dictionary approach achieved 97% accuracy. The accuracy of OCR output is lower when processing onscreen error messages and other domain-specific terminology relative to standard English text. The accuracy of fullscreen image processing using OCR was improved in this work. The systems discussed in this work learned to detect and correct mistakes in OCR output.

Table 7.6: Classification accuracy on TESTING dataset for OCR approaches.

Approach	**Classification Accuracy**	**Comment**
Deep learning (Ascii)	14%	Encoding not crisp enough for training
Baseline	61%	Low performance on error message text
Spell-check (Google 'Showing results for')	69%	Surprisingly close to baseline
Spell-check (autocorrect)	80%	Excels at single letter substitutions and/or added spaces
Dictionary	88%	Cannot generalize beyond training observations
Deep Learning (Morse)	91%	Difficulty with frame shifts within text
Deep Learning (BinAscii)	92%	Difficulty with frame shifts within text
Deep Learning (MorseFreq)	96%	Letter frequency encoding resolves classification after frame shifts in text
Deep Learning (BinAsciiFreq)	96%	Letter encoding schemes BinAscii and Morse produced similar results
Deep Learning with Dictionary (BinAsciiFreqDict)	97%	Slight increase in performance combining a deep learning algorithm with memorization of a dictionary

AVRA is initialized with a set of contexts and related *keywords* using supervised learning, and each context is initialized with a dedicated DNN text classifier. Each DNN was implemented in theano [27] with 5000 outputs, meaning that up to 5000 *keyword* classes can be trained into each context. It was important to configure the DNNs such that each DNN has output vectors with distinct IDs. This is critical to the merging of the results from multiple DNNs in the recommendation filtering stage. In a given DNN, the *keywords* trained into it are perhaps a few hundred strings of text to begin with. For example, for Java programming, 248 terms cast to lowercase such as "indexoutofboundsexception" were trained into the DNN. The remaining outputs of the DNN are available for AVRA to fill with learned information over time. Further discussion on unsupervised learning is presented in Chapter 9.

7.3 Chapter Summary

This Chapter described a method for recognizing onscreen keywords with a DNN, in the presence of spelling mistakes from an OCR conversion of images to text. A DNN supervised learning process was described for recognizing mistakes made by the OCR software when processing specific *keywords*. The ability to recognize onscreen text is essential to the thesis statement that AVRA can provide action recommendations without integration into each individual program executing on the computer. At this point in the thesis, the recognition of onscreen information has been fully elaborated, and the decision of what actions to recommend is presented in the following chapter.

Chapter 8

Ranking Recommendations

The task for AVRA's RS is to rank the possible recommendations according to recognition confidence score (the combined classification confidence scores from the DNN and CNN) and the confidence score for the user preferences (based upon past user behavior selecting recommendations from particular contexts and individual recommendations). When many recommendations are possible, the highest ranked recommendations should be presented to the user. This chapter presents the RS as a hybrid recommendation filtering algorithm performing content-based, context-aware ranking of predictions. If the user behavior changes over time, the recommendation rankings can be automatically adjusted to recommend new actions.

The number of possible recommendations resulting from the context-aware parsing of the text on the screen is usually much higher than the 3 recommendations AVRA can present to the user in the GUI. Consider that each DNN has up to 5000 outputs, and there may be several hundreds or even thousands of DNNs feeding the RS, leading to an action recommendation search space of millions of possibilities. AVRA therefore includes a recommendation filtering algorithm described in Algorithm 6.

The recommendation filtering algorithm used is a hybrid filtering approach incorporating a content-based, context-aware ranking of predictions modified by user history-based and probabilistic approaches. The filtering algorithm is content-based as it ranks recommendations according to attributes of the content, specifically based upon text detected in image data. The filtering algorithm is context-aware as it weighs the user affinity for each context in its objective function when ranking candidate action recommendations. For example, when processing an image that contains a Java error message *keyword*, the algorithm may rank the corresponding action recommendation lower as a

result of the user having a history of ignoring Java-related action recommendations. The filtering algorithm is user-history-based as its recommendations are modified based upon the user's responses to past recommendations. The filtering algorithm is probabilistic as it weighs several probabilities: the probability that a recognized context is in fact present, the probability that a recognized *keyword* is in fact on the screen, the probability that a user will select a recommendation given the past history of that user in terms of contexts and specific recommendations.

AVRA maintains a user profile which includes a history of all actions taken by the user with respect to the GUI. This user profile improves the chances of recommending an action that will be selected by the user by observing past successful RS recommendation predictions. The user provides feedback to the RS filtering algorithm by accepting or rejecting recommendations. The user-RS interaction forms meta-data which guides the adjustment of the weights in the RS algorithm. The RS "evicts" a recommendation when the recommendation is replaced by an update to the GUI to make room for a recommendation that resulted in a higher ranking. Similarly, when the user accepts a recommendation by clicking a button in the user interface (or pressing a hotkey), positive feedback is recorded in the RS database. These eviction and acceptance events are the mechanism whereby AVRA solicits negative and positive feedback from the user.

By following this recommendation approach, this RS follows the advice: "human reasoning is kept within the knowledge discovery loop" [57]. For example, recommendations clicked by the user leads to those recommendations becoming easier to recommended in the future. This happens because the systm becomes biased towards the things it recognizes are preferred by this particular user. Similarly, if one recommendation or a set of recommendations is not interesting to the user, she will opt not to click on these recommendations, leading to increased difficulty for AVRA to recommend the related action or actions. Again this is manifested as adjustments the the weights in the RS algorithm. Therefore, the cold start problem has a low effect on AVRA because the recommendations are strongly driven by onscreen activity (user activity), which leads naturally to an initial set of recommendations (clicks or recommendation rejections).

Consider in detail the filtering algorithm shown in Algorithm 6 consisting of 3 loops. The results from Algorithm 2 are inputs to this algorithm which produces recommendations and then narrows them down to 3 options to present to the user. In the first loop on lines 1 to 9, each DNN that relates to the current context classifies the related input into one of the classes on which it was trained with a finite confidence level between 0 and 1. Upon completion of this classification loop, another loop computes the user preferences

for each possible recommendation (lines 10 to 21). The user preferences are based upon their history of clicking on, and therefore accepting, past recommendations, as well as the count of accepted (clicked) recommendations for each context. This approach increases the likelihood of a recommendation being proposed when the user tends to accept recommendations of a given type or in a given context. The converse is also true, where a recommendation is made less likely if the user tends not to accept a given recommendation or the recommendations in the context of that specific recommendation. In the third loop on lines 22 to 26, the recommendation ranking is adjusted to take into account the computed probabilities from the second loop, and the highest ranked recommendation at each index is assigned to a new list P_{max}. Finally, the 3 highest ranked recommendations in P_{max} are loaded into the GUI buttons. At this point when the buttons are updated, any evictions of recommendations from the buttons are recorded in the database.

P_{max} represents a ranked list of results as the result of the evaluation of several normalized tensors. Not that P_{max} is not normalized so that the contents sum to 1. Instead the indices for the array elements with the highest 3 values in the P_{max} tensor are selected as the recommendations to offer to the user. As depicted in Figure 8.2, AVRA's RS combines five tensors as inputs to make recommendations. In Equation 8.1, each tensor (curves in Figure 8.2) represents the values of each feature within the related tensor. The context tensor contains c features and has the shape $1xc$. The keyword tensor has shape cxk associating k keywords to c contexts. Similarly the $keywordClicks$ and $keywordEvicts$ history tensors are also of dimension cxk. Finally, the $contextClicks$ tensor is of shape $1xc$. The output of AVRA's RS is a flattened version of P_{max} of shape $1x(kc)$. The argmax operator helps to return the indices of the three highest valued recommendations to populate the user interface.

$$\underset{(k*c)\times 1}{P_{max}} = f(\underset{1\times c}{contextClicks}, \underset{c\times k}{keywordClicks}, \underset{c\times k}{keywordEvicts}, \underset{c\times k}{keywordtensor}, \underset{1\times c}{contexttensor}) \tag{8.1}$$

It is helpful to understand RS Algorithm 6 with a step by step contrived simple example using Figure 8.1 as input. Imagine that there are 2 contexts learned into AVRA: console programming ($DNNid = 0$), and programming in the Eclipse IDE ($DNNid = 1$). Within console programming AVRA has learned 2 keywords: *chmod* (*detectedIndex* $= 0$) and *lshw* (*detectedIndex* $= 1$). Within the Eclipse context AVRA has learned 2 keywords: $nullpointerexception$ (*detectedIndex* $= 2$) and $outofmemmoryerror$ (*detectedIndex* $= 3$). In this example, AVRA captures a screen-

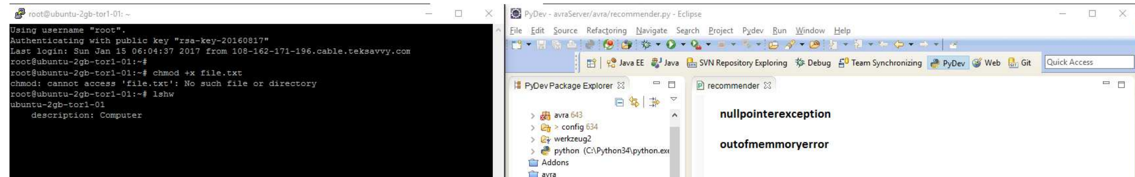


Figure 8.1: A contrived example image for explaining how AVRA's RS makes recommendations based upon screenshots.

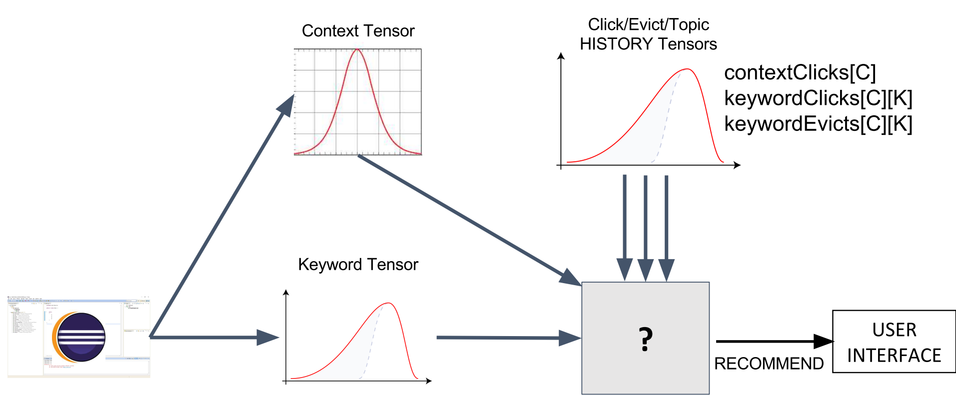


Figure 8.2: A depiction of the information flow through AVRA's RS.

shot containing two side by side windows: one Eclipse window and one console window. Showing two onscreen contexts reveals how AVRA can understand multiple onscreen windows at the same time. With keywords in bold font for clarity, the OCR text for the screenshot of Figure 8.1 is the following:

`root@ubuntu-2gb-torl - PyDev - avraServer/avra/recommender.py le Edit Source J - Eclipse Refactoring Navigate Search I ?, Java EE $11 Java I=1 % I yl A 634 (C:\Python34\python.exc 0 Project Pydev Run Window Help sing username "root". ,uthenticating with public key "rsa-key-20160817" Fast login: Sun Jan 15 06:04:37 2017 from 108-162-171-196.cable.teksavvy.con iclot@ubuntu-2gb-tort-01:4 ,00t@ubuntu-2gb-tort-01:-#` **`chmod`** `+x file.txt hmod: cannot access 'file.txt': No such file or directory ioot@dbuntu-2gb-tort-01:-#` **`lshw`** `untu-2gb-tort-01 description: Computer a)`

```
SVN Repository Exploring 3%;" Debug b*0 Team Synchronizing e PyDev Web Git
B recommender EZ nullpointerexception outofmemmoryerror IQuick Access
RiDevPadmgeE4Momrn > avra 643 > 2dr > config ) werkzeug2 python Addons
avra I=1 X
```

In this example, AVRA's CNN detects that the computer screen is showing a console window (CNN confidence = 50%) and has also detected an Eclipse IDE window (CNN confidence = 99%). These two activated contexts trigger their respective text filters and DNN text classifiers. Each DNN ($DNNid = \{0, 1\}$) receives from a context-specific text filter a list of candidate keywords to be classified. The list of candidate keywords for console programming ($DNNid = 0$) is $\{chmod, hmod, lshw\}$, and the list of candidate keywords for Eclipse ($DNNid = 1$) is $\{nullpointerexeption, outofmemmoryerror\}$. The console programming DNN finds classification results and confidence levels using the function
$argmax(BinAscii10Freq20Dict[DNNid](keyword))$, which returns the detected keyword index and related detection confidence level $[DetectedIndex, ClassificationConfidence]$. For brevity let $b()$ be a new name for $BinAscii10Freq20Dict$. The console programming DNN returns $[0, 1.0]$ for $argmax(b[0]('chmod'))$. The RS then checks that this result is the best so far for this keyword in this context and that the confidence is above 50%. This restriction is satisfied and so $P_{DNN}[0][0] \leftarrow 1.0$. Next the RS observes that the console programming DNN returns $[0, 0.7]$ for $argmax(b[0]('hmod'))$. The RS now checks if this is the strongest detection confidence for 'hmod' thus far in this RS iteration. Checking $0.7 > P_{DNN}[0][0]$ the RS finds that $0.7 \not> 1$, and so the RS moves on to the next keyword. The RS gets $[1, 1.0]$ for $argmax(b[0]('lshw'))$. It then checks that 1.0 is the best result so far for 'lshw' in this context and that the confidence is above 50%. Satisfied, the RS stores $P_{DNN}[0][1] \leftarrow 1.0$. Moving on to the next context ($DNNid = 1$), the RS finds that the DNN for the Eclipse context returns $[0, 1.0]$ for $argmax(b[1]('nullpointerexeption'))$ and for $argmax(b[1]('outofmemmoryerror'))$. Following this procedure has resulted in the RS focusing in on the strongest signal for each onscreen keyword. These observations result in the assignments $P_{DNN}[1][2] \leftarrow 1.0$ and $P_{DNN}[1][3] \leftarrow 1.0$. At this point the RS has completed the first loop of Algorithm 6 (lines 1 to 9). The RS now has a model of which keywords have been detected, and with what confidence they have been detected. The RS now proceeds to summarize the user behaviour so that it can be incorporated into the recommendation rankings. In this example the user has accepted recommendations for the Eclipse context 1 time for a *nullpointerexeption* action recommendation. For the console context the user has accepted recommendations 100 times, where the

accepted recommendations were half for *lshw* and half for *chmod*. These observations result in the following assignments:

$contextClicks[0] \leftarrow 100$
$contextClicks[1] \leftarrow 1$
$keywordClicks[0][0] \leftarrow 50$
$keywordClicks[0][1] \leftarrow 50$
$keywordClicks[1][2] \leftarrow 1$
$keywordClicks[1][3] \leftarrow 0$

The RS has recorded recommendation evictions as follows:

$keywordEvicts[0][0] \leftarrow 500$
$keywordEvicts[0][1] \leftarrow 500$
$keywordEvicts[1][2] \leftarrow 1000$
$keywordEvicts[1][3] \leftarrow 1000$

The RS then computes the overall number of contexts and clicks, followed by a bias to adjust the weights for each context according to the user's click history for that context ($P_{context}$):

$contextCount \leftarrow 2$
$contextClickTotal \leftarrow 101$
$P_{context}[0] \leftarrow 100/101$
$P_{context}[1] \leftarrow 1/101$

For $DNNid = 0$, the $keywordClickTotal$ is computed as 50+50, whereas $keywordEvictTotal$ is $500 + 500$. For $DNNid = 1$, the $keywordClickTotal$ is computed as $0 + 1$, and $keywordEvictTotal$ is $1000 + 1000$. With the above information the RS can compute the following keyword-specific probabilities:

$P_{click}[0][0] \leftarrow 0.495 = ((100/101) * 50)/100$
$P_{click}[0][1] \leftarrow 0.495 = ((100/101) * 50)/100$
$P_{click}[1][2] \leftarrow 0.010 = ((1/101) * 1)/1$
$P_{click}[1][3] \leftarrow 0.010 = ((1/101) * max(1, 0))/1$

Consider at this point the reasoning for the max operations when computing P_{click}. Had the max been omitted for $P_{click}[1][3]$ it would be 0 rather than 0.010. In the calculation of $P_{max}[1][3]$ below, this would rank the keyword $outofmemmoryerror$ in the same way as a keyword that did not appear onscreen at all. The purpose of the max is to slightly improve the odds of first-time recommendations in the RS, especially when there are not higher ranked context-specific keywords onscreen to trigger action recommendations. An odd consequence of using the max function here is that the probabilities P_{click} add to slightly more than 1.0. This is not a real problem as these numbers are used for ranking rather than true event prediction. The RS proceeds to calculate the eviction probabilities used as negative feedback in the ranking of action recommendations:

$P_{evict}[0][0] \leftarrow 0.495 = ((100/101) * 500)/1000$
$P_{evict}[0][1] \leftarrow 0.495 = ((100/101) * 500)/1000$
$P_{evict}[1][2] \leftarrow 0.005 = ((1/101) * 1000)/2000$
$P_{evict}[1][3] \leftarrow 0.005 = ((1/101) * 1000)/2000$

Selecting K_1 as 0.999 to highly reward clicks and K_2 as 0.001 to slightly punish evictions, the RS can now calculate which recommendations to display to the user by computing and ranking P_{max} as follows:

$$P_{max}[i][g] \leftarrow P_{DNN}[i][g] * (1 + K_1 * P_{click}[i][g]) * (1 - K_2 * P_{evict}[i][g])$$

$P_{max}[0][0] \leftarrow 1.494 = 1.0 * (1 + 0.999 * 0.495) * (1 - 0.001 * 0.495)$
$P_{max}[0][1] \leftarrow 1.494 = 1.0 * (1 + 0.999 * 0.495) * (1 - 0.001 * 0.495)$
$P_{max}[1][2] \leftarrow 1.010 = 1.0 * (1 + 0.999 * 0.010) * (1 - 0.001 * 0.005)$
$P_{max}[1][3] \leftarrow 1.010 = 1.0 * (1 + 0.999 * 0.010) * (1 - 0.001 * 0.005)$

Finally, $argmax(P_{max}, 3)$ selects to display to the buttons action recommendations for either $\{chmod, lshw, nullpointerexeption\}$ or $\{chmod, lshw, outofmemmoryerror\}$.

Recommendation history informs the usefulness of new candidate recommendations. However, ignored recommendations are much less meaningful than a clicked recommendation. Firstly, the user may not be at the computer screen when the recommendation was evicted. Secondly, a click indicates a stronger positive conscious remark on a recommendation than a recommendation eviction indicates negatively on that recommen-

dation. Thirdly, a recommendation may be evicted as a result of a change in the content of the screen, triggering a context change, rather than the user's active decision to ignore the recommendations available in the GUI. Finally, there are many more evictions than clicked recommendations, and so clicks should be weighted more heavily. For these reasons the weighting K_1 for clicks is much higher than the weighting for evictions K_2. Overspecialization of the filtering algorithm is explored in Section 3.7.

The AVRA database contains the following information: a list of recommendations, recommendation history including user activity profile, user registration information, browser history, and the history of processed image text. The provenance of each recommendation is accessible programmatically, as a SQL history of detected *keywords*, recommendations based on recognized *keywords*, and accepted recommendations.

Relating the RS back to the thesis statement, the RS performs a shallow search with the CNN context detection, and with OCR and DNNs performs deeper context-driven analysis.

Regarding the action and state representations discussed in [131], AVRA's RS model focuses on state representation. The CNN detects context from the current screen state, and then relevant DNNs extract information from that state. Another major difference between AVRA and next-command prediction approached such as [115] is that AVRA seeks out information from many different programs visible onscreen, while command prediction only detects the command history within one program.

While utilizing an ontology or taxonomy to model recommendation scores is a powerful approach which encapsulates entity relationships, in this work a context-based scoring was used. In principle, an ontology-based approach could be incorporated into AVRA's RS in addition to the context-based approach. In this work context is approached from the perspective of [10], where recommendations can be annotated with additional situation-based information called the context. Rather than interpreting the computer screen in every possible context, this work is about narrowing down the search for onscreen meaning to a select few contexts. A drawback of the RS described here is that it ignores the number of times a *keyword* appears onscreen.

This chapter presented AVRA's RS. The RS decides which recommendations to present to the user taking into account the user behavior, and classification confidence scores of the DNN and CNN. The RS goes to the heart of the thesis statement that AVRA can provide action recommendations related to onscreen messages. The RS personalizes to the user recording the history of user interactions with the GUI buttons. At this point in the thesis the components of AVRA have been described and the capabilities explained.

The next challenge is to move beyond supervised learning, at to see how AVRA can act autonomously to learn new contexts, keywords, and action recommendations.

ALGORITHM 6: Recommendation Ranking Algorithm: Rank action recommendations by recognition confidence and user history

Input: g index of text G; $contextClicks[i]$; $keywordClicks[i][g]$; $keywordEvicts[i][g]$; Bias variables K_1, K_2: $K_1 >> K_2, K_1 + K_2 = 1$; DNN $keyword$ recognition results P_{DNN},; Probability distributions $P_{regex}, P_{max}, P_{take} = zeroes()$

Output: 3 recommended actions in Buttons[0..2]

```
1  for each i ∈ DNNid.keys() do
2      for each ocrText ∈ DNNinput[i] do
3          [DetectedIndex, ClassificationConfidence] ←
           argmax(BinAscii10Freq20Dict[i](ocrText))
4          if ClassificationConfidence > 0.5 AND ClassificationConfidence >
           P_DNN[i][DetectedIndex] then
5              P_DNN[i][DetectedIndex] ← ClassificationConfidence
6          end
7      end
8      contextClicks[i] = Σ clicks(DNNclasses[i].values())
9  end
10 contextCount ← Length(DNNid.keys())
11 contextClickTotal ← Σ_{i=1}^{Length(contextCount)} contextClicks[i]
12 for each i ∈ DNNid.keys() do
13     P_context[i] ← contextClicks[i]/contextClickTotal
14     classesCount ← Length(DNNclasses[i].keys())
15     keywordClickTotal ← Σ_{g=1}^{classesCount} keywordClicks[i][g]
16     keywordEvictTotal ← Σ_{g=1}^{classesCount} keywordEvicts[i][g]
17     for each g ∈ DNNclasses[i].keys() do
18         P_click[i][g] ←
           (P_context[i] * max(1, keywordClicks[i][g]))/max(1, keywordClickTotal)
19         P_evict[i][g] ← (P_context[i] * keywordEvicts[i][g])/keywordEvictTotal
20     end
21 end
22 for each i ∈ DNNid.keys() do
23     for each g ∈ DNNclasses[i].keys() do
24         P_max[i][g] ← P_DNN[i][g] * (1 + K_1 * P_click[i][g]) * (1 − K_2 * P_evict[i][g])
25     end
26 end
27 Buttons[0..2] ← argmax(P_max, 3)
```

Chapter 9

Unsupervised Learning

AVRA learns through the inference of operational knowledge from observations of the computer over time. AVRA monitors the computer screen with regular screen capture images, and analyzes the content of each image to understand what type of visual information is present (the context) and which keywords are on the screen. Each context is associated with a list of keywords, and at any given time there may be multiple onscreen contexts and multiple onscreen keywords. Each keyword in each context is associated with one action. For example, if the Eclipse IDE is present (context=eclipse) and a compiler error is detected in the onscreen text (keyword=NullPointerException), then AVRA can recommend to the user to open a web page containing advice on escaping this error with a try/catch block. In order to offer the most relevant action recommendations, AVRA produces recognition scores for context detection and keyword detection, and combines these scores with a history of user actions to produce an overall score for every possible recommendation.

Several definitions are required in order to discuss unsupervised learning in a compact format. A candidate keyword k is one text snippet within the onscreen text O. The notation $k \in O$ means that the keyword k is in the set O, in this case because it is a substring of O. An action can be referred to as $g \in G$, where g is a particular action and G is the set of all actions known to AVRA. Similarly, a particular context c is one element in a set of contexts learned into AVRA, denoted as $c \in C$. The set of all keywords learned by AVRA is K, and after discovering $k \in O$, AVRA can integrate k to become $k \in K$.

The challenge discussed in this Chapter of the thesis is to learn new contexts, keywords, and actions without human intervention. How can AVRA autonomously learn new visual contexts into C beyond "eclipse", and new context-specific keywords such

as `NullPointerException` into K? Even if learning new contexts and keywords were accomplished, how can AVRA learn which actions G are associated with contexts and keywords? That is the topic of this chapter. More formally, this unsupervised learning challenge is to:

(TASK 1) Identify keyword k within onscreen text O leading to action g in context c.

(TASK 2) Recognize context c if it appears again.

(TASK 3) Recognize keyword k if it appears again in onscreen text O when context c is recognized.

(TASK 4) Enable AVRA to recommend action g when context c and keyword k are recognized onscreen at the same time.

This Chapter describes a novel approach to unsupervised learning for a computer assistant. Once AVRA can watch users and draw causal relationships from user actions, it can learn new information unforeseen by its developers. The first step on the path to a working solution was to relax the constraints on the problem and solve easier problems of unsupervised action learning without context learning (Sections 9.1) and supervised context learning (Section 9.2). The solutions to these simplified problems are then combined and expanded upon in Section 9.3 to answer the larger question of how to learn new actions, contexts and keywords. Next, performance experiments are detailed in Section 9.4.

9.1 Unsupervised Action Learning Without Context

Consider a relaxed version of (TASK 1), where there is only one context. The objective is therefore to identify what onscreen text k in the onscreen text O leads to action g. In this relaxed case, (TASK 2) is not necessary, (TASK 3) simplifies to recognizing when text k appears within the onscreen text O, and (TASK 4) simplifies to recommending action g when text k appears onscreen.

Let the input to the unsupervised learning algorithm be the stream of tokenized timestamped OCR text $O(t_1)$ produced when processing each computer screen image. Each image is associated with a timestamp t_1. Next, let the actions $g \in G$ be the detected user interest text (e.g. browser search, clipboard history, keystrokes). Each element g in G is an action performed by the user which could conceivably be replayed by AVRA on

behalf of the user. Each observed user action is associated with a timestamp t_2, yielding a stream of timestamped actions $G(t_2)$ and corresponding keyword $K(t_2)$. A dictionary F can store the relationship between keywords and actions as $F < k, g >$ allowing AVRA to identify the desired action g when it detects keyword k.

The learning algorithm can iterate through the OCR text $O(t_1)$ and actions $K(t_2)$, and store into F wherever $K(t_2)$ came soon after the OCR text $O(t_1)$ appeared onscreen, and the user interest text $K(t_2)$ was a substring of the onscreen text $O(t_1)$. These constraints are expressed as $[t_2 > t_1]$ *and* $[K(t_2)$ *in* $O(t_1)]$ *and* $[t_2 - t_1 < windowSize]$. Following this approach, the algorithm can learn from scenarios where the user copies onscreen text (a substring of $O(t_1)$) and pastes into a search engine producing the action text $K(t_2)$ for action $G(t_2)$. After learning this pattern in F, AVRA can recommend the relevant action when a keyword appears onscreen, without the user copying and pasting and searching. Algorithm 7 implements these concepts. It provides a method for determining what onscreen text O leads to action G in this single context problem. The approach is to search for an onscreen *keyword* that the user searched for in the past (verbatim) after seeing it on the screen.

ALGORITHM 7: Learning Algorithm: What onscreen text O leads to action G

Input: OCR text of computer screen at time $t1$: $O(t1)$; Detected user interest text (e.g. browser search, clipboard history, keystrokes) at time $t2$: $G(t2)$

Output: Database of problem / solution pairs $F < O, G >$

for ***each*** *O(t1) in history* **do**
  for ***each*** *K(t1) in O(t1)* **do**
    for ***each*** *G(t2) in history* **do**
      if $t2 > t1$ *and* $G(t2)$ *in* $K(t1)$ *and* $t2 - t1 < windowSize$ **then**
        $F.store(K(t1), G(t2))$
      end
    end
  end
end

Screenshot	dog	mule	Search Engine dog Result 1 Result 2 Result 3
Screenshot Name	0.jpg	1.jpg	2.jpg
F<K,G>	-	-	<dog,search(dog)>
O(t)	dog	mule	dog
K(t)	-	-	Search(dog)
t	0	1	2

Figure 9.1: Unsupervised learning with only one context.

Figure 9.1 illustrates how F is assembled as AVRA observes the user in time. At time $t = 0$ in the first column, the user and AVRA see an image of a dog and the onscreen word dog. Next, at time $t = 1$, some other information is seen on the screen, and finally at time $t = 2$, the user searches for the word "dog" and AVRA stores into F the fact that the recently observed onscreen word "dog" led to the user performing a search action for that word.

A weaknesses of the unsupervised learning approach in Algorithm 7 is the following. Due to the strict nature of the learning algorithm, when the unsupervised learning system observes the user multiplying numbers in a calculator, the exact sequence of operations (e.g. $25000 * 4$) is stored as a pattern to be recommended later on, rather than being learned symbolically as the multiplication of two numbers.

9.2 Supervised Context Learning

An approach to unsupervised action learning without context learning was discussed in the previous Section. Before adding unsupervised context learning back into the problem definition, consider a different relaxed problem related to (TASK 2): supervised context learning. To learn contexts, a CNN is trained on a set of labeled images. Transfer learning

with inception v3 [230] and cross-validation training improve the learning process to an acceptable level of precision and recall. The content of a computer screen is very dynamic and complex, and so the focus in training the CNN to recognize contexts was on high recall at the expense of precision. This approach lowered the chance that contexts would be missed (false negative) when the detection of the context is uncertain.

The challenge in training the CNN with supervised learning is acquiring many images that look like a particular context, in order to carry out the supervised context learning. At least 30 images representing the context should be used to train the CNN in order to avoid total failure of the training. However, 300 to 800 training images is a "good" image dataset size for each context. Only when a sufficient number of images have been collected can the images be used to train the new context into the CNN, or reinforce an existing context with new information. Several image collection approaches are possible:

(METHOD 1) Capture an image representative of the context each time AVRA learns a new keyword into F.

(METHOD 2) Collect images from image search engines based upon context-specific keywords.

(METHOD 3) Given one or more images representing a context, collect additional images using reverse image search.

(METHOD 4) Leverage collaborative filtering to collect context-specific images identified by other AVRA users.

For (METHOD 1), collecting the context training images locally with AVRA was accomplished by simply retaining the screen captures stored by AVRA during routine operation. Working backward from the time when K stored a new keyword, the image captured at time t_1 when k appeared onscreen, the image captured at t_1 should be a picture containing the context of interest C_i. The downside of this approach is that it requires many observations to collect sufficient data to train the CNN. This approach was further improved by sampling the images just before and after t_1 and including them in the training data if they were similar to the image taken at t_1. Similarity was established with a perceptual hash comparing the image taken when the keyword was onscreen, and the images taken at nearby timestamps. For example, if the image taken at time t_1 is a picture of the Eclipse IDE showing a stacktrace containing `NullPointerException`, then the next image taken is likely also a picture of the Eclipse IDE. If these images are in fact similar, then the difference in perceptual hash values between the image taken

at time t_1 and the image taken at time image t_{1+1} would be small. Similarly the image taken at time t_{1-1} may be a useful training example if the perceptual hash difference from the image taken at t_1 is small. Image similarity can be controlled by tuning an image similarity hyperparameter.

(METHOD 2), scraping representative images from the Internet to form training datasets, was implemented in nodejs. The program cycled through a hand-crafted list of keywords based upon K relating to the desired context (e.g. "eclipse IDE java programming") and submitted these keywords to image search engine APIs. The search engine submissions returned lists of URLs for images and additional information about these images such as image type and size. The image search was narrowed to include only large images with specific image formats. The next step involved manual data validation where non-representative images were deleted by a human operator. A further step of duplicate image deletion was accomplished with an automated tool.

Whereas (METHOD 2) relied on keyword-based search engines, (METHOD 3) involved querying perceptual hash search engines (e.g. TinEye [235] and Yandex [261]). To find novel images related to the already collected image(s), the perceptual hash of the known image can be used to identify similar images. The downside of this approach was that there may be no such images available, or the identified images may be copies of the submitted image with tiny modifications (e.g. added or modified text).

For (METHOD 4), the collaborative filtering of user actions in a distributed framework with many clients, requiring several instances of an action to be observed before learning a pattern is a reasonable expectation. It is the basis of the collaborative filtering concept that data for one user can be applied to another user.

To learn new contexts in an unsupervised fashion, one or more of these approaches must be automated, removing all human intervention. For example, with (METHOD 2) image search keywords are produced manually, and images are validated manually. With (METHOD 1) the user must perform the same action many times, and the image similarity metric must be flexible enough to allow differences between images but strict enough to reject irrelevant images from polluting the training data.

9.3 Unsupervised Context Learning

Having outlined solutions to unsupervised action learning and supervised context learning, enough of the solution is revealed that one can begin to consider the full scope of the unsupervised learning problem. Consider AVRA's design shown in Figure 9.2. How

can AVRA autonomously identify keywords K within onscreen text O leading to action G in context C?

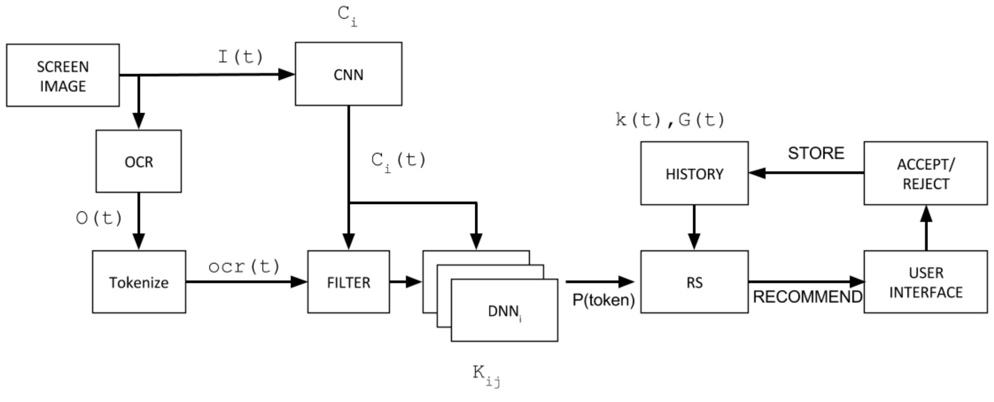


Figure 9.2: AVRA design overview.

Consider that AVRA has just detected that a keyword k captured at timestamp t_1 and recognized in image $I(t_1)$ was followed by user action $g(t_2)$. AVRA must decide if this keyword belongs to an existing context or a new one. New challenges emerge when attacking this broader problem definition. First, the fit between a new image $I(t_1)$ and existing trained CNN context C is required to understand how well the new image fits into the set of features that define each context. Second, the relative fit between keyword k and every existing context in C must be quantified in order to decide into which context new information should be stored. Third, AVRA requires an autonomous method for extrapolating novel images from the set of acquired images, as described previously in Section 9.2.

To obtain the image 'fit', the CNN can tell the unsupervised learning algorithm how much a new image 'looks like' the contexts it was already trained to recognize simply by processing the image in the same way as AVRA interprets screenshots. This capability is exposed by simply processing the image $I(t_1)$ through the CNN and observing the classification confidence score for each context. The output of the CNN indicates how much the image looks like each context.

A trained word embedding model should contain a representation that encodes the semantic understanding of words. The vectors for words can be manipulated to compare ideas, as previously described in the famous $king - man + woman = queen$ example [148]. To find the fit of a new keyword k with an existing context C_i, one or more trained word embedding models are interrogated to find out if k and many keywords of interest K are represented in the model. If so, the cosine similarity between the keywords K already in the context C_i and the new keyword k is computed. More specifically, AVRA's unsupervised learning algorithm relies on the Google News word embedding model to obtain the conceptual distance between words [147]. If a word is not found in the word embedding model, then the distance is set to 0. The average similarity between k and the keywords in context C_i represents the relative fit of keyword k into C_i. If each context contains n keywords, and there are m contexts trained into AVRA, then keyword k must be compared to $m * n$ elements.

At this point the 'fit' between a new keyword and each context is computed as the similarity between a keyword and each keyword in each context. Each of the $m * n$ calculations peers into the Google News word2vec model, requiring several hours to execute. To accelerate the comparison to several seconds, an average vector for all the keywords in each context is computed, and then compared with the candidate keyword. Several approaches are well known for computing a vector to represent a set of words in a word embedding model, including the average vector approach implemented in AVRA [117], and k-means clustering [172, page 5]. One major advantage of this new approach with average vectors is that computing the average vectors is accomplished outside the word2vec model, and so it executes very quickly. Further complicating matters, some keywords AVRA learned during supervised learning are not available (not trained into) in the word2vec model, and so the vector for those keywords in the model does not exist. Therefore the fit between the candidate keyword and those missing vectors were not taken into account in creating the average vector. To fix this, the fit between the average vector and the vector for the new keyword k is multiplied by a ratio of A (the number of vectors used to make the average vector) and B (the total number of keywords in the context of interest). And so if all the context keywords are in the model, the ratio is 1.0, if none are, then the ratio is 0.0, and if half are in the model, then the ratio is 0.5. The calculation of the mean similarity of therefore the similarity between k and the average vector, multiplied by the ratio.

Another acceleration technique used was memoization. Any intermediate results (e.g. distance('paris','france')) is not re-calculated when the result is needed later on.

As discussed previously, approaches to getting more images representative of a new CNN context requires fully automating one or more of (METHOD 1) (making multiple observations of $k + C_i \rightarrow g$ before learning a new context, and obtaining similar images nearby in time using perceptual hashing), (METHOD 2) (keyword-based image search engines), (METHOD 3) (perceptual-hash reverse image search), and (METHOD 4) (collaborative filtering). (METHOD 1) was already fully automated, and provides a small number of useful images as a starting point for the CNN training dataset. To automate (METHOD 2), images obtained from (METHOD 1) were fed to an image labeling API (Google Vision API [77]) in order to come up with a set of keywords, and these keywords were submitted to image search engines to obtain new image examples. The resulting images were a poor fit for the contexts tested (e.g. console, Eclipse IDE) as a result of the small number of labels that the image annotation API was able to extract from the available images. Obtaining keywords from images was possible, and scraping images based on these keywords was also possible, but the quality of the generated keywords was too low to be useful for an autonomous use case. For example, the labels added to an image of a console window were: Text, Font, Brand, Screenshot, Design, Presentation, Line, and Document. Searching for images using these labels does not return additional images of console windows. (METHOD 2) automation was therefore not successful. Surprisingly, full automation of (METHOD 3) was similarly disappointing. Reverse image search did sometimes return novel examples of the submitted context (e.g. image of a console window returned additional examples of console windows). However, reverse image search tended to return either no results (e.g. a fullscreen image of the Eclipse IDE) identical copies of the submitted image (e.g. an image of a celebrity) or results focusing on the "wrong" features of the submitted image (e.g. for a screenshot of a desktop, the perceptual hash caused the results to be the image shown as the desktop background with the desktop icons removed, focusing on the irrelevant background image at the expense of the real goad of finding images of desktop backgrounds). Full automation of (METHOD 3) was therefore not successful. Collaborative filtering (METHOD 4) was not implemented as part of this work.

Having described how the text fit and image fit are computed, and how a dataset to train the CNN can be obtained for new contexts, the unsupervised learning process can now be discussed in additional detail. For each new keyword k under consideration, if AVRA has seen enough examples of the keyword (and related images for context training) triggering an action, then AVRA will begin assessing which context the keyword belongs in. This may be a new context or an existing context. This keyword may already exist in

one context and now also belongs in another. Let $I_{examples}$ be a list of the images related to one keyword k. AVRA begins by assessing the keyword in relation to the keywords already learned for each context C_i. The average fit between the new keyword k and the existing keywords in C_i is computed using the ratio with average vector comparison approach described above. Next, the average fit between the images $I_{examples}$ and the context is computed by processing them through the CNN and averaging the classification confidence for class C_i. If $I_{examples}$ does indeed contain similar features to the already trained CNN class, then a high average fit is expected. To associate the keyword to an existing class, the average image fit must exceed hyperparameter h_1 and the keyword fit must exceed hyperparameter h_2, and the average image fit multiplied by the keyword fit must exceed any previously encountered "best context fit". In other words, the class with the strongest keyword and image similarity is assigned the keyword unless either the keyword or images are too dissimilar from any existing context. In that case a new context is learned.

If the keyword is learned into an existing context C_i, then the CNN can be retrained with images $I_{examples}$, and the keyword identification system for C_i is also updated to recognize the new keyword. If, however, the keyword is learned into a new context, then AVRA finds additional distinct images according to (METHOD 1), in an attempt to increase the number of images available for training. This larger image set is used to retrain the CNN to identify the new context. The keyword identification system is also updated to recognize the new keyword.

AVRA's unsupervised learning algorithm described in this Section is presented in Algorithm 8. Similar to Algorithm 7, it can learn causal relationships between onscreen keywords and user actions. However, the added advantage in Algorithm 8 is that it can also learn features from what the screen looks like when the keyword is present (image contexts). On the first line of Algorithm 8, the observations made by AVRA are processed to identify a set of keywords (*new_keywords*) that appeared onscreen prior to the user performing a related search. This part of the algorithm work as described in Algorithm 7. For each keyword k in *new_keywords*, a list of images $I_{examples}[k]$ is also collected. Next, for each keyword and corresponding action $([k, g])$, if there were enough causation examples observed, the algorithm checks each context to see which has the highest context fit (lines 6 to 13). If a best context is found, then the new keyword k is associated to the action g in the DNN for that context. However, if no suitable context was identified, a new context is trained. In the event that the DNN is bypassed as discussed in Chapter 3.7, the keyword k and action g are added to AVRA's database

as usual. However, in that case the DNN is not trained on the new keyword.

This method for unsupervised learning can be viewed as partitioning the space of all images and words into sub-regions by context and keyword. Keyword clustering is one part of the partitioning, and image clustering is the other. The unsupervised learning approach in AVRA incrementally clusters sets of keywords and stereotypes of images. Figure 9.4 shows the block diagram for AVRA's unsupervised learning approach. A sufficient number of new keyword identifications (TASK 1) causes a decision engine to assess a keyword $k(t)$ recognized in image $I(t)$. Next, to accomplish context recognition (TASK 2), the fit between the existing CNN contexts and the new image is computed (Context Similarity in Figure 9.4). Further to (TASK 2), the images provided to characterize the potential new context are extended by testing the images taken just before and after time t with a perceptual hash, and keeping images with a difference less than $h3$ from $I(t)$. The resulting set of images is called $I_{examples}$. The context similarity task returns the list of contexts sufficiently similar to the potentially new context (with a similarity threshold of $h1$). DNN training to recognize a new keyword within a context (TASK 3) is only initiated once a keyword has been assigned a context. To assign a keyword to a context (new or existing), the keyword fit is first computed (Keyword Clustering in Figure 9.4), and a list of contexts with sufficiently similar keywords is returned. The text similarity threshold is hyperparameter $h2$. If no context has a sufficiently high keyword fit and context fit, n new context is present, and the CNN is retrained to recognize the new context. However, if there are contexts with sufficiently high keyword fit and context fit, the context with the highest combination of context and keyword fit (computed by multiplying them together) is assigned the new *keyword* k. When the keyword is assigned a context, in addition to the DNN training being initiated, action g is associated to the new keyword k in AVRA's database (TASK 4). To extract user actions from the computer, a browser history program was developed to read out keyword search terms and links from the browser along with visit timestamps and page titles. This information was fed into AVRA's database to form the user action history (G).

The key overlap between AVRA's shallow image processing integration and prior work on fullscreen image processing with a CNN to take decisions ([153]) is the use of a CNN to process the image of the screen, and then using fully connected layers of a Deep Neural Network (DNN) to make a decision. In the case of AVRA, the DNN output is a recommendation to be ranked based upon supervised or unsupervised learning, whereas in [153] the outputs represent joystick positions learned through reinforcement learning.

ALGORITHM 8: AVRA's unsupervised learning algorithm.

Input: Minimum observations of keyword and subsequent action $h0 : 1$; Minimum image fit $h1 : 0.1$; Minimum keyword fit $h2 : 0.1$; Minimum CNN recognition confidence to add new image to training data $h3 : 0.1$; Maximum perceptual hash difference to add new image to training data $h4 : 35$; Detected user search text, URL, and timestamp recorded at time $t2 : G(t2)$; Snippets of onscreen text observed at time $t : ocr(t)$; Maximum time from observation of keyword to action by user $windowSize : 10\ s$; List of CNN contexts $contexts$

Output: Keyword added to new context or existing context, or nothing learned.

```
1  [I_examples, new_keywords] = detectNewKeywords(G, ocr, windowSize)
2  for each [k, g] in new_keywords do
3  |   if length(I_examples[k]) ≥ h0 then
4  |   |   best_context_fit = 0
5  |   |   best_context = None
6  |   |   for each context in contexts do
7  |   |   |   keyword_fit = modelTextFit(k, context.keywords())
8  |   |   |   img_fit = average(CNN_classify(I_examples[k], context))
9  |   |   |   if img_fit > h1 and keyword_fit > h2 and img_fit * keyword_fit >
   |   |   |   best_context_fit then
10 |   |   |   |   best_context = context
11 |   |   |   |   best_context_fit = img_fit * keyword_fit
12 |   |   |   end
13 |   |   end
14 |   |   if best_context then
15 |   |   |   train_DNN(best_context, k, g)
16 |   |   |   train_CNN(best_context, I_examples[k])
17 |   |   end
18 |   |   else
19 |   |   |   c = newContextID()
20 |   |   |   I_examples[k] = moreImages(I_examples[k], k, CNN_contexts, h3, h4)
21 |   |   |   train_new_DNN(c, k, g)
22 |   |   |   train_CNN(c, I_examples[k])
23 |   |   end
24 |   end
25 end
```

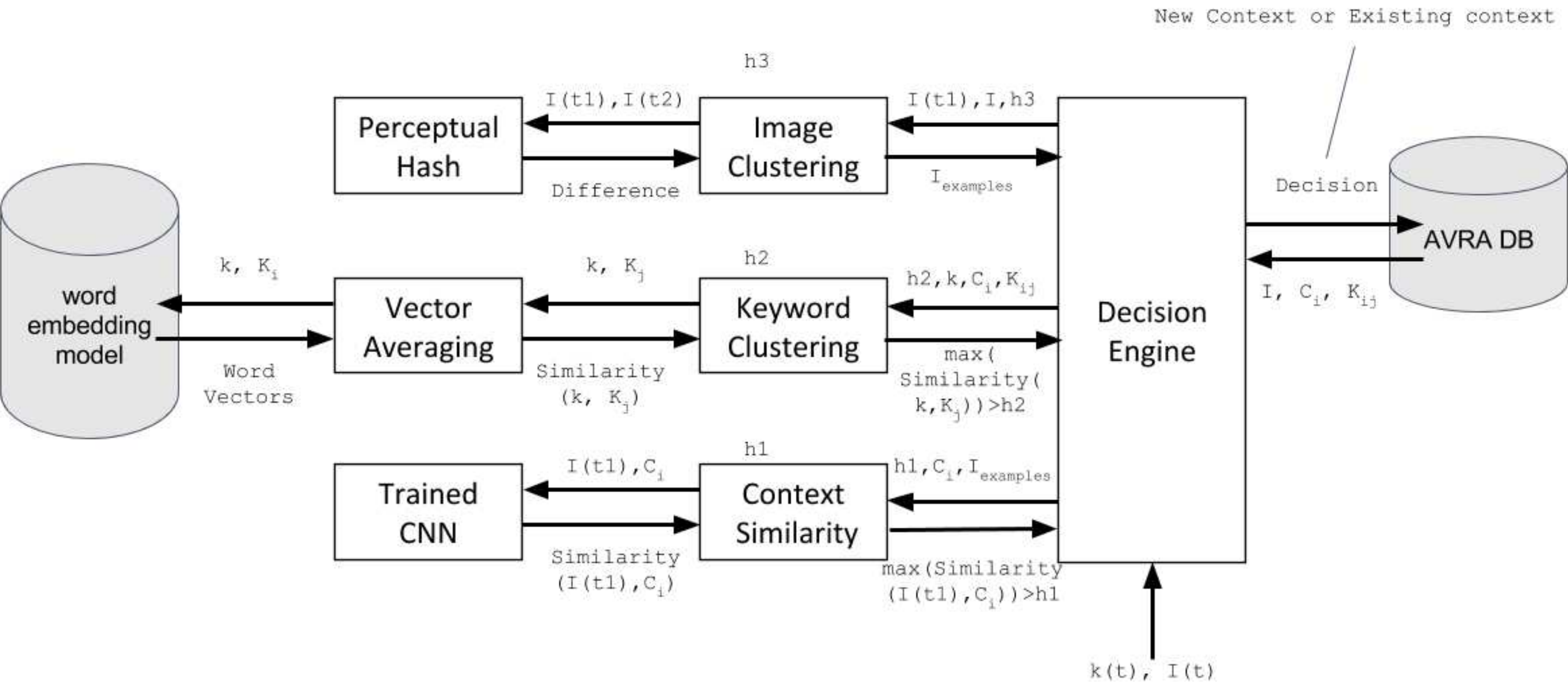


Figure 9.4: Block diagram for AVRA's unsupervised learning algorithm.

Screenshot	dog	Search Engine dog Result 1 Result 2 Result 3	minimumSimilarity←0.65 text_fit(dog, animals)←0.8 text_fit(dog, colors)←0.1 image_fit(0.jpg, animals)←0.9 image_fit(0.jpg, colors)←0.7 fit(dog,animals)←0.72=0.8*0.9 fit(dog,colors)←0.07=0.1*0.7 Decision: Add dog to animals CNN.train(animals, 0.jpg) DNN[animals].train(dog)
Screenshot Name	0.jpg	1.jpg	
O(t)	dog	dog	
K(t)	-	Search(dog)	
t	0	1	2
Known contexts (keywords)	Animals (cat, mule), Colors (red, green)	Animals (cat, mule), Colors (red, green)	Animals (cat, mule, dog), Colors (red, green)

Figure 9.3: Unsupervised learning with multiple contexts learning a new concept and adding it to an existing context.

Screenshot	dog	Search Engine dog Result 1 Result 2 Result 3	minimumSimilarity←0.65 text_fit(dog, shapes)←0.0 text_fit(dog, colors)←0.1 image_fit(0.jpg, shapes)←0.6 image_fit(0.jpg, colors)←0.7 fit(dog,shapes)←0.0=0.0*0.6 fit(dog,colors)←0.07=0.1*0.7
Screenshot Name	0.jpg	1.jpg	Decision: Add dog to new context CNN.train(newContext, 0.jpg) DNN[newContext].train(dog)
O(t)	dog	dog	
K(t)	-	Search(dog)	
t	0	1	2
Known contexts (keywords)	Shapes(round, line), Colors (red, green)	Shapes(round, line), Colors (red, green)	newContext (dog), Shapes(round, line), Colors (red, green)

Figure 9.5: Unsupervised learning with multiple contexts learning a new concept and adding it to a new context.

Having described above the unsupervised learning algorithm in AVRA, consider two examples of how it works. Figure 9.3 shows how an existing context can be extended with a new keyword, while Figure 9.5 shows an example of AVRA learning a new context.

Starting with the second column of Figure 9.3, AVRA sees an image of a dog and the text `dog`. The image at timestamp for $t = 0$ is $0.jpg$. At that time AVRA has already learned through supervised learning two contexts `Animals` and `Colors`. Each context contains two keywords. The `Animals` context contains keywords `cat` and `mule`, while the `Colors` context contains the keywords `red` and `green`. At timestamp $t = 1$, AVRA detects user action g, where the word `dog` is searched for in a browser. The image recorded at timestamp $t = 1$ is $1.jpg$. At time $t = 3$ the unsupervised learning algorithm makes the connection that the text `dog` led to the action `Search(dog)`. Moving to the top right cell in Figure 9.3, the algorithm loads the hyperparaemters $h1$ and $h2$ as 0.65. Next, the fit of the text `dog` is computed in comparison to the average vector

for the context `Animals`, with a result of 0.8. The fit of `dog` with the average vector for the keywords in context `Colors` computes to 0.1. 0.*jpg* is then compared with 1.*jpg* using a perceptual hash to detect if the images are close enough together for 1.*jpg* to be representative of the potential new context. The image difference is too great, and so only 0.*jpg* is used in the next step. The image fit is calculated by passing 0.*jpg* to the CNN for classification. It outputs that 0.*jpg* strongly activates the context `Animals` (0.9 - perhaps detecting the eyes and other body features common to all animals), and also activates the context `Colors` (0.7 - perhaps picking up on the fact that the image of the dog is mostly one solid white color). The ratio for both the `Animals` and `Colors` contexts was 1.0, and so the ratio did not modify the decision at the output in this case. The overall fit of keyword `dog` into context `Animals` was 0.72, exceeding the threshold of 0.65. At 0.07, the overall fit of keyword `dog` into context `Colors` did not exceed the threshold of 0.65, and so it was discarded. With only one context vying to accept the new keyword `dog`, it was added to the context `Animals`.

In the second example of Figure 9.5, AVRA sees an image of a dog and the text `dog`. The image at timestamp for $t = 0$ is 0.*jpg*. At that time AVRA has already learned through supervised learning two contexts `Shapes` and `Colors`. Each context contains two keywords. The `Shapes` context contains keywords `round` and `line`, while the `Colors` context contains the keywords `red` and `green`. At timestamp $t = 1$, AVRA detects user action g, where the word `dog` is searched for in a browser. The image recorded at timestamp $t = 1$ is 1.*jpg*. At time $t = 3$ the unsupervised learning algorithm makes the connection that the text `dog` led to the action `Search(dog)`. Moving to the top right cell in Figure 9.5, the algorithm loads the hyperparaemters $h1$ and $h2$ as 0.65. Next, the fit of the text `dog` is computed in comparison to the average vector for the context `Shapes`, with a result of 0.0. The fit of `dog` with the average vector for the keywords in context `Colors` computes to 0.1. 0.*jpg* is then compared with 1.*jpg* using a perceptual hash to detect if the images are close enough together for 1.*jpg* to be representative of the potential new context. The image difference is too great, and so only 0.*jpg* is used in the next step. The image fit is calculated by passing 0.*jpg* to the CNN for classification. It outputs that 0.*jpg* activates the context `Shapes` (0.6), and also activates the context `Colors` (0.7). The ratio for both the `Shapes` and `Colors` contexts was 1.0 because all of the keywords in each context was used to compose their average vector. The overall fit of keyword `dog` into context `Shapes` was 0.0, below the threshold of 0.65. At 0.07, the overall fit of keyword `dog` into context `Colors` also did not exceed the threshold of 0.65. With no remaining context into which the keyword `dog` can be trained, a new

context `newContext` was created and the keyword `dog` was added to it. The arbitrary name `newContext` reflects the fact that AVRA does not know what the overall context is going to store in the future.

A problem surfaced when assessing AVRA's ability to learn new *keywords* into existing contexts created through supervised learning. The word embedding component of the unsupervised learning algorithm was mostly unsuccessful adding to the supervised learning data. It emerged that the problem was the ratio. The ratio is small when many of the keywords from supervised learning (e.g. nullpointerexception) are not contained in the word embedding model generated from the Google News dataset [147], or other general word embedding models. The model in question contains 3 million words, but this was not sufficient. One approach to force the unsupervised learning model to work was to ignore the ratio, setting it to 1.0 instead of calculating the correct value. Setting ratio to 1.0 is of course a sub-optimal solution, but with this approach AVRA was able to learn new *keywords* into existing contexts created through supervised learning.

A better approach to learn new *keywords* into existing contexts created through supervised learning was to re-purpose (METHOD 2) to collect website contents instead of images. The idea was to scrape text from search engine results (web pages), where the search query is built using the keywords AVRA knows about, and the new keyword AVRA wants to classify into a new or existing context. Using the text from these web pages as a corpus, one can train a new word embedding model that can relate a high ratio of the keywords to each other. The drawback of this approach is that scraping the pages and training the word embedding model is very slow. To build a model relating n words to each other in pairs requires P_2^n search operations. For groups of three keywords this becomes P_3^n, and for groups of four keywords P_4^n. The time required to perform these operations is bounded by $O(n!)$, which grows quickly. It can take days even for $n < 1000$ executed on a parallel cloud computing system. Luckily, the unsupervised learning process can be trained as an offline server-side process without slowing down the user experience at all, and the search results can be cached and extended over time. To build the corpus, the first 30 results for each search term was downloaded. Search terms were composed of distinct sets of 3 or 4 keywords (e.g. *horse baseexception chicken*). Stopwords were removed using the NLTK Stopwords corpus by Porter [180] [32, page 47]. Stemming was applied using the `PorterStemmer` module of the `gensim` library [187]. Some web pages from the results were skipped because the server refused the connection, the file on the server was not a web page (e.g. PowerPoint file), or the page contained no text. Each valid link returned by the search engine was processed into a text string,

and all of the results were concatenated into a single corpus tokenized by spaces. Next, a word embedding model was trained on the corpus, exposing the similarity between words by computing the cosine similarity in the word embedding model between vectors for words.

At this point in the thesis, it has been described to the reader how AVRA has the ability to make sense of new words by building its own word embedding models completely unsupervised. Crucially, supervised and unsupervised learning can be combined in AVRA, to accomplish transfer learning. AVRA can learn new things autonomously as long as the related keywords are trained into one of the word embedding models that informs the textual context relationships between them, or the information relating these concepts is obtainable by building a word embedding using documents obtained by searching on the web. Multiple word embedding models could be used by AVRA as a general knowledge reference.

9.4 Performance Evaluation for Unsupervised Learning

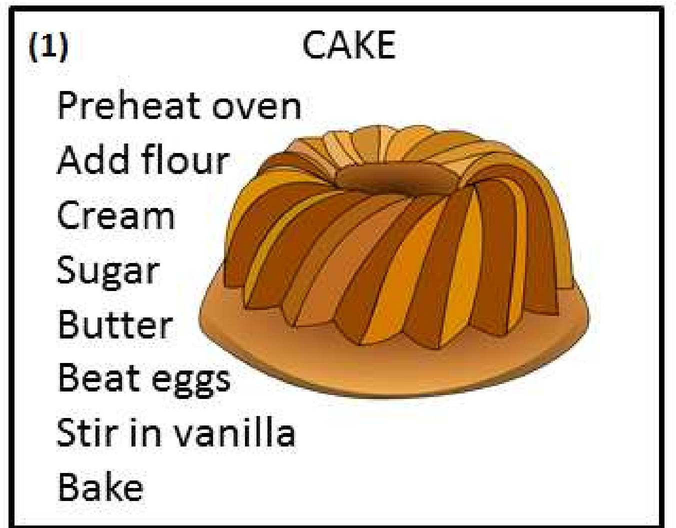
Learning in AVRA is data driven. In this Section the operation of AVRA's unsupervised learning algorithm with real data is explored. A high-level view of a word embedding model for several contexts is presented to clarify the ability of AVRA to classify new keywords into existing contexts. Visualization for AVRA's image recognition system similarly reveals that AVRA can successfully discriminate between different sets of images. To collect data quickly, a test automation program was used to model a user using the computer (a 'bot'). This bot was used to carry out use cases such as extending an existing context with new information, and creating a new context in AVRA's model. Examples of extending an existing context and creating a new context are provided as validation of the AVRA prototype's ability tpo apply unsupervised learning.

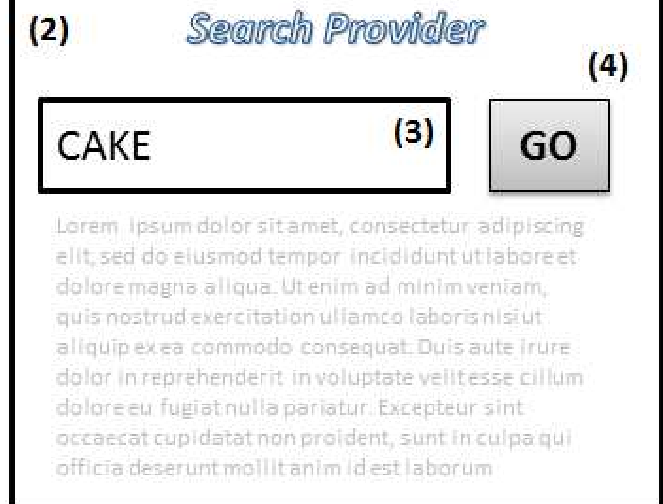
Visualizing Unsupervised Keyword Input Vectors and Images

t-SNE was used to get a sense for what the word embedding model represents. As shown in Figure 9.7, t-SNE clusters into a two dimensional space the 300 dimensional vectors for four contexts (`eclipse`, `console`, `gene`, and `desktop`). The average vectors for these contexts are represented by the locations of the large labels. The separation of the average vectors confirms that the model can classify a new point based upon the

cosine distance between the new point and each average vector.

In addition to the contexts discussed in Chapter 4 (`eclipse`, `console`, `gene`, `desktop`, and `facebook`), images for the contexts `daisy`, `dendelion`, `roses`, `sunflower`, and `tulip` from a TensorFlow tutorial were trained into contexts as well[178]. They were trained into AVRA's CNN to show how many contexts are recognized in relation to each other. Figure 9.8 shows a 3D t-SNE plot with 5 images from each context (7 images for console). Figure 9.7 provides another view of the data in 2 dimensions, revealing that images from the same context cluster together. This indicates that the classifier can differentiate between the image classes (onscreen contexts).





Bot opens recipe web page

Bot opens search web page, fills in search box, and presses search button

Figure 9.6: Example of the bot traversing screens to generate user activity data

Automated Data Generation for Unsupervised Learning

The bot created to carry out use cases was implemented using the `pyautogui` library [229]. The bot was configured to traverse screens and click buttons. To generate data, the bot was programmed to (1) open a web page (e.g. a recipe web page describing how to bake chocolate cupcakes) and then (2) open a second web page containing a search engine. Next (3) the bot would fill in the search box, and (4) click the search button

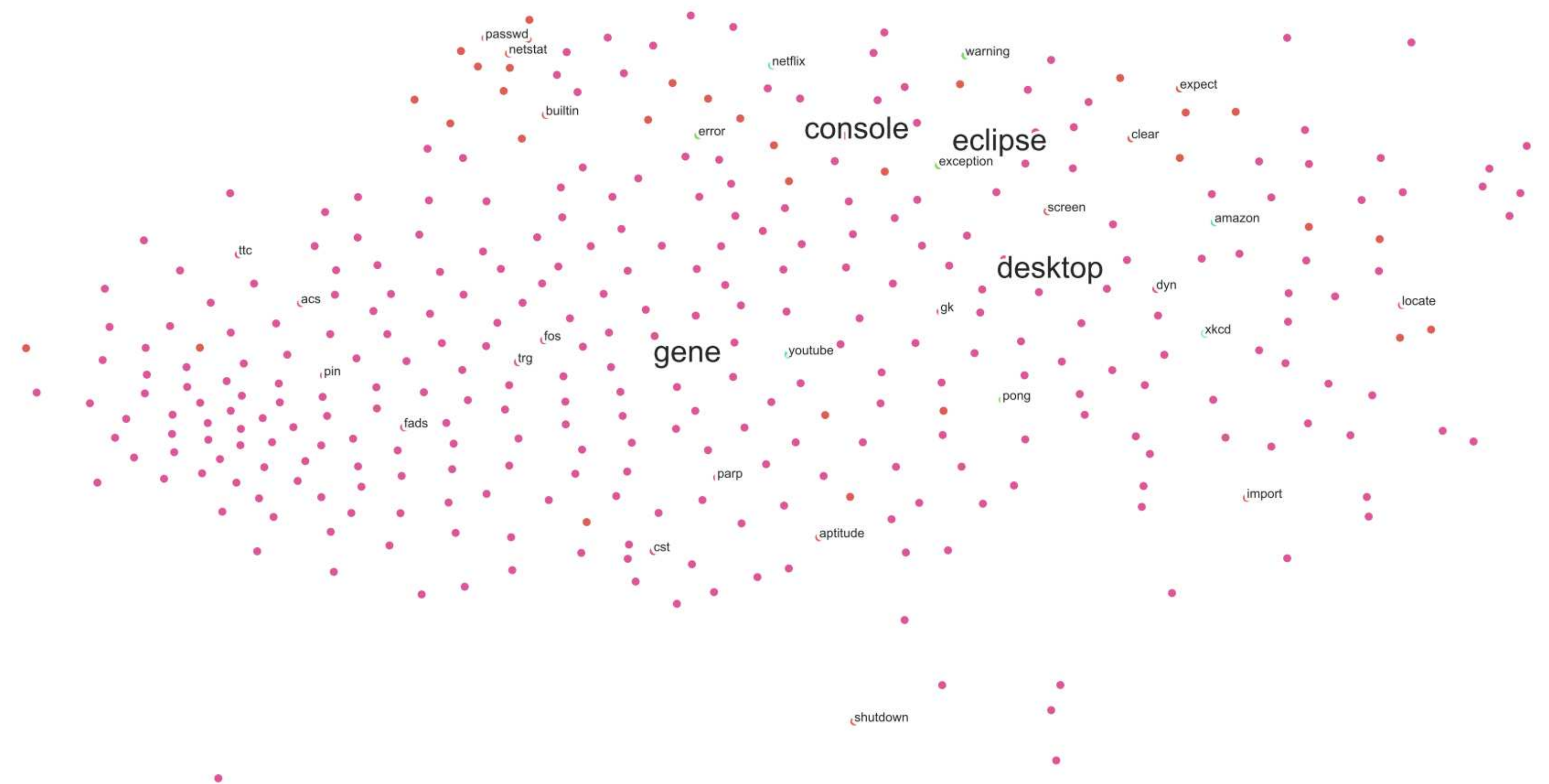


Figure 9.7: t-SNE representation of keywords from 4 contexts in 2D space. Each dot in the figure represents a keyword. A few dots are labeled to give the reader a sense for what the dots represent. The average vectors for these contexts are represented by the locations of the large labels. Only keywords with vectors in the Google News word2vec model were used to train this t-SNE model. The default settings were used for perplexity (30). The model was executed for 1,000 iterations.

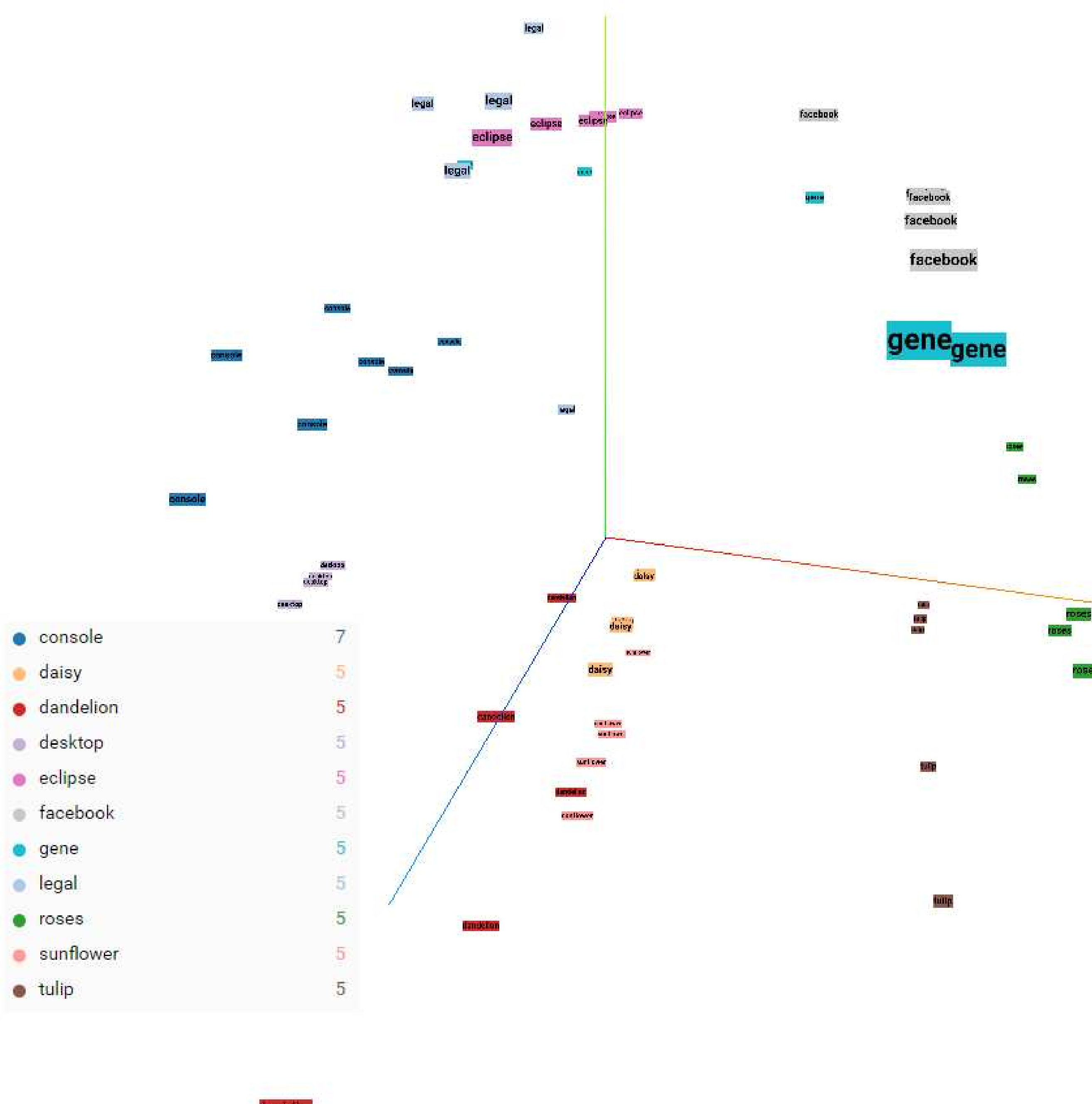


Figure 9.8: t-SNE representation of the feature vectors for images representing 11 contexts in 3D space. Each label in the figure represents an image. The text and color indicate the context the image belongs to. The clustering of the images by context indicates that the model learned to recognize and differentiate between the contexts. The default settings were used for perplexity (30). The model was executed for 2,936 iterations.

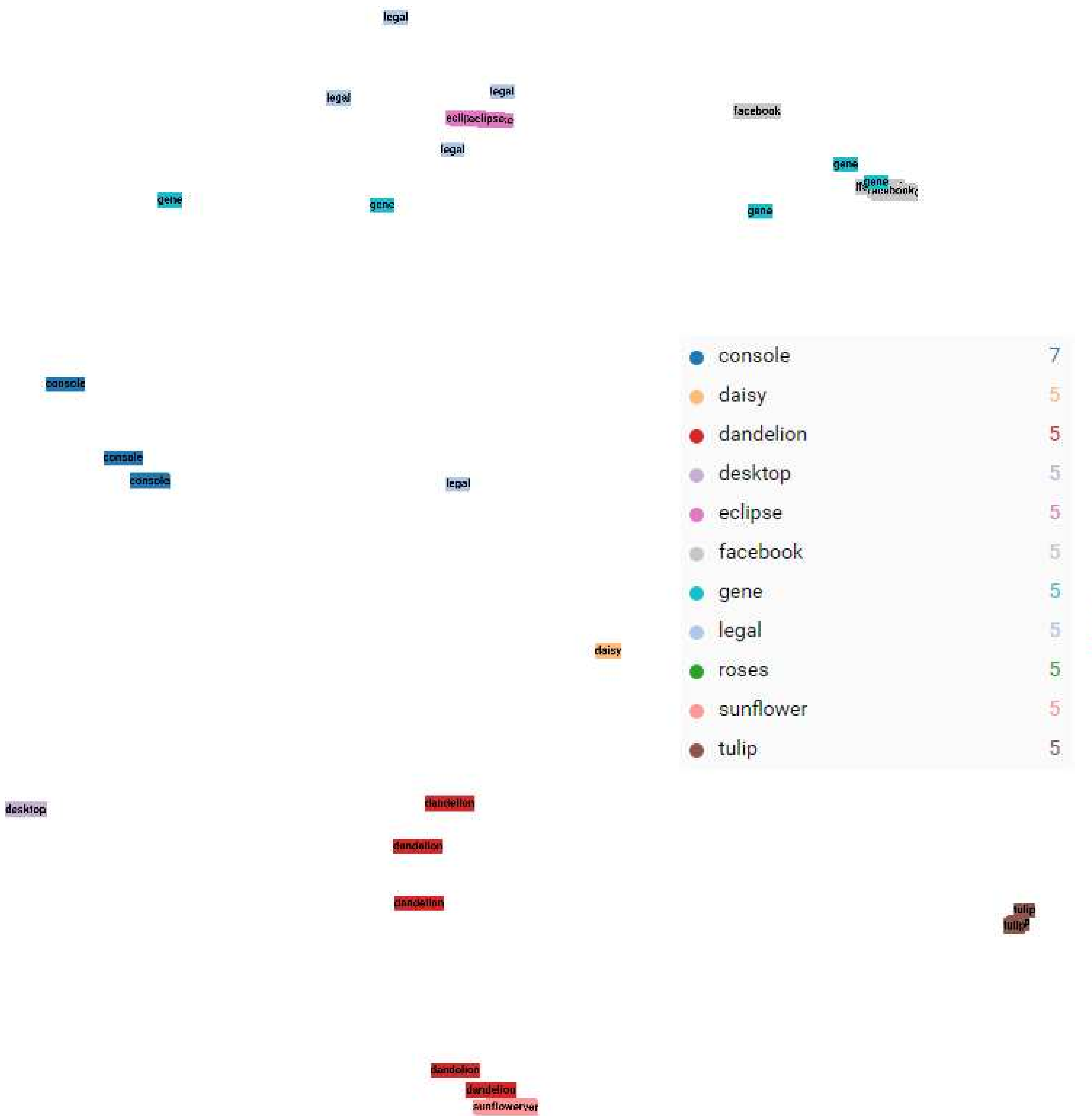


Figure 9.9: t-SNE representation of the feature vectors for images representing 11 contexts in 2D space. Each label in the figure represents an image. The text and color indicate the context the image belongs to. The clustering of the images by context indicates that the model learned to recognize and differentiate between the contexts. The default settings were used for perplexity (30). The model was executed for 2,936 iterations.

and observe the results. Each loop of actions (1) through (4) depicted in Figure 9.6, represents an instance of the user observing onscreen information and then looking it up on the internet in a browser window. The purpose of the bot was to generate data while AVRA runs in the background, and then to test if AVRA learned to recommend the actions it observed when the context and keyword were later present onscreen.

Unsupervised Learning Extending Existing AVRA Context

Table 9.1 presents a real example to show how AVRA extends the Eclipse IDE context created using supervised learning. For this example, 5 existing contexts were included in the computations, to give the reader a sense for the computations AVRA performs without overwhelming the reader with many contexts and keywords. Setting the stage for this example, the bot moved from the Eclipse IDE where it ran a program and generated an error, to a browser window where it typed and searched for keywords related to this error message. All the while, AVRA was running in the background. When AVRA observed the causal relationship between the onscreen error in the IDE, and the search action in the browser, it stored the data in the AVRA database. When sufficient copies of the action were observed, the unsupervised learning algorithm was triggered to try and learn the new keywords into AVRA's RS.

Starting with the first row of Table 9.1, AVRA found that images when *thread* was onscreen strongly activated the *eclipse* context (0.95) and that word *thread* was semantically similar to the keywords in *console* (0.28), *eclipse* (0.15), *desktop* (0.30), and *gene* (0.25). Because only one context demonstrated sufficient image and word similarity, the new keyword *thread* was trained into AVRA for the *eclipse* context. Continuing with the second row of Table 9.1, AVRA found that images when *exception* was onscreen strongly activated the eclipse context (0.94) and that word *exception* was semantically similar to the keywords in *console* (0.16), and *eclipse* (0.51). Because, once again, only one context demonstrated sufficient image and word similarity, the new keyword *exception* was trained into AVRA for the *eclipse* context. The unsupervised learning algorithm in AVRA does make mistakes. For example, in the third row of Table 9.1, AVRA recognized the images of the Eclipse IDE but just missed the hyperparameter cutoff of 0.10 to consider the keyword *throwing* semantically similar to the *eclipse* context. Instead of learning *throwing* into *eclipse*, AVRA incorrectly learned the keyword into a new context.

Table 9.1: Extending an existing context after observing the user. A **bolded** result in the table below indicates a result that exceeded the required threshold.

Event	Context Similarity	Word Clustering	AVRA Decision	Correct?
New keyword *thread*	*console* 0.02 **eclipse 0.94** *desktop* 0.01 *facebook* 0.02 *gene* 0.00	**console 0.28** **eclipse 0.15** **desktop 0.30** *facebook* 0.00 **gene 0.25**	Train *thread* into *eclipse*	YES
New keyword *exception*	*console* 0.02 **eclipse 0.94** *desktop* 0.01 *facebook* 0.02 *gene* 0.00	**console 0.16** **eclipse 0.51** *desktop* 0.01 *facebook* 0.00 *gene* 0.01	Train *exception* into *eclipse*	YES
New keyword *throwing*	*console* 0.02 **eclipse 0.94** *desktop* 0.01 *facebook* 0.02 *gene* 0.00	**console 0.14** *eclipse* 0.09 *desktop* 0.01 *facebook* 0.00 *gene* 0.01	Image clustering. Train new context for *throwing*	NO

Unsupervised Learning Creating New AVRA Context

Table 9.2 presents a real example to show how AVRA creates a new context using unsupervised learning. For this example, 5 existing contexts were included in the computations. Prior to the events listed in Table 9.2, the bot moved from the a browser window containing a cake recipe to a browser search window where it typed and searched for keywords related to the recipe (*chocolate*, *cake*, and *cupcake*). All the while, AVRA was running in the background collecting images and extracting onscreen text. When AVRA observed the causal relationship between the onscreen recipe text, and the search actions in the browser, it stored the data in the AVRA database. When sufficient copies of the action were observed, the unsupervised learning algorithm was triggered to try and learn the new keywords into AVRA's RS.

Starting with the first row of Table 9.2, AVRA found that images of a recipe website taken when the word *cupcake* was onscreen strongly activated the facebook context (0.90) and that word *cupacke* was semantically similar to the keywords in *desktop* (0.18). Because no context demonstrated sufficient image and word similarity, a new context was created in AVRA. Continuing with the second row of Table 9.2, AVRA found that images captured when the word *chocolate* was onscreen strongly activated the $facebook$ context (0.90) as well as the new context *newContext* (0.70). The word *chocolate* was semantically similar to the keywords in *desktop* (0.12), and *newContext* (0.55). Because only one context demonstrated sufficient image and word similarity, the new keyword *chocolate* was trained into AVRA for the *newContext* context. For the third row of Table 9.2, AVRA recognized the images of the recipe website, and considered the keyword *cupcake* semantically similar to the context *newContext*. AVRA learned the keyword *cake* into the correct context. This example shows that when the risk of concept drift is highest, for a new context with only a few keywords, AVRA does generally manage to build up the new context. There are cases such as the third row in Table 9.1, where keyword clustering or image clustering fails to group an action into an existing context where it belongs, fracturing the context into two (or more) contexts.

Unsupervised Learning Relationships Between Keywords

It is interesting to ask how long it takes to train a new word embedding model for a new *keyword* (e.g. "nullpointerexception") that is not in AVRA's default model, and how well that model works, given the fact that the raw data was built through analyzing web pages returned by a search engine.

Table 9.2: Creating a new context after observing the user. A **bolded** result in the table below indicates a result that exceeded the required threshold.

Event	Context Similarity	Word Clustering	AVRA Decision	Correct?
New keyword *cupcake*	*console* 0.02 *eclipse* 0.04 *desktop* 0.03 **facebook 0.90** *gene* 0.01	*console* 0.07 *eclipse* 0.00 **desktop 0.18** *facebook* 0.00 *gene* 0.05	Image clustering. Train *cupcake* into new context *newContext*	YES
New keyword *chocolate*	*console* 0.02 *eclipse* 0.04 *desktop* 0.03 **facebook 0.90** *gene* 0.01 **newContext 0.70**	*console* 0.04 *eclipse* 0.00 **desktop 0.12** *facebook* 0.00 *gene* 0.05 **newContext 0.55**	Train *chocolate* into *newContext*	YES
New keyword *cake*	*console* 0.02 *eclipse* 0.04 *desktop* 0.03 **facebook 0.90** *gene* 0.01 **newContext 0.70**	*console* 0.05 *eclipse* 0.00 **desktop 0.12** *facebook* 0.00 *gene* 0.05 **newContext 0.60**	Train *cake* into *newContext*	YES

To evaluate the ability of the generated model to classify new entities, two small sets of related keywords were created for testing purposes: `Animals` ($horse, dog, cow, pig$) and `Java` ($baseexception, exception, standarderror, importerror$), and the similarity (from the cosine distance) to a new keyword `chicken` was measured. An effective model should find that `chicken` has a lower cosine distance to the average vector for `Animals` than it does compared to the average vector for `Java` keywords.

The $23MB$ corpus of text was downloaded and trained in approximately 30 minutes for 9 keywords. $1,744$ of the links produced usable text. In the collected corpus, the frequency of the stemmed keywords was as follows: $hors(8,137)$, $dog(14,412)$, $cow(4,914)$, $pig(9,933)$, $baseexcept(434)$, $except(9,109)$, $standarderror(256)$, $importerror(544)$, $chicken(6,252)$. The average sentence length was 110.9 characters, and $47,566$ distinct keywords were translated into word vectors in the trained model. The model was created under various hyperparameter configurations (random seed value, training iterations between 5 and 50, context window size between 10 and 40) and each configuration was tested 10 times. All of these measurements resulted in assignment of the new keyword `chicken` to the context `Animals`. Generally, there was a negative similarity between the keyword `chicken` and the context `Java`, while there was always a positive similarity between the keyword `chicken` and the context `Animals`. Very surprisingly, the outcome was positive even when the number of dimensions (also called the number of features) used to represent word vectors was varied between 10 and 100. Table 9.3 provides a look at some of the raw data, showing the similarity of `chicken` to each of the keywords in AVRA's model.

9.5 Chapter Summary

This chapter presented AVRA's unsupervised learning approach and explained with examples how AVRA combined supervised and unsupervised learning to accomplish transfer learning. These examples support the thesis statement that a deep learning artificial intelligence can provide action recommendations related to onscreen messages using unsupervised learning.

Table 9.3: New keyword unsupervised learning from word embedding.

# Features	sim(chicken, horse)	sim(chicken, dog)	sim(chicken, cow)	sim(chicken, pig)	sim(chicken, baseexception)	sim(chicken, exception)	sim(chicken, standarderror)	sim(chicken, importerror)	Average * 100 for Animals	Average * 100 for Java	Chicken in Animals
10	-0.02	0.57	0.83	0.74	0.23	0.45	0.29	-0.06	52.88	22.74	True
10	0.26	0.48	0.83	0.75	-0.19	-0.15	-0.16	0.04	58.03	-11.50	True
10	0.01	0.47	0.87	0.73	-0.38	-0.21	-0.36	0.03	51.84	-22.70	True
10	0.10	0.72	0.78	0.72	0.11	0.20	0.13	0.14	57.76	14.46	True
10	-0.02	0.63	0.80	0.83	-0.05	0.07	0.04	0.13	56.13	4.74	True
20	-0.01	0.39	0.78	0.68	-0.08	0.14	-0.03	0.06	45.75	2.18	True
20	0.12	0.33	0.79	0.62	-0.18	0.04	-0.13	0.04	46.50	-5.71	True
20	-0.01	0.23	0.84	0.58	-0.02	0.13	0.00	0.17	40.84	7.02	True
20	0.08	0.29	0.77	0.65	-0.02	0.12	0.00	0.11	44.86	5.28	True
20	0.07	0.26	0.82	0.64	-0.14	-0.02	-0.13	0.14	44.61	-4.02	True
30	0.04	0.18	0.79	0.59	-0.04	0.22	-0.01	0.09	39.98	6.42	True
30	0.03	0.16	0.79	0.52	-0.08	0.03	-0.07	0.08	37.41	-1.06	True
30	0.08	0.19	0.81	0.57	-0.08	0.08	-0.08	0.08	41.20	-0.17	True
30	0.06	0.18	0.77	0.55	-0.05	0.14	-0.03	0.10	39.12	3.81	True
30	0.03	0.21	0.79	0.52	-0.05	0.11	-0.04	0.09	38.78	2.70	True
40	0.06	0.21	0.77	0.47	-0.10	0.04	-0.11	0.11	37.81	-1.41	True
40	0.08	0.23	0.73	0.49	-0.03	0.16	-0.05	0.06	38.38	3.41	True
40	0.09	0.18	0.81	0.55	-0.07	0.14	-0.08	0.09	40.50	1.90	True
40	0.04	0.21	0.73	0.49	-0.04	0.18	0.01	0.07	36.79	5.22	True
40	0.04	0.25	0.76	0.51	-0.02	0.17	-0.01	0.13	38.89	6.64	True

Chapter 10

Epilogue

This thesis concludes with a summary of the future work to continue the development of AVRA, and a reflection on the claims of the thesis statement in light of the previous chapters.

10.1 Thesis Conclusion

This work presented a new way to create virtual assistants. The following novel methods were described: an unsupervised learning algorithm, a fast text filtering algorithm, a deep learning text classifier, and a recommendation filtering algorithm. The development of a recommendation system for personalized reasonable response time 1-click task recommendations was presented. The AVRA system mines information from screen capture data, rather than interfacing with individual applications. The developed system supports multiple OSes, application domains, and configurations. Recommendations are presented to the user in an intuitive 3-button user interface. Various tools related to the development of AVRA's design were discussed, and the performance of AVRA was characterized.

The thesis questions began with the question "Is it possible for a virtual assistant for personal computer users to interpret images of the computer screen to provide action recommendations?" Chapter 3 proved that yes, it is feasible to create such a system. The second thesis question asked "Could such a system learn unsupervised?" The answer is yes, as described in Chapter 9. Expanding on Chapter 3, Chapters 5, 6, 7, 8, and 9 answered the thesis question "How could one implement a proof of concept for this system?"

The hypotheses proposed in the thesis statements have been confirmed in this work.

The first statement of the thesis was that a deep learning artificial intelligence can provide useful action recommendations related to onscreen messages. In Chapter 3.7 the ability to provide recommendations based upon onscreen *keywords* was verified and characterized in terms of accuracy and precision. The utility of the recommendations was further demonstrated with specific use cases in Chapter 4.

The second statement of the thesis was that the action recommendations can be provided without integration into each individual program executing on the computer. The system design described in detail in Chapter 3 provides a detailed explanation for how action recommendations can be provided to the user without integration into individual programs running on the computer.

The third statement of the thesis was that the action recommendations can be provided within a reasonable response time, and can be acted upon with a single mouse click. Chapter 3.7 provided proof that AVRA can return results to the user in within a reasonable response time. The neural network training method that enables action recommendations to be encoded into a form that can be launched with one click was described in Chapter 7.1. The approach was to map each learned *keyword* to a specific action apriori.

Finally, the last statement of the thesis was that the action recommendations can be personalized to the user by utilizing the user history and unsupervised learning. The personalization mechanism in the RS was described in Chapter 8 and characterized in Chapter 3.7.2. The unsupervised learning algorithm was described and characterized in Chapter 9.

The takeaway message from this work is that without integration into each individual program executing on the computer, software can provide useful and personalized action recommendations related to onscreen messages. These action recommendations can be provided within a reasonable response time, and can be acted upon with a single mouse click. Unsupervised learning can be used to automate the training of this system.

10.2 Proposed Future Work

Future work beyond the conclusion of the thesis is described in this chapter. Future work on AVRA will involve speeding up the individual component execution time in order to reintegrate the DNN, additional supervised learning, modeling capabilities, and an examination of privacy considerations as described below.

Text processing with a neural network was implemented for this work but was removed from AVRA after it became clear that the execution time of the DNNs did not scale with the number of contexts, and the response time was too slow even for a small number of contexts. Acceleration of the DNN implementation, perhaps requiring a redesign, would be the ideal. Replacing the RS and text processing with neural components would result in an end-to-end neural system. Such a system would be interesting to study, as it could in principle perceive input from the computer screen, learn from that input without supervision, and recommend actions. This idea could perhaps be implemented inside a physical robot body with a camera that looks at the computer screen and perceives information.

Adding additional domain-specific data to the initial corpus of knowledge AVRA ships with will improve the user experience and the tool's overall utility. Specifically, this involves training additional data into the system by integrating data from [154] and other similar programming formula providers into the recommendation generating database of the DNN outputs.

There are several interesting directions in which this research can proceed regarding increasing AVRA's modeling capabilities. First and foremost, AVRA should include a feature for text parsing across lines using bounding boxes. Detecting window shape and the association of text to the bounding box is solvable with existing deep learning image processing technology. This would resolve the issues observed in Chapter 4. AVRA's screen classifier could then make two predictions when the screen is split between two applications (e.g. eclipse and chrome), leveraging automated segmentation of the screen image to associate specific text to each context. Expansion of AVRA's input processing to include recognition of symbolic representation of the computer's state is one such direction. This idea involves recognizing images in addition to keywords. Regarding this increased input processing, at this time only textual *keywords* are detected by AVRA to produce recommendations. Understanding additional types of *keyword* information may involve processing auditory signals (e.g. speech and music) as well as graphical hints (e.g. an open lock indicating an SSL certificate problem). Regarding processing graphical *keywords*, CNNs could be evaluated for the classification of objects within context, and related recommendations could then be generated. Content-based image recognition and semantic segmentation of images will be key topics in further investigating the processing graphical *keywords*.

Another interesting direction for the research that was touched on briefly in the thesis is a more sophisticated symbolic representation of the computer's state, such as

the application of ontologies and knowledge bases to improve recommendation diversity. This is useful because a virtual agent that understands the relationship between recommendations can do a better job of proposing 3 different recommendations rather than 3 very similar ones. Perhaps wide and deep learning could replace the RS and DNNs described in this work. Following such an approach would require a new approach to forgetting recommendations over time, and would need to balance execution time with recommendation score.

Adding state information to AVRA's RS should be investigated and developed further in future work. AVRA makes the Markovian assumption about opportunities to recommend tasks based upon onscreen information. However, this is clearly a simplifying assumption that does not hold in general. For example, interpreting video frames requires knowledge of current and past states of the computer screen. This is also true of events observed before and after the user moves a window or changes views with the ALT+TAB command. Many such examples come to mind. Now, to remove the Markovian assumption would require some sort of memory system such as an LSTM, or a generative predictive model such as an RNN. Perhaps a deep reinforcement learning system could be applied to this problem. The representation of time-domain information, regardless of the mechanism and algorithms employed to accomplish the task, would enable the virtual agent to model sequences of events and to make predictions about actions prior to the triggering information appearing onscreen. Successive snapshots of the computer screen could be used to generate recommendations based upon sequences of detected computer screen states rather than the current approach of time-invariant analysis of the computer screen driving action recommendations. The first steps in that direction are already in progress as the unsupervised learning algorithm of Chapter 9 requires knowledge of the sequence of events in order to learn. Another advantage that RNN/LSTM would provide is long-term predictive capability such as recommendations based on the time of day, and understanding long sequences of actions by the user as distinct patterns.

Regarding collaborative filtering, the current approach of storing user data in the cloud makes it possible to implement collaborative filtering of recommendations, which may be implemented in upcoming versions of AVRA. Perhaps a Bayesian network will be employed to integrate user recommendations from one user's RS database into the RS of other users.

Finally, the privacy considerations required to deploy AVRA should be investigated in detail, as well as the privacy implications of such systems being adopted. As mentioned

earlier in this work, AVRA retains images of computer screens which may be compromising to the user if released. It may be insufficient to limit the aggregation of personal data as images can sometimes contain personal identifying information. Furthermore, the collaborative recommendation generating system must avoid learning the bad behavior of one or more users being recommended to other users. This type of problem comes up in search engine recommendations as many users search for lewd topics but the search engine does not want to recommend these searches to other users. These privacy considerations should be addressed in future work.

Appendix A

Glossary of Terms

Accuracy of Classification The true positive rate divided by the number of samples

(ANN) Artificial Neural Network A computational model of a neuron which combines many individual neurons into a network [198]

(AVRA) Automated Virtual Recommendation Agent AVRA is a recommender system that assists the user by producing action recommendations regarding on-screen tasks

(AWS) Amazon Web Services A cloud computing infrastructure service

(BEP) Break Even Point The point where precision is equal to recall [199, page 161] [23]

(CBF) Content-Based Filtering A recommender systems scoring approach [24] [175]. CBF provides a mechanism to rank recommendations where the recommendation score is increased if items related to the recommendation in question were rated positively in the past

(CER) Character Error Rate proportion of characters in a dataset that were incorrectly classified during image to text conversion [38, page 51] [61]

(CFTR) Cystic Fibrosis Transmembrane conductance Regulator The CFTR gene codes for the CFTR protein, which is related to the Cystic Fibrosis disease process

(CNN) Convolutional Neural Network A CNN is a deep learning image classifier which applies many small filters to an image in order to produce a hierarchical understanding of the contents of the image [198] [120]

Context Each visual pattern recognized by the CNN in this work as a distinct class is called a context. These contexts can be thought of as visual topics. The context is used to consider what the computer screen looks like when a given keyword of interest is detected

(CPU) Central Processing Unit

(CWL) Charter Word List A dataset representing natural English text collected from the set of each word in the Canadian Charter of Rights and Freedoms [9]

(DBN) Deep Belief Network A stack of fully connected RBMs where each layer accepts pattern representations from the level below it, and learns to encode these patterns into output classes [198]

(DL) Deep Learning DL is an approach in artificial intelligence using many layers of neurons in a neural network to implement machine learning. The hallmarks of DL are the formation of hierarchical representations of data within the layers on the neural network, and the training of the neural network using a learning algorithm [70]

(DNN) Deep Neural Network A DNN is a fully connected multi-layer neural network classifier

Dropout Dropout is used to improve neural network training by providing the training learner with an approximate representation of the ground truth data. With dropout, the connection from one neuron to the next neuron is dropped, with some specified probability. The main advantage of this approach is reducing overfitting. See also, Random noise injection

Epoch One iteration of the optimization process

(GB) GigaByte

(GNoT) Google Now on Tap GNoT is a general purpose on-demand search tool for mobile phones that is meant to intervene only when prompted by the user mobile phones. GNoT recognizes multiple items in one image, recommending further information or actions in the form of onscreen “cards”

(GP-GPU) General Purpose Graphics Processing Unit A hardware accelerator typically applied to speed up computations as compared to CPU processing of the

same machine learning task. GPUs contain deep pipelining and parallel processing of operations

(GUI) Graphical User Interface In AVRA, the GUI consists of a set of 3 buttons which the user can click on in order to trigger the actions described in the button

(JEL) Java Error List An error message text corpus containing a set of 248 error messages produced by the JVM.

(JVM) Java Virtual Machine

K-means K-means is one of several non-neural clustering approaches where the centroid for each cluster is adjusted for many iterations until the points nearest to each centroid are closest to their assigned centroid (and therefore not closer to the centroid of another cluster) [172, page 5]

Keyword A text string containing one or more words is a keyword. For example, the error message "Error: NullPointerException" can be one keyword

(LSH) Locality Sensitive Hashing LSH is used for comparing the similarity of text or images. Strictly speaking, LSH is a one way hash function which quickly reduces an input string or image to a fixed-length hash with the special property that similar strings (or images) generate hash outputs with low Hamming distance between them [99]

(LSTM) Long Short Term Memory A specialized RNN that can retain of forget information

(MAP) Mean Average Precision A metric used to assess an entire precision-recall curve, gives more weight to correct results the higher they are ranked

Memoization With the goal of reducing program execution time, memoization stores the results of function calls for specific input values into memory, returning the result without computing it if the inputs to the function match an already stored result

(MI) Mixed-Initiative In MI systems, the breakdown of work between the computer and the human focuses on the strength of each participant in the iterative problem solving activity [13]

(MOE) Measures of Effectiveness MOE define user needs, and are used to evaluate how an implementation performs in comparison to the stated objectives, regardless of implementation details [189]

(MOP) Measures of Performance MOP defines the user requirements, and specify the technical performance characteristics the system must meet. Each MOP should relate back to one or more MOEs [189]

Morse A text encoding scheme using dots, dashes and spaces [4]

(NLP) Natural Language Processing The study of understanding text or speech spoken or written by humans in a natural way

(NLU) Natural Language Understanding More narrow than NLP, NLU is the process of a system understanding unstructured text or speech input and extracting intent as well as the related entities

Objective The function that is being maximized or minimized in an optimization process

(OCR) Optical Character Recognition Optical Character Recognition software translates an image containing text into text, sometimes with translation errors

Precision The rate of true negative classification of data is called precision [191, page 780]. See Equation 2.1

(RBM) Restricted Boltzmann Machine A layer in a neural network where the neurons do not connect to each other (the restriction) but rather accept input from a previous layer (or the input to the overall network), and then feed this information forward (feed-forward) [218]

Random noise injection Noise injection is used to improve neural network training by providing the training learner with an approximate representation of the ground truth data. With random noise injection, the connection from one neuron to the next neuron is modified by some statistical noise, with some specified probability. The main advantage of this approach is reducing overfitting. See also, Dropout

Recall The rate of true positive classification of data is called recall [191, page 902]. See Equation 2.2

(RNN) Recurrent Neural Network A neural network architecture where the output is fed back into the input, as opposed to an RBM where information is strictly fed forward. By feeding information back to the input, the current state can be based upon past states of the network

(ROC) Receiver Operating Characteristic A graphical method of assessing the trade-off between the sensitivity and specificity probabilities of a classifier [137]

(RS) Recommender System RS recommend items to a user, often based upon some data about the user and the items to be recommended

(SCC) Supreme Court of Canada

(SL) Supervised Learning Training a machine learning system classifier using many labeled examples

Specificity The true negative rate for a classifier is called specificity [191, page 902]. See Equation 2.3

(t-SNE) t-Distributed Stochastic Neighbor Embedding t-SNE is a dimensionality reduction tool [130]

(TC) Text Categorization The assignment of category labels to a document

(TER) Term Error Rate TER measures the error rate for specific terms

(TPU) Tensor Processing Unit A cloud-based hardware accelerator for training and executing machine learning models.

(UL) Unsupervised Learning learning without labeled examples organized into a dataset [88]

(VM) Virtual Machine A cloud computing term referring to the configuration settings defining the virtual settings of a computer which is in fact executing on a shared hardware system in a cloud computing environment. Whereas a virtual machine can be instantiated and destroyed by software, a real machine is composed of physical parts

(WER) Word Error Rate The proportion of words in a dataset that were incorrectly classified during image to text conversion [61]

Bibliography

[1] Frontpage - tkinter wiki. http://tkinter.unpythonic.net/wiki/. [Online; accessed 2017-01-19].

[2] Imagenet. http://www.image-net.org/. [Online; accessed 2017-01-17].

[3] ponty/pyscreenshot: python screenshot. https://github.com/ponty/pyscreenshot. [Online; accessed 2017-01-19].

[4] International morse code. Technical Report Recommendation ITU-R m.1677-1, International Telecommunication Union, October 2009.

[5] SS64.com. An A-Z Index of the Bash command line. http://ss64.com/bash/. [Online; accessed 2016-07-29].

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan

Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Simple image classification with inception. `https://github.com/tensorflow/models/blob/master/tutorials/image/imagenet/classify_image.py`, 2016. [Online; accessed 2017-06-20].

[8] ABBYY. OCR Software from ABBYY. Best text recognition for Windows and Mac. `https://www.abbyy.com/finereader/`, 2016. [Online; accessed 2016-05-20].

[9] Constitution Act. Canadian charter of rights and freedoms. *Part II, section*, 35(2), 1982.

[10] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[11] Aipoly. Aipoly - vision through artificial intelligence. `http://aipoly.com/`. [Online; accessed 2017-01-18].

[12] Areej Alasiry, Mark Levene, and Alexandra Poulovassilis. Mining named entities from search engine query logs. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, IDEAS '14, pages 46–56, New York, NY, USA, 2014. ACM.

[13] J. E. Allen, C. I. Guinn, and E. Horvtz. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications*, 14(5):14–23, Sep 1999.

[14] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[15] Amazon.com, Inc. Alexa. `https://developer.amazon.com/public/solutions/alexa`. [Online; accessed 2016-07-27].

[16] American Foundation for the Blind. Screen readers - browse results - american foundation for the blind. `http://www.afb.org/prodBrowseCatResults.aspx?CatID=49`. [Online; accessed 2017-01-18].

[17] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Founda-*

tions of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on, pages 377–386. IEEE, 2010.

[18] Aneto Okonkwo. Now on Tap update: Text Select and Image Search. `https://blog.google/products/search/now-on-tap-update-text-select-and-image/?m=1`. [Online; accessed 2017-01-28].

[19] Apache Software Foundation. Apache2. `https://httpd.apache.org`.

[20] API.ai. Assistant by Api.ai - Assistant.ai. `https://assistant.ai/`. [Online; accessed 2016-07-27].

[21] Apple Inc. iOS - Siri - Apple (CA). `http://www.apple.com/ca/ios/siri/`. [Online; accessed 2016-07-27].

[22] Roland Bader. *Proactive Recommender Systems in Automotive Scenarios*. PhD thesis, Citeseer, 2013.

[23] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[24] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[25] Dana H Ballard. Modular learning in neural networks. In *AAAI*, pages 279–284, 1987.

[26] Youssef Bassil and Mohammad Alwani. Ocr post-processing error correction algorithm using google online spelling suggestion. *arXiv preprint arXiv:1204.0191*, 2012.

[27] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[28] Markus Beissinger. How to install Theano on Amazon EC2 GPU instances for deep learning. `http://markus.com/install-theano-on-aws/`, 2015. [Online; accessed 2016-07-28].

[29] Karissa Bell. Facebook messenger chief: It will be years before everyone has m. `http://mashable.com/2016/04/18/facebook-m-not-ready-for-years/#rnO6Nl3m1sqg`, 2016. [Online; accessed 2016-07-27].

[30] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[31] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 3–10, 2010.

[32] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. "O'Reilly Media, Inc.", 2009.

[33] Alessandro Bissacco, Mark Cummins, Yuval Netzer, and Hartmut Neven. PhotoOCR: Reading text in uncontrolled conditions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 785–792, 2013.

[34] S. Blankson. *Search Engine Optimization (SEO): How to Optimize Your Website for Internet Search Engines: Google, Yahoo!, Msn Live, Aol, Ask, Altavista, Fast, Gigablast, Snap, Looksmart and More*. Blankson Enterprises Limited, 2008.

[35] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[36] Eric Brill and Robert C Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics, 2000.

[37] Warick M Brown, Tamás D Gedeon, and David I Groves. Use of noise to augment training data: a neural network method of mineral–potential mapping in regions of limited known deposit examples. *Natural Resources Research*, 12(2):141–152, 2003.

[38] Horst Bunke and Terry Caelli. *Hidden Markov models: applications in computer vision*, volume 45. World Scientific, 2001.

[39] CanLII. CanLII - Hyperlinking Tool. `http://www.canlii.org/en/tools/hyperlink_tool.html`. [Online; accessed 2016-08-01].

[40] CanLII. CanLII - Tools. `http://www.canlii.org/en/tools/`. [Online; accessed 2016-08-01].

[41] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models And Methods In Applied Sciences*, 1(4):301–307, 2007.

[42] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer networks*, 31(11):1623–1640, 1999.

[43] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Metaquerier over the deep web: Shallow integration across holistic sources. In *In Proceedings of the VLDB Workshop on Information Integration on the Web*, 2004.

[44] Naehyuck Chang, Inseok Choi, and Hojun Shim. Dls: dynamic backlight luminance scaling of liquid crystal display. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(8):837–846, 2004.

[45] Barbara Chapman, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*, volume 10. MIT press, 2008.

[46] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[47] Y. N. Chen, Y. Huang, H. Y. Lee, and L. S. Lee. Unsupervised two-stage keyword extraction from spoken documents by topic coherence and support vector machine. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5041–5044, March 2012.

[48] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. ACM.

[49] Luis Pedro Coelho and Willi Richert. *Building machine learning systems with Python*. Packt Publishing Ltd, 2015.

[50] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[51] Kristin Cook, Nick Cramer, David Israel, Michael Wolverton, Joe Bruce, Russ Burtner, and Alex Endert. Mixed-initiative visual analytics using task-driven recommendations. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pages 9–16. IEEE, 2015.

[52] Greg Corrado. Computer, respond to this email. `https://research.googleblog.com/2015/11/computer-respond-to-this-email.html`, November 2015.

[53] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[54] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[55] EL Sabawi. Kali linux 1.0.3 brings accessibility features for blind users. `https://youtu.be/KMxqJInDdF0`. [Online; accessed 2017-01-18].

[56] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[57] Alex Endert, M Shahriar Hossain, Naren Ramakrishnan, Chris North, Patrick Fiaux, and Christopher Andrews. The human is the loop: new directions for visual analytics. *Journal of intelligent information systems*, 43(3):411–435, 2014.

[58] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[59] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

[60] Facebook. Facebook. `https://www.facebook.com/`. [Online; accessed 2016-08-03].

[61] Faisal Farooq. *Language models and automatic topic categorization for information retrieval in handwritten documents*. PhD thesis, State university of new york at buffalo, 5 2008.

[62] Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. *arXiv preprint arXiv:1605.02276*, 2016.

[63] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[64] A. Fujiyoshi, M. Suzuki, and S. Uchida. Syntactic detection and correction of misrecognitions in mathematical ocr. In *2009 10th International Conference on Document Analysis and Recognition*, pages 1360–1364, July 2009.

[65] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[66] U. Garain, A. Jain, A. Maity, and B. Chanda. Machine reading of camera-held low quality text images: An ica-based image enhancement approach for improving ocr accuracy. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, Dec 2008.

[67] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[68] Jonathan Gemmell, Thomas Schimoler, M Ramezani, L Christiansen, and B Mobasher. Resource recommendation for social tagging: a multi-channel hybrid approach. *Recommender Systems & the Social Web, Barcelona, Spain*, 2010.

[69] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), 2013.

[70] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning (adaptive computation and machine learning series). *Adaptive Computation and Machine Learning series*, page 800, 2016.

[71] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[72] Google. An in-depth look at Googles first Tensor Processing Unit (TPU). `https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu`. [Online; posted 2017-05-12; accessed 2017-06-13].

[73] Google. Google Developers - Google Now. `https://developers.google.com/schemas/now/cards`. [Online; accessed 2016-07-27].

[74] Google. Screen search - Google. `https://www.google.com/search/about/learn-more/screen-search/`. [Online; accessed 2017-01-19].

[75] Google Chrome Developers. Accessible components: Screen readers – polycasts 50. `https://youtu.be/Lktz1KXbTOU`. [Online; accessed 2017-01-18].

[76] Google Inc. Google Scholar. `https://scholar.google.ca/`. [Online; accessed 2016-08-02].

[77] Google Inc. Vision API - Image Content Analysis — Google Cloud Platform. `https://cloud.google.com/vision/`. [Online; accessed 2017-01-28].

[78] Google Inc. Google Chrome. `https://www.google.com/chrome/`, 2016.

[79] Luis Gravano, Panagiotis G. Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Lauri Pietarinen, and Divesh Srivastava. Using q-grams in a dbms for approximate string processing. *IEEE Data Eng. Bull.*, 24(4):28–34, 2001.

[80] Kristian A Gray, Bethan Yates, Ruth L Seal, Mathew W Wright, and Elspeth A Bruford. Genenames. org: the HGNC resources in 2015. *Nucleic acids research*, page gku1071, 2014.

[81] Roedy Green. Canadian Mind Products Java & Internet Glossary. http://mindprod.com/jgloss/jgloss.html. [Online; accessed 2016-07-29].

[82] Gregory Grefenstette and Lawrence Muchemi. Determining the characteristic vocabulary for a specialized dictionary using word2vec and a directed crawler. *CoRR*, abs/1605.09564, 2016.

[83] Shengbo Guo. *Bayesian recommender systems: Models and Algorithms*. Australian National University, 2011.

[84] Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. Semantic similarity from natural language and ontology analysis. *Synthesis Lectures on Human Language Technologies*, 8(1):1–254, 2015.

[85] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 1949.

[86] Jessi Hempel. Facebook launches m, its bold answer to siri and cortana. http://www.wired.com/2015/08/facebook-launches-m-new-kind-virtual-assistant/, 2015. [Online; accessed 2016-07-27].

[87] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[88] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995.

[89] Hironsan. Deep Learning Enables You to Hide Screen when Your Boss is Approaching. http://ahogrammer.com/2016/11/15/deep-learning-enables-you-to-hide-screen-when-your-boss-is-approaching/. [Online; accessed 2017-02-21].

[90] Ali Hussain Hitawala and Pradap Konda. py-stringmatching, 2016. [Online; accessed 2016-11-03].

[91] S. Hoffstaetter. pytesseract 0.1.6. https://pypi.python.org/pypi/pytesseract/, 2016. [Online; accessed 2016-05-20].

[92] Lawrence B Holder and Diane J Cook. Automated activity-aware prompting for activity initiation. *Gerontechnology: international journal on the fundamental aspects of technology to serve the ageing society*, 11(4):534, 2013.

[93] Eric Horvitz, Paul Koch, and Johnson Apacible. Busybody: creating and fielding personalized models of the cost of interruption. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 507–510. ACM, 2004.

[94] Hvass Laboratories. Hvass-Labs/TensorFlow-Tutorials. `https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/15_Style_Transfer.ipynb`, 2016. [Online; accessed 2017-06-13].

[95] Hwaci. Sqlite home page. `https://sqlite.org/`. [Online; accessed 2017-01-19].

[96] Heikki Hyyrö. Explaining and extending the bit-parallel approximate string matching algorithm of myers. Technical Report A-2001-10, Department of Computer and Information Sciences, University of Tampere, Tampere, Finland, 2001.

[97] IBM. IBM Watson Developer Community. `https://developer.ibm.com/watson/`. [Online; accessed 2016-07-27].

[98] IBM. Starter Kits — IBM Watson Developer Cloud. `https://www.ibm.com/watson/developercloud/starter-kits.html#conversational-agent`. [Online; accessed 2016-07-27].

[99] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[100] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):10, 2008.

[101] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. *Deep Features for Text Spotting*, pages 512–528. Springer International Publishing, Cham, 2014.

[102] Matthew A Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[103] Matthew A Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[104] Nicholas R Jennings and Michael Wooldridge. Applications of intelligent agents. In *Agent technology*, pages 3–28. Springer, 1998.

[105] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001–. [Online; accessed 2016-09-01].

[106] Youn-ah Kang and John Stasko. Characterizing the intelligence analysis process: Informing visual analytics design through a longitudinal field study. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 21–30. IEEE, 2011.

[107] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. i. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazn, and L. P. de las Heras. Icdar 2013 robust reading competition. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1484–1493, Aug 2013.

[108] Josh Karlin. Launchy. `https://www.launchy.net/`, 2016.

[109] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015. [Online; accessed 2017-06-13].

[110] P Kaushik, SS Intille, and K Larson. User-adaptive reminders for home-based medical tasks. *Methods Inf Med*, page 47, 2008.

[111] Anthony Kay. Tesseract: an open-source optical character recognition engine. *Linux Journal*, 2007:2–5, 2007.

[112] Ido Kissos and Nachum Dershowitz. Ocr error correction using character correction and feature-based word classification. *arXiv preprint arXiv:1604.06225*, 2016.

[113] Dag Kittlaus. The team behind Siri debuts its next-gen AI Viv at Disrupt NY 2016. `https://www.youtube.com/watch?v=MI07aeZqeco`, 2016. [Online; accessed 2016-07-27].

[114] Grzegorz Kondrak. N-gram similarity and distance. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval*, SPIRE'05, pages 115–126, Berlin, Heidelberg, 2005. Springer-Verlag.

[115] Benjamin Korvemaker and Russell Greiner. Predicting unix command lines: adjusting to user patterns. In *AAAI/IAAI*, pages 230–235, 2000.

[116] Viv Labs. Viv. `http://viv.ai/`. [Online; accessed 2016-07-27].

[117] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents.

[118] Yann Lecun. Ethics of artificial intelligence - general issues. `https://youtu.be/tNWqOgNDnCw?t=1h23m21s`. [Online; accessed 2017-01-17].

[119] Yann LeCun. EmTech MIT 2015: Augmented Knowledge Teaching Machines to Understand Us. `https://youtu.be/a3DzL7Ad4PU?t=17m10s`, 2015. [Online; accessed 2016-07-27].

[120] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[121] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[122] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

[123] Leon. simhash 1.7.0, 2016. [Online; accessed 2016-11-07].

[124] VI Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[125] Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. Pass-join: A partition-based method for similarity joins. *Proceedings of the VLDB Endowment*, 5(3):253–264, 2011.

[126] W. Li, X. Xie, J. Hu, Z. Zhang, and Y. Zhang. Using big data from the web to train chinese traffic word representation model in vector space. In *2016 12th World Congress on Intelligent Control and Automation (WCICA)*, pages 2304–2307, June 2016.

[127] LISA lab. Multi cores support in Theano. `http://deeplearning.net/software/theano/tutorial/multi_cores.html`. [Online; accessed 2016-07-28].

[128] Matthew Lynlet. Google unveils google assistant, a virtual assistant thats a big upgrade to google now. `https://techcrunch.com/2016/05/18/google-unveils-google-assistant-a-big-upgrade-to-google-now/`, 2016. [Online; accessed 2016-07-27].

[129] Laurens Maaten. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, pages 384–391, 2009.

[130] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[131] Omid Madani, Hung Hai Bui, and Eric Yeh. Efficient online learning and prediction of users' desktop actions. In *IJCAI*, pages 1457–1462, 2009.

[132] Stephen Makonin, Daniel McVeigh, Wolfgang Stuerzlinger, Khoa Tran, and Fred Popowich. Mixed-initiative for big data: The intersection of human+ visual analytics+ prediction. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 1427–1436. IEEE, 2016.

[133] Vincent Quirante Malic. Analyzing historians' perspectives using semantic measures. *IConference 2016 Proceedings*, 2016.

[134] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[135] Self-Organizing Feature Maps. Self-organization and associative memory. *Self*, 8, 1988.

[136] Luis Martinez, Luis G Perez, and Manuel J Barranco. Incomplete preference relations to smooth out the cold-start in collaborative recommender systems. In *Fuzzy Information Processing Society, 2009. NAFIPS 2009. Annual Meeting of the North American*, pages 1–6. IEEE, 2009.

[137] Randall Matignon. *Neural network modeling using SAS enterprise miner*. AuthorHouse, 2005.

[138] Jonas McCallum. autocorrect: 0.2.0. `https://pypi.python.org/pypi/autocorrect/0.2.0`, 2014–2016.

[139] Sean M McNee, John Riedl, and Joseph A Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM, 2006.

[140] Gabor Melli. Concept mentions within kdd-2009 abstracts (kdd09cma1) linked to a kdd ontology (kddo1). In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[141] Microsoft. Bing images. `http://www.bing.com/images`. [Online; accessed 2016-08-03].

[142] Microsoft. Cortana - Meet your personal assistant - Microsoft - Global. `https://www.microsoft.com/en/mobile/experiences/cortana/#`. [Online; accessed 2016-07-27].

[143] Microsoft. Error Message Guidelines (Windows). `https://msdn.microsoft.com/en-us/library/windows/desktop/ms679325.aspx`, 2016. [Online; accessed 2016-05-20].

[144] Microsoft Corp. Microsoft Bot Framework. `https://dev.botframework.com/`. [Online; accessed 2017-01-18].

[145] Microsoft Corporation. Youre the expert: Teach cortana when to connect you to users - building apps for windowsbuilding apps for windows. `https://blogs.windows.com/buildingapps/2016/04/07/youre-the-expert-teach-cortana-when-to-connect-you-to-users/#X0iFzqIBPlSh0uAL.97`. [Online; accessed 2017-01-19].

[146] Dave Mielke. BRRLTY. `http://mielke.cc/brltty/`. [Online; accessed 2017-01-18].

[147] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[148] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, volume 13, pages 746–751, 2013.

[149] Bonan Min, Shuming Shi, Ralph Grishman, and Chin-Yew Lin. Towards large-scale unsupervised relation extraction from the web. *Int. J. Semant. Web Inf. Syst.*, 8(3):1–23, July 2012.

[150] José Antonio Miñarro-Giménez, Oscar Marín-Alonso, and Matthias Samwald. Applying deep learning techniques on medical corpora from the world wide web: a prototypical system and evaluation. *CoRR*, abs/1502.03682, 2015.

[151] Dharitri Misra, Siyuan Chen, and George R Thoma. A system for automated extraction of metadata from scanned documents using layout recognition and string pattern search models. In *Archiving.... IS & T's Archiving Conference*, volume 1509, page 107. NIH Public Access, 2009.

[152] Koji Miyahara and Michael J Pazzani. Improvement of collaborative filtering with the simple bayesian classifier 1. 2002.

[153] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[154] Mike Mol. Rosetta Code. `http://rosettacode.org/`. [Online; accessed 2016-07-29].

[155] Alvaro E. Monge and Charles P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 267–270. AAAI Press, 1996.

[156] Tom Morris, Amit Dovev, Stefan Weil, and zdenop. ImproveQuality tesseract-ocr/tesseract Wiki GitHub. `https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality`, 2016. [Online; accessed 2016-06-07].

[157] MPC-HC Team. Home MPC-HC. `https://mpc-hc.org/`. [Online; accessed 2017-01-26].

[158] Jörg P Müller. The right agent (architecture) to do the right thing. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 211–225. Springer, 1998.

[159] Randall Munroe. xkcd: Game ais. `https://xkcd.com/1002/`. [Online; accessed 2017-01-18].

[160] Emerson Murphy-Hill and Gail C Murphy. Recommendation delivery. In *Recommendation Systems in Software Engineering*, pages 223–242. Springer, 2014.

[161] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[162] Felix Naumann and Melanie Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2(1):1–87, 2010.

[163] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[164] David Necas, Esa Mtt, Mikko Ohtamaa, et al. python-Levenshtein: The levenshtein python c extension module contains functions for fast computation of levenshtein distance and string similarity. `https://github.com/ztane/python-Levenshtein`, 2010–. [Online; accessed 2016-09-01].

[165] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[166] NextOS. NextOS - Denise - Creating Virtual Life. `http://www.nextos.com/`. [Online; accessed 2016-07-27].

[167] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[168] Nvidia. NVIDIA Volta AI Architecture. `https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/`. [Online; accessed 2017-06-13].

[169] NVIDIA Corporation. CUDA FAQ — NVIDIA Developer. `https://developer.nvidia.com/cuda-faq`. [Online; accessed 2016-07-29].

[170] Hyacinth S Nwana. Software agents: An overview. *The knowledge engineering review*, 11(03):205–244, 1996.

[171] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[172] Emilio Soria Olivas, José David Martín Guerrero, Marcelino Martinez Sober, Jose Rafael Magdalena Benedito, and Antonio José Serrano López. Handbook of research on machine learning applications and trends: Algorithms, methods, and. 2010.

[173] Jason W Osborne. *Best practices in quantitative methods*. Sage, 2008.

[174] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 938–947, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[175] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 1999.

[176] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[177] X. Peng, H. Cao, and P. Natarajan. Document image ocr accuracy prediction via latent dirichlet allocation. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 771–775, Aug 2015.

[178] Pete Warden. Tensorflow for poets. `https://petewarden.com/2016/02/28/tensorflow-for-poets/`. [Online; accessed 2017-01-17].

[179] Peter Evers. images-scraper . `https://github.com/pevers/images-scraper`, 2016. [Online; accessed 2016-07-07].

[180] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[181] Pearl Pu and Denis Lalanne. Design visual thinking tools for mixed initiative systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 119–126. ACM, 2002.

[182] R Quian Quiroga, Leila Reddy, Gabriel Kreiman, Christof Koch, and Itzhak Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005.

[183] Alec Radford. Introduction to deep learning with python. `http://www.slideshare.net/indicods/deep-learning-with-python-and-the-theano-library`, 2014.

[184] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[185] Google Brain Rafal Jozefowicz. Tensorflow introduction, 2016. Presented 2016-02-20. [Online; accessed 2017-01-15].

[186] Andreas Rauber, Dieter Merkl, and Michael Dittenbach. The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, 2002.

[187] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[188] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender Systems Handbook*. Springer, 2015.

[189] Garry J Roedler and Cheryl Jones. Technical measurement. 2005.

[190] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[191] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[192] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[193] Nick Satt. Hands-on with facebook m: the virtual assistant with a (real) human touch. `http://www.theverge.com/2015/10/26/9605526/facebook-m-hands-on-personal-assistant-ai`, 2015. [Online; accessed 2016-07-27].

[194] Nicholas Sawadsky and Gail C Murphy. Fishtail: from task context to source code examples. In *Proceedings of the 1st Workshop on Developing Tools as Plug-ins*, pages 48–51. ACM, 2011.

[195] Nicholas Sawadsky, Gail C Murphy, and Rahul Jiresal. Reverb: Recommending code-related web pages. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 812–821. IEEE Press, 2013.

[196] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. *Collaborative Filtering Recommender Systems*, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[197] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[198] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[199] Hinrich Schütze. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, 2008.

[200] Barry Schwartz. *The paradox of choice : why more is less.* Ecco, New York, 2004.

[201] scikit learn. Receiver operating characteristic (roc) scikit-learn 0.18.1 documentation.

[202] scikit-learn developers. sklearn.manifold.TSNE - scikit-learn 0.18.1 documentation. `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`. [Online; accessed 2017-01-19].

[203] F. Seide, G. Li, X. Chen, and D. Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *2011 IEEE Workshop on Automatic Speech Recognition Understanding*, pages 24–29, Dec 2011.

[204] Daniel Shapiro, Nathalie Japkowicz, Mathieu Lemay, and Miodrag Bolic. Fuzzy string matching with character-level deep neural network classification. *Applied Artificial Intelligence*, submitted 2016.

[205] Daniel Shapiro, Hamza Qassoud, and Miodrag Bolic. What a neural network sees in a computer screen. In *GRADUATE RESEARCH POSTER COMPETITION 2017*, Ottawa, Ontario, Mar. 30 2017.

[206] Daniel Shapiro, Hamza Qassoud, and Miodrag Bolic. Filtering noisy text to accelerate classification. *Neurocomputing*, submitted 2017.

[207] Daniel Shapiro, Hamza Qassoud, Mathieu Lemay, and Miodrag Bolic. Neural network text filtering. `https://github.com/hamzaqassoud/Neural_Network_Filtering`, 2016. [Online; accessed 2016-11-03].

[208] Daniel Shapiro, Hamza Qassoud, Mathieu Lemay, and Miodrag Bolic. Unsupervised deep learning recommender system for personal computer users. In *Proceedings of the Sixth International Conference on Intelligent Systems and Applications (INTELLI 2017)*. IARIA, July 2017 (to appear).

[209] Daniel Shapiro, Hamza Qassoud, Mathieu Lemay, and Miodrag Bolic. Visual deep learning recommender system for personal computer users. In *Proceedings of the Second International Conference on Applications and Systems of Visual Paradigms (VISUAL 2017)*. IARIA, July 2017 (to appear).

[210] Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *AISTATS*, pages 886–894, 2014.

[211] David A Sierra, Debra J Gilbert, Deborah Householder, Nick V Grishin, Kan Yu, Pallavi Ukidwe, Sheryll A Barker, Wei He, Theodore G Wensel, Glen Otero, et al. Evolution of the regulators of g-protein signaling multigene family in mouse and human. *Genomics*, 79(2):177–185, 2002.

[212] Jocelyn Sietsma and Robert JF Dow. Creating artificial neural networks that generalize. *Neural networks*, 4(1):67–79, 1991.

[213] A. Silberpfennig, L. Wolf, N. Dershowitz, S. Bhagesh, and B. B. Chaudhuri. Improving ocr for an under-resourced script using unsupervised word-spotting. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 706–710, Aug 2015.

[214] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[215] Adam Smith. Kite - Programming copilot. `https://kite.com/`, 2016. [Online; accessed 2016-05-17].

[216] R. Smith. An overview of the tesseract ocr engine. *2013 12th International Conference on Document Analysis and Recognition*, 2:629–633, 2007.

[217] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[218] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[219] Charlie Sorrel. Word Lens: Augmented Reality App Translates Street Signs Instantly. `https://www.wired.com/2010/12/word-lens-augmented-reality-app-translates-street-signs-instantly/`. [Online; accessed 2017-06-26].

[220] SoundHound Inc. SoundHound - Hound App. `http://www.soundhound.com/hound`. [Online; accessed 2016-07-27].

[221] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[222] Stack Exchange Inc. class - Print the Python Exception/Error Hierarchy - Stack Overflow. `http://stackoverflow.com/questions/18296653/print-the-python-exception-error-hierarchy`. [Online; accessed 2016-07-29].

[223] Stack Exchange Inc. Stack Overflow. `stackoverflow.com`. [Online; accessed 2016-07-29].

[224] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931. ACM, 2016.

[225] Z. Su, B. R. Ahn, K. Y. Eom, M. K. Kang, J. P. Kim, and M. K. Kim. Plagiarism detection using the levenshtein distance and smith-waterman algorithm. In *Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on*, pages 569–569, June 2008.

[226] Supreme Court of Canada. Canadian Broadcasting Corp. v. Canada (Attorney General). 1 SCR 19, 2011 SCC 2, 2011.

[227] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[228] Kirsten Swearingen and Rashmi Sinha. Beyond algorithms: An hci perspective on recommender systems. In *ACM SIGIR 2001 Workshop on Recommender Systems*, volume 13, pages 1–11. Citeseer, 2001.

[229] Al Sweigart. Welcome to PyAutoGUIs documentation! PyAutoGUI 1.0.0 documentation. `https://pyautogui.readthedocs.io/en/latest/`. [Online; accessed 2017-02-08].

[230] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[231] Hiroyuki Tanaka. editdistance: 0.3.1. `https://pypi.python.org/pypi/editdistance`, 2016.

[232] TensorFlow. Metric ops — tensorflow, 2017. [Online; accessed 2017-01-15].

[233] The Werkzeug Team. werkzeug. `http://werkzeug.pocoo.org/`.

[234] P. Thompson, J. McNaught, and S. Ananiadou. Customised ocr correction for historical medical text. In *2015 Digital Heritage*, volume 1, pages 35–42, Sept 2015.

[235] TinEye. TinEye Reverse Image Search. `https://www.tineye.com/`. [Online; accessed 2017-02-06].

[236] tmux-users. tmux. `https://tmux.github.io/`.

[237] Tomas Mikolov et. al., Google. Word2vec. `https://code.google.com/archive/p/word2vec/`.

[238] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[239] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[240] James Turk, Michael Stephens, et al. jellyfish: a library for doing approximate and phonetic matching of strings. `https://github.com/jamesturk/jellyfish`, 2015–. [Online; accessed 2016-09-01].

[241] Peter D Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *European Conference on Machine Learning*, pages 491–502. Springer, 2001.

[242] Esko Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1-3):100–118, 1985.

[243] Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, and Erik Duval. Context-aware recommender systems for learning: a survey and future challenges. *IEEE Transactions on Learning Technologies*, 5(4):318–335, 2012.

[244] Alessandro Vinciarelli. Noisy text categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1882–1895, 2005.

[245] Kai Wang and Serge Belongie. Word spotting in the wild. In *European Conference on Computer Vision*, pages 591–604. Springer, 2010.

[246] Richard C Wang and William W Cohen. Language-independent set expansion of named entities using the web. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 342–350. IEEE, 2007.

[247] Richard C Wang, Nico Schlaefer, William W Cohen, and Eric Nyberg. Automatic set expansion for list question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 947–954. Association for Computational Linguistics, 2008.

[248] Tong Wang, Vish Viswanath, and Ping Chen. Extended topic model for word dependency. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference of the Asian Federation of Natural Language Processing*, page 506, 2015.

[249] Cristina L Ward, Satoshi Omura, and Ron R Kopito. Degradation of CFTR by the ubiquitin-proteasome pathway. *Cell*, 83(1):121–127, 1995.

[250] Wayfindr. Wayfindr - empowering vision impaired people. `https://www.wayfindr.net/`. [Online; accessed 2017-01-18].

[251] L. T. Weng, Y. Xu, Y. Li, and R. Nayak. Exploiting item taxonomy for solving cold-start problem in recommendation making. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 113–120, Nov 2008.

[252] Casey Whitelaw, Ben Hutchinson, Grace Y Chung, and Gerard Ellis. Using the web for language independent spellchecking and autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, volume 2, pages 890–899. Association for Computational Linguistics, 2009.

[253] Mark Whiting, Kristin Cook, Georges Grinstein, Kristen Liggett, Michael Cooper, John Fallon, and Marc Morin. Vast challenge 2014: The kronos incident. In *Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on*, pages 295–300. IEEE, 2014.

[254] M. Wick, M. Ross, and E. Learned-Miller. Context-sensitive error correction: Using topic models to improve ocr. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1168–1172, Sept 2007.

[255] Christopher D Wickens, Huiyang Li, Amy Santamaria, Angelia Sebok, and Nadine B Sarter. Stages and levels of automation: An integrated meta-analysis. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 54, pages 389–393. SAGE Publications, 2010.

[256] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[257] Wikimedia Foundation. Deep learning - Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Deep_learning`, 2016. [Online; accessed 2016-05-20].

[258] William E Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. Technical report, Educational Resources Information Center (ERIC), 03 1990.

[259] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[260] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[261] Yandex. Yandex.Images: search for images on the internet, search by image. `https://yandex.com/images/`. [Online; accessed 2017-02-06].

[262] Q. Ye and D. Doermann. Text detection and recognition in imagery: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1480–1500, July 2015.

[263] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[264] Giancarlo Zaccone. *Getting Started with TensorFlow*. Packt Publishing Ltd, 2016.

[265] Pengyi Zhang and Dagobert Soergel. Towards a comprehensive model of the cognitive process and mechanisms of individual sensemaking. *Journal of the Association for Information Science and Technology*, 65(9):1733–1756, 2014.

[266] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 649–657. Curran Associates, Inc., 2015.

[267] G. Zorpette. Can we copy the brain? *IEEE Spectrum*, 54(6):8–8, June 2017.