UNIVERSITY OF OTTAWA

MASTER'S THESIS

Head and Shoulder Detection using CNN and RGBD Data

Author:
Wassim EL AHMAR

Supervisor:
Dr. Robert LAGANIÈRE

A thesis submitted in fulfillment of the requirements
for the degree of Master of Applied Science

in the

VIVA Research Lab
School of Electrical Engineering and Computer Science

Declaration of Authorship

I, Wassim EL AHMAR, declare that this thesis titled, "Head and Shoulder Detection using CNN and RGBD Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

UNIVERSITY OF OTTAWA

Abstract

Engineering

School of Electrical Engineering and Computer Science

Master of Applied Science

Head and Shoulder Detection using CNN and RGBD Data

by Wassim EL AHMAR

Alex Krizhevsky and his colleagues changed the world of machine vision and image processing in 2012 when their deep learning model, named Alexnet [1], won the ImageNet Large Scale Visual Recognition Challenge with more than 10.8% lower error rate than their closest competitor. Ever since, deep learning approaches have been an area of extensive research for the tasks of object detection, classification, pose estimation, etc...This thesis presents a comprehensive analysis of different deep learning models and architectures that have delivered state of the art performances in various machine vision tasks. These models are compared to each other and their strengths and weaknesses are highlighted.

We introduce a new approach for human head and shoulder detection from RGB-D data based on a combination of image processing and deep learning approaches. Candidate head-top locations(CHL) are generated from a fast and accurate image processing algorithm that operates on depth data. We propose enhancements to the CHL algorithm making it three times faster. Different deep learning models are then evaluated for the tasks of classification and detection on the candidate head-top locations to regress the head bounding boxes and detect shoulder keypoints. We propose 3 different small models based on convolutional neural networks for this problem. Experimental results for different architectures of our model are highlighted. We also compare the performance of our model to mobilenet [2].

Finally, we show the differences between using 3 types of inputs CNN models: RGB images, a 3-channel representation generated from depth data (Depth map, Multi-order depth template, and Height difference map or DMH), and a 4 channel input composed of RGB+D data.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Robert Laganière for trusting my abilities and believing in my potential. The time I have spent under his supervision as a member of the VIVA Research lab has been extremely beneficial for me. I have learned a lot from him and gained vital research and industrial experience under his supervision.

I also thank my family, my parents and siblings, who have been supportive of me for the past couple of years, both financially and emotionally.

Finally, I owe my thanks to all the members of the VIVA Research lab for their help and assistance whenever needed. Namely, I express my gratitude to Farzan Erlik Nowruzi for everything I have learned from him

Contents

List of Figures

List of Tables

List of Abbreviations

ACF	**A**ggregated **C**hannel, **F**eatures
CHL	**C**andidate **H**eadtop **L**ocations
CNN	**C**onvolutional **N**eural **N**etworks
CPU	**C**entral **P**rocessing **U**nit
DMH	**D**epth map, **M**ulti-order depth template, and **H**eight difference map
FC	**F**ully **C**onnected
FPS	**F**rames **P**er **S**econd
GPU	**G**raphical **P**rocessing **U**nit
HOD	**H**istogram of **O**riented **D**epths
HOG	**H**istogram of **O**riented **G**radients
HSV	**H**ue, **S**aturation, **L**ightness
ILSVRC	**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
IOU	**I**ntersection **O**ver **U**nion
MOD	**M**ulti **O**rder **D**epth template
MSE	**M**ean **S**quarred **E**rror
NMS	**N**on **M**aximum **S**uppression
PCK	**P**robability of **C**orrect **K**eypoints
ReLU	**Re**ctified **L**inear **U**nit
RGB	**R**ed **G**reen **B**lue
RGBD	**R**ed **G**reen **B**lue **D**epth
RPN	**R**egion **P**roposals **N**etwork
SSD	**S**ingle **S**hot multibox **D**etector

Chapter 1

Introduction

Since the dawn of history, humans have been developing new ways and means to make their lives easier and more comfortable. Technology today has reached astonishingly new heights. Many ideas that we couldn't even imagine a few decades ago are today a reality and part of our everyday lives. We live today in the so called *information era* as a result of the digital revolution.

There has been several milestones that defined the information age. The birth of the internet in 1969, the first personal computer released in 1975, and the release of graphical user interface in 1984 are among the most important milestones. Today, artificial intelligence and deep learning are making their way more and more to our lives. With many useful applications including self driving cars, intelligent robots, virtual assistants, etc, deep learning has become a very active field of interest for researchers.

Deep learning is based on the concept of artificial neural network which is a programming paradigm that aims to allow a system to learn by example. It is inspired by the way our brains process information. Although the first neural network was mentioned in the late 1950s [3], it wasn't until 2012 that deep learning made the breakthrough with Alexnet [1] with state of the art performance in the ILSVRC challenge [4]. Most research conducted nowadays in the field of machine vision is based on deep learning approaches. Deep learning allowed state of the art performance on different machine vision tasks like image classification, segmentation, object detection, pose estimation...

In this thesis, we explain the work we have done for the task of human head and shoulder detection in indoor environments using deep learning approaches. We elaborate on the different models we have developed and different experiments we performed. We compare our work to other popular deep learning approaches in the literature and provide a comprehensive analysis of the results.

The remainder of this chapter is organized as follows. We briefly introduce depth sensors in Section 1.1. We provide an overview of CNN, image classification, object detection, and keypoint detection in Sections 1.2 1.3, 1.4, and 1.5 respectively. Current challenges to object and keypoint detection are given in Section 1.6. The problem statement is given in Section 1.7. Finally, we show the structure of this thesis in Section 1.9.

1.1 Depth Data

A depth sensor is a device that measures the distance between the sensor and pixel data in a captured frame. Depth data is usually stored in a 16 bit single channel image. Depth sensors are usually used along with a regular RGB camera to produce synchronized color and depth images. The first depth sensor to be widely used was the Microsoft Kinect [1], released as a peripheral for Xbox 360 [2]. Ever since, several models of depth sensors have been released. The applications of depth sensors include:

FIGURE 1.1: Microsoft Kinect depth sensor

- Gesture recognition for gaming
- 3D modelling
- Scene understanding and reconstruction

[1] https://developer.microsoft.com/en-us/windows/kinect
[2] https://www.xbox.com/en-CA

- Collision avoidance in autonomous vehicles

1.2 Convolutional Neural Networks

Convolutional neural networks are artificial neural networks. They are inspired by the way the human brain works. With CNN, a classification/detection system *learns* the features that are relevant for its operation on its own and dynamically assigns weights, or importance, to different features. This allows CNN to be versatile and applicable to different machine vision problems. Traditional neural networks receive a vector as an input which is then processed by a series of fully connected layers of neurons. This makes traditional neural networks impractical for image analysis as the computational cost would be too heavy. CNN solve this problem through the use of convolution operations applied on filters having predefined *width, height* and *depth* which makes them suitable for receiving images as input and extracting features from them. A feature extractor is a CNN that receives an image as an input, performs certain operations on it through a series of hidden layers, and outputs feature maps at the last layer that contain features relevant for the image analysis problem.

1.3 Image Classification

Image classification is the process of assigning a label (class) to a certain input image. Traditional methods for classification required the manual crafting of the features of a certain class (descriptors) and the training of a classifier (Support vector machines [5], boosting [6], random forests [7]...). However, such approaches were still lacking in terms of accuracy and processing speed. With the advancement in technology and the ability to harness graphic card power for processing, classification using CNN is able to achieve state of the art performance in speed and accuracy. A CNN is trained using a labelled set of images of different classes (people,cars,trees...). The output of a classification CNN is a vector of probabilities corresponding to the classes. Each probability denotes the likeliness of this image belonging to the corresponding class.

Figure 1.2 shows the general workflow of a CNN image classifier. An image is processed through the CNN hidden layers and class probabilities are output by the model. Different types of CNN layers are explained in Section 2.2 of Chapter 2.

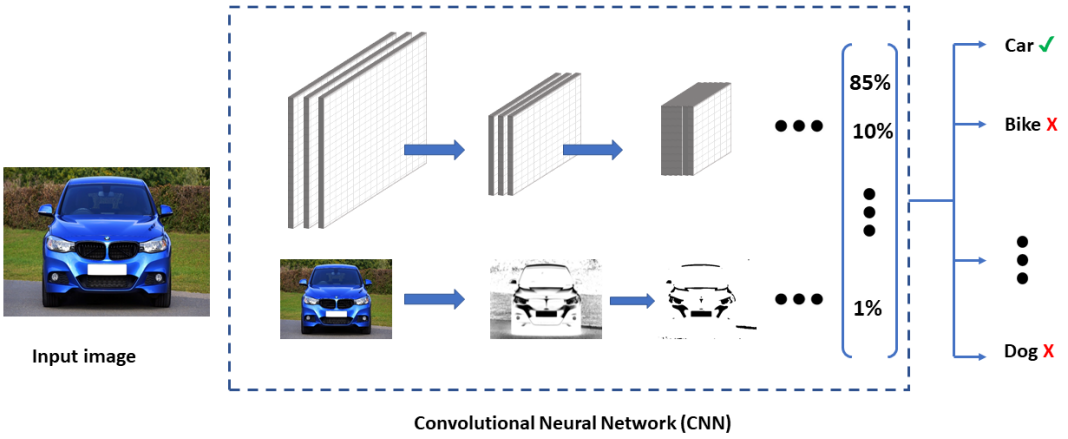

FIGURE 1.2: An image is processed through a CNN to generate class probabilities.

1.4 Object Detection

Image classification raises a challenge when multiple objects of different classes are present in one image. For example, an image from an outdoor scene might include cars, pedestrians, trees, traffic lights... Since image classification only assigns one label to an image, we are not able to know all the classes that are present in this image or their locations. With object detection algorithms, the goal is to localize the exact location of an object of a certain class in an image. CNN also achieve state of the art performance in the task of object detection. Instead of outputting a vector of probabilities, an object detection CNN outputs a set of bounding boxes with corresponding confidence scores and class labels. Object detection is an important task in applications such as people counting and self driving cars. In the latter, a self driving car needs to reliably be able to detect different objects in its surroundings to make correct decisions. For example, it needs to detect cars and pedestrians for collision avoidance, road lanes to stay on track, and traffic lights and road signs to maintain the safety of passengers. Figure 1.3 shows objects of different classes detected using

a CNN.

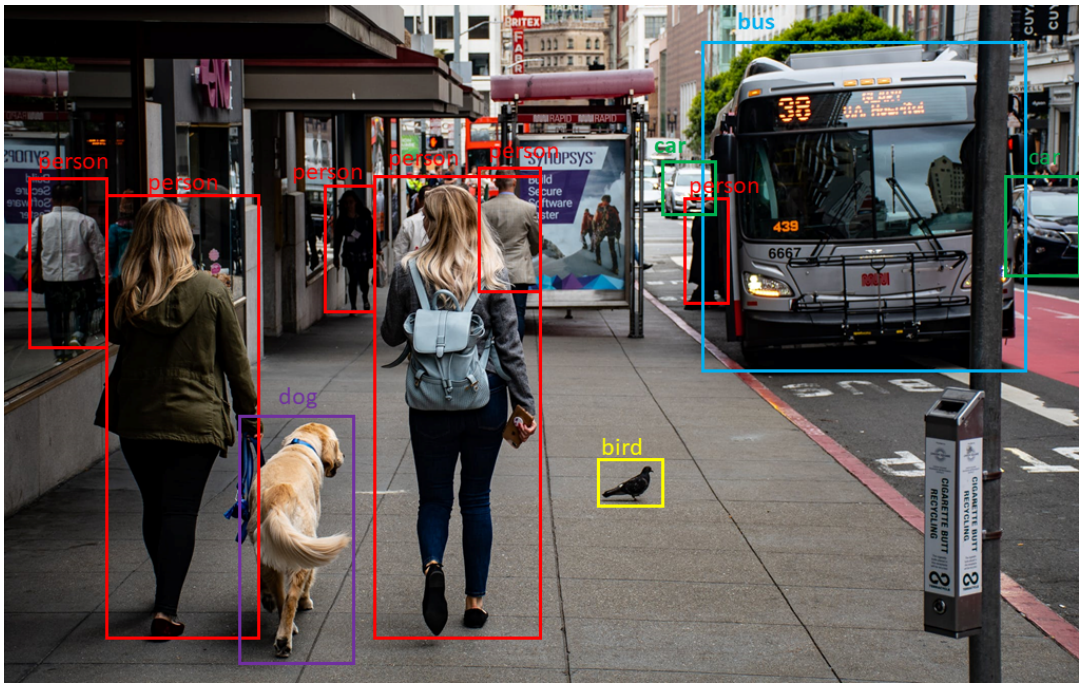

FIGURE 1.3: Results of object detection run on an image with different classes.

1.5 Keypoint Detection

Keypoint detection is the process of detecting points of interest on objects in an image. For example, detecting the facial landmark positions of a person would assist in predicting the mood of that person. It can also be used in driver assistance systems to detect when a driver falls asleep during driving. Detecting the shoulder keypoint locations of a person is a key step in the process of pose estimation. In turn, this allows us to recognize the action a person is taking and predict their future movements. CNN can also be used to predict shoulder keypoints. A CNN would be trained on keypoint-annotated images. During prediction, the CNN outputs the coordinates of the keypoints. CNN also achieve state of the art performance in keypoint detection. For example, OpenPose [8] is a CNN architecture for detecting the keypoints of a human skeleton. It is considered the state of the art in pose estimation methods with a shoulder detection precision of 91.3% on the MPII dataset [9]. Figure

1.4 shows the results of human pose estimation using OpenPose [8].

FIGURE 1.4: Human pose estimation using OpenPose

1.6 Challenges

Even with the great advancements achieved in the field of machine vision and object detection, certain challenges to object and keypoint detection still exist. The main two challenges are:

- Occlusion: One of the main challenges to object detection is occlusion. It is difficult for a model to detect objects that are fully or partially occluded. This is specially true for congested scenes like a crowded street.
- Feature broadness: This is a challenge to both traditional and deep learning approaches. For traditional approaches, it is difficult to hand craft a descriptor that comprehensively describes all possible states of an object. For example, in

pedestrian detection, various styles in clothing and appearance, and the different possible articulations of human poses make it hard to define one descriptor that efficiently detects pedestrians. Feature broadness also affects deep learning approaches. With deep learning, the performance of a model is reliant on the size of the training data. The more examples a model trains on, the better it is expected to be in predicting test images. However, generating training data requires manual labour for data annotations.

1.7 Problem Statement

Reliable detection of human heads and shoulders is still a challenging task. Although deep learning allowed for state of the art performance in detection, this usually comes at a high computational cost. Since head detection can be used for important people counting and congestion analysis applications, it is important to have a system capable of reliably detecting human heads and shoulders. In addition, if the system is to be embedded into a smart camera, it needs to be efficient in terms of power consumption, size, and computational cost.

1.8 Contribution

In this thesis, we developed a reliable system for detecting human heads and shoulders in indoor environments using image processing techniques and deep learning. We optimize our approach taking into consideration the model size and computational cost.

We compare our approach to other state of the art solutions in the tasks of object and shoulder keypoint detection. The different approaches are compared in terms of accuracy, quality of detections, and computational cost. We also show the effect that adding depth information to the input has on both head and shoulder detection. We

show that our approach achieves comparable performance to state of the art methods while being more efficient to run on embedded systems.

1.9 Thesis structure

This thesis is composed of 5 Chapters and the overview of each chapter is given below:

- **Chapter 1** "Introduction:" This chapter gives some background information over certain important concepts and technologies that are relevant in the scope of this thesis. Namely, we briefly introduce CNN and three of their important applications (classification, object detection, and shoulder detection) and identify the important challenges.
- **Chapter 2** "Convolutional Neural Networks:" We provide an explanation of the different building blocks of CNN and provide an overview of famous CNN feature extractors and classifiers.
- **Chapter 3** "Related Work:" We provide a literature review of methods utilizing RGBD data for human detection, in addition to famous CNN and deep learning approaches used in the tasks of object and keypoint detection.
- **Chapter 4** "Methodology:" In this chapter, we explain the details of our implementation and the different variations of our approach. We detail the configuration and parameters used in our methods and discuss how we compare to previously existing solutions.
- **Chapter 5** "Experimentaion and Results:" This chapter shows the results of the different experiments we performed. We also provide an analysis of the results and highlight the advantages and disadvantages of different approaches.
- **Chapter 6** "Conclusion:" In the last chapter, we conclude our thesis with a summary of the experimentations performed and the results achieved. We highlight the future work that can be performed to further optimize the task of human head and choulder detection.

Chapter 2

Convolutional Neural Networks

Our work includes the utilization of depth information of a frame to generate proposals of possible human head locations in an image. Those proposals then need to be classified as heads, and non heads. For proposals classified as heads, we need to regress the head bounding box, in addition to detecting the shoulder keypoint locations. Our approaches relies on CNN to perform these tasks. Hence, in this chapter, we will introduce CNN and famous CNN feature extractors.

The structure of this chapter is as follows: Section 2.1 gives an overview of how traditional systems for object detection work. Section 2.2 explains the concept of convolutional neural networks and feature extraction. Different CNN feature extractors and classifiers are described in section 2.3.

2.1 Traditional detection methods

Traditional methods for object detection [10],[11],[12] required the system designer to handcraft the features the detection model should look for (descriptor), and then apply a method like sliding windows to look for those features in an input image. At every location of the sliding window, the features are calculated and fed to a classification model like Support Vector Machines (SVM)[5], adaboost[6], random forests[7].

The two main problems that plagued traditional methods were the difficulty in generalizing a model, and the processing time. Because features had to be specifically designed by the system developer, it was difficult to develop a system that

would work for two different detection problems. That is because features that are relevant for detecting a pedestrian, for example, are different than the features relevant for detecting a car. The second major problem with traditional methods is the slow processing time. Features need to be calculated at every location of every sliding window to cover the entire area of the image. In addition, to solve the problem of detecting objects at different scales, the images had to be resized at different scales(image pyramid) and the sliding window approach applied to every scale to detect objects of different sizes [11]. The image pyramids method is illustrated in figure 2.1.

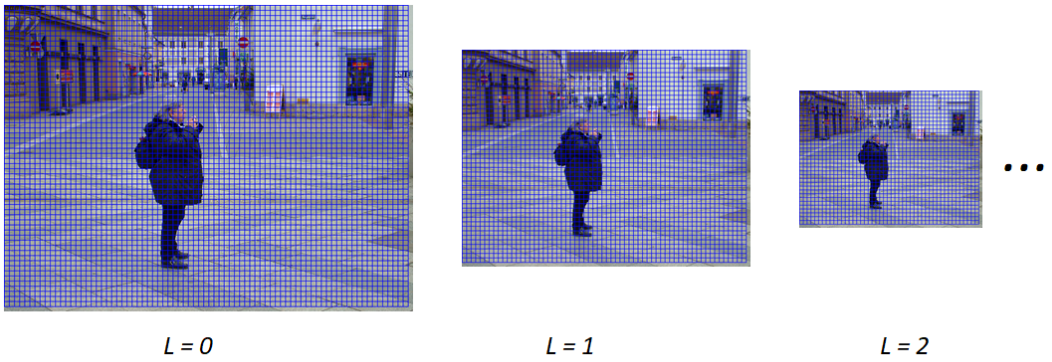

FIGURE 2.1: Illustration of the concept of image pyramids. Sliding windows (in blue) are applied to each level of the pyramid

Consider a pedestrian detection system using the Aggregated Channel Features (ACF) [11]. The ACF approach builds a descriptor from the aggregation of 10 channels. Histogram of oriented gradients (6 channels), Luminance and Chrominance (LUV) representation (3 channels), and 1 channel representing the gradient magnitude. A sliding window approach is applied on each level of an image pyramid to detect objects at different scales. At each location of the sliding window, the ACF features are calculated and Adaboost[6] is used for classification. Assume that the input images are of size 640×480, and the sliding window used is of size 64×128. A stride (how much the sliding window is shifting horizontally and vertically) of 4 is used for sliding windows, and the image pyramid that we designed consists of 4 levels and the image is scaled down by 20% at every level. The number of windows that need to be evaluated by the system are given in table 2.1

Pyramid Level	Image Size	Total number of windows
1	640 x 480	12672
2	512 x 384	7168
3	384 x 288	3200
4	256 x 192	768
Total		23808

TABLE 2.1: Number of windows to be evaluated at different levels of image pyramid

We can see that for this example, where the image is only resized 3 times and the sliding window stride is set to 4, there are 23808 locations where features in the image need to be calculated and evaluated using a classifier. This makes such systems inefficient for real time applications.

2.2 Convolutional Neural Networks

Convolutional neural networks are artificial neural networks. With CNN, a classification/detection system *learns* the features that are relevant for its operation on its own and dynamically assigns weights, or importance, to different features. This allows CNN to be versatile and applicable to different machine vision problems. Traditional neural networks receive a vector as an input which is then processed by a series of fully connected layers of neurons. This makes traditional neural networks impractical for image analysis as the computational cost would be too heavy. CNN solve this problem as their structure of *width, height* and *depth* makes them suitable for receiving images as input and extracting features from them. A feature extractor is a CNN that receives an image as an input, performs certain operations on it through a series of hidden layers, and outputs feature maps at the last layer that contain features relevant for the image analysis problem. A feature extractor is usually composed of several layers, the most important ones being: **Convolutional layers, Pooling layers** and **Fully-Connected layers**.

2.2.1 Convolutional Layers

Convolutional layers are the basic building blocks of any CNN. They are responsible for feature extraction. A convolutional layer receives a 3 dimensional input of width, height, and depth and applies a set of filters on it. A filter, also called kernel, is a 3 dimensional matrix of rows (r), columns (c), and depth. Every filter of a convolutional layer needs to have the same depth as the input shape. For example, if we have a 640×480 RGB image as an input to a convolutional layer (input shape is $640 \times 480 \times 3$), every filter in that layer must have a depth of 3. The values of the filter matrix, also called *weights*, are randomly initialized at first. As the model is training, it updates the values of these weights to reduce the error from what the network outputs and what the ground truth value is.

Every filter is applied to the entire input shape. A filter will slide over every $r \times c$ block of pixels of the input shape. At every location, the element wise multiplication of the filter matrix and the corresponding input values over all input channels is calculated. Equation 2.1 below shows the output of the convolution of an input shape I of i columns, j rows and d depth with a kernel K of $(2m+1)$ columns, $(2m+1)$ rows, and d depth.

$$Out[i,j] = \sum_{p=0}^{d} \sum_{u=-m}^{m} \sum_{v=-m}^{m} K_p[u,v] \times I_p[i-u, j-v] \tag{2.1}$$

Where:

- $Out[i,j]$ represents the output map's value at location $[i,j]$
- $K_p[u,v]$ represents the kernel's p^{th} channel value at location $[u,v]$ offset from its center
- $I_p[i-u, j-v]$ represents the input's p^{th} channel value at location $[i-u, j-v]$

The resulting element wise multiplication represents a value at a specific location in the output feature map of that filter. We can then deduce that every filter in a convolutional layer outputs a 2 dimensional feature map. The output of a convolutional

layer is a 3 dimensional shape of width (w), height (h), and depth (N) where N represents the number of filters applied in that convolutional layer. The output width and height depends on the padding and stride that are applied when a filter is convolving over the input shape. Padding is used when we want the output of a layer to be of a specific shape, where the stride is how much the filter is shifting during its convolution over the input. Figure 2.2 shows an example of a convolutional layer.

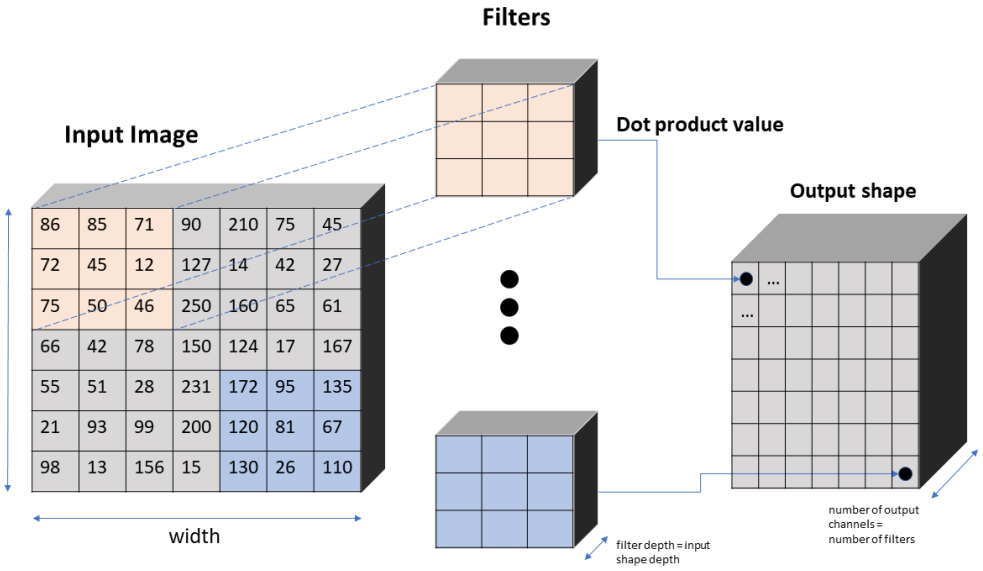

FIGURE 2.2: Sample convolutional layer.

The output feature map has the shape of $W2 \times H2 \times D2$:

- W2 = $\left\lfloor \frac{W1-F+2P}{S+1} \right\rfloor$
- H2 = $\left\lfloor \frac{H1-F+2P}{S+1} \right\rfloor$
- D2 = K

Where:

- K = Number of filters applied
- F = Filter size (assuming the conventional square filter)
- S = Convolutional stride
- P = Padding

In Figure 2.2, padding is used in order to have the shape of the output feature map the same as the input shape. In this case, the filter will be applied at every pixel such that the pixel being convolved is in the center of the filter. A stride of 1 is used in this case.

Filters detect patterns, also called features. Those patterns can be edges, corners, circles, squares... Since usually in a CNN the output feature maps of a convolutional layer are input to another convolutional layer, the deeper the network, the more complex the filters are and the more sophisticated the detected patterns become. It can then be said that in theory, the deeper the CNN is, the better it's performance gets as we are able to extract more complex features. While that is true in most cases, experiments in the literaure have shown that deeper networks might lead to the problems of overfitting [13]. Overfitting occurs when the model tunes its parameters excessively to improve its performance on train data. This leads the model to capture noise features in training data and does not allow it to generalize well to test data. In addition, deeper CNN also mean larger models and higher processing requirements causing an issue for embedded systems.

2.2.2 Activation Functions

In CNN, activation functions are applied to the outputs of hidden layers. Inspired by how our brain functions, where certain neurons in our brain respond to certain stimuli. For example, specific neurons are stimulated in our brain when we feel pain, while other neurons are stimulated when we smell a pleasant scent. We need certain neurons to fire up on certain inputs where a significant feature is detected by those neurons. Activation functions are used to achieve that by introducing non-linearity to the output of a layer. Without the activation function, no matter how many layers a CNN has, the output would always be a linear function, making it difficult to solve complex problems. The purpose of activation functions is to transform linear output values (after adding a bias) of a layer to non-liner values that will be input to the next layer. It is then possible, after achieving non-linearity, to solve complex problems and process complicated forms of data (images, text, audio...) Different

activation functions are discussed below.

- Sigmoid Function [14]: Is an activation function that maps the output to a value between 0 and 1. Defined as $F(x) = \frac{1}{1+e^{-x}}$. The main issues plaguing the sigmoid activation function are the vanishing gradients problem, in addition to the fact that its output is not zero-centered, making gradient updates go too far in different directions making optimization harder.
- Tanh Function: Defined as $F(x) = \frac{2}{1+e^{-2x}-1}$. Although the output of this function is zero-centered, it is still susceptible to the vanishing gradients problem.
- Relu Function [1]: Found to give better performance and faster convergence than other activation functions in most cases. Defined as $F(x) = max(0, x)$. The output is set to 0 if the input is negative and left unchanged if it is positive. The more positive the neuron, the more active it is. Relu function proved to be $6\times$ better than tanh on the landmark imagenet classification dataset [1].

2.2.3 Pooling layers

Pooling layers [15] are typically added after convolutional layers in a CNN to reduce the output shape. Pooling layers are defined by a filter size and a stride. The filter is convolved over the entire input feature map. At every block of pixels of every layer in the feature map where the filter is operating, the maximum value of the pixels in that block are stored in the corresponding location of the output feature map of the pooling layer. By reducing the shape of the input shape, maxpooling helps in reducing the model total number of parameters with a rather inexpensive operation enhancing computation speed. By reducing the number of features, maxpooling also helps in reducing overfitting.

The intuition for using maxpooling is that we are looking for specific patterns like edges and curves, for example, in an image. There are lots of irrelevant pixels that we do not need. With maxpooling, we assume that the pixels forming those features of interest have the highest values in a certain block of the image, defined

by the filter size. Hence, maxpooling allows us to preserve the pixels of interest and disregard others. Figure 2.3 shows a simple maxpool operation on a single layer of an input feature map.

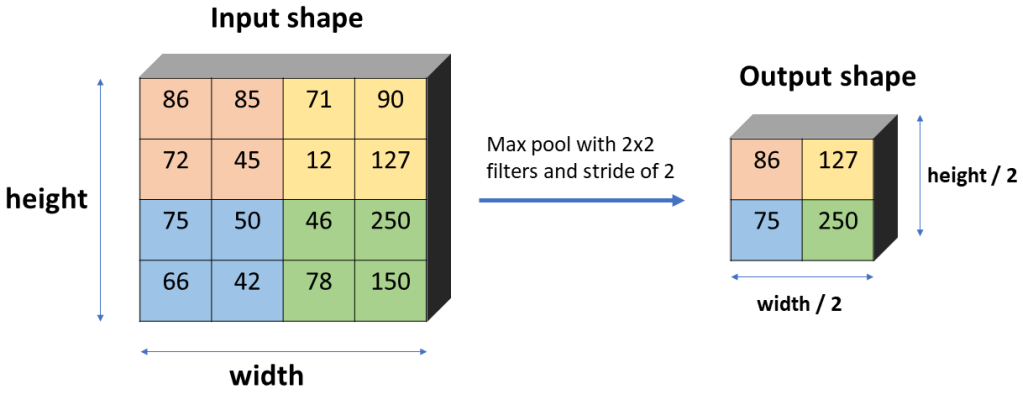

FIGURE 2.3: Simple maxpooling operation on a single layer of an input feature map.

The pooling layer output has a shape of $W2 \times H2 \times D$ where D is the depth of the input shape. $W2$ and $H2$ are calculated as follows:

- $W2 = \frac{W1-F}{S+1}$
- $H2 = \frac{H1-F}{S+1}$

Where:

- $W1$ is the width of the input shape
- $H1$ is the height of the input shape
- F is the size of the filter (square)
- S is the stride used during convolving

The most common filter size used is 2×2, decreasing the input shape size by a factor of 2.

2.2.4 Fully-Connected Layers

Fully connected layers are usually added at the end of the CNN. Fully connected layers connect every node in the current layer to every node in the previous layer. After extracting the high level features from the hidden layers of a CNN, those features are input to the fully connected layer that will find the features that best correspond to a specific class and generate a probability for this input belonging to this class. The output of a fully connected layer is a $1xN$ vector where N is the number of classes the CNN is training to classify. Every value in the output vector corresponds to the probability of the input belonging to every class. To generate those probabilities, a softmax function is used as an activation for the output of a fully connected layer. The input to the fully connected layer needs to be a one dimensional vector. So the output of the last layer before the fully connected layer is vectorized before being input to the fully connected layer. Figure 2.4 illustrates a CNN with fully connected layers in its end.

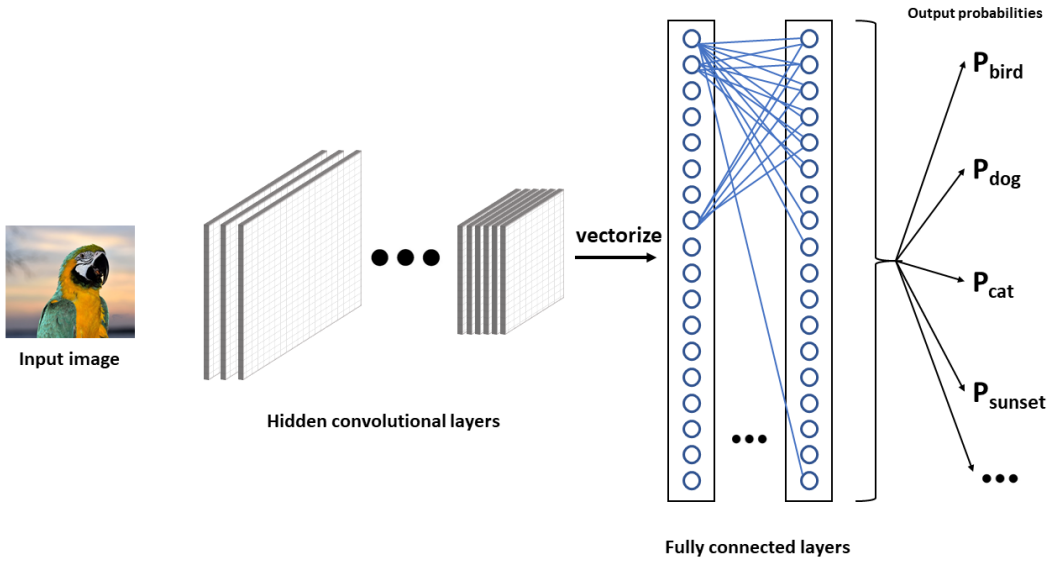

FIGURE 2.4: Figure illustrating a CNN with fully connected layers in its end. After convolutional layers, features are vectorized and input to the fully connected layers that output probabilities of the input belonging to every class the model is training to classify

2.3 Classification Models and Feature Extractors

Classification models based on deep learning do not require any form of object proposals. This is one of the main reasons that allowed such models to reach real-time performance speeds. In general, classification models are composed of a feature extractor, and fully connected layers at its end to perform classification. The last feature map generated from the feature extractor is vectorized and input to the fully connected layers for classification. Different image classification systems are explained in this section.

The first model that could be described as a *deep* neural network dates back to the 1980s and was developed by *Kunihiko Fukushima* [16] for pattern recognition. In his conclusion, Fukushima gave a description of his model that best describes all deep learning approaches today: *It also has a function of self- organization, which progresses by means of "learning without a teacher"*.

2.3.1 Alexnet

The model developed by Alex Krizhevsky and his colleagues in 2012, named Alexnet [1] is considered the first breakthrough in deep learning based models for image classification. Alexnet participated in the 2012 imagenet classification challenge and outperformed all other non-deep learning models by a large margin. It achieved a top-5 error rate (which is the rate of which the system failed to output the true class of an image among its top 5 predictions) of 15.3%. The second best result had a significantly higher top-5 error rate of 26.2%. This achievement sparked the interest in CNN research and usage that is still ongoing today. The alexnet architecture, shown in table 2.2 consists of 5 convolutional layers, 3 maxpool layers, and 4 fully connected layers. The output of the last fully connected layer is a vector of size 1000. The *ith* value in this vector corresponds to the probability of the input belonging to the *ith* class of the imagenet classes.

Neural networks have a capacity to learn. This capacity is usually governed by the network size. After a network reaches its training capacity through training, it

Layer		Dimensions	Kernel Size	Stride	Activation
Input	Image	227x227x3	-	-	-
1	Convolution	55x55x96	11x11	4	relu
	Maxpool	27x27x96	3x3	2	relu
2	Convolution	27x27x256	5x5	1	relu
	Maxpool	13x13x256	3x3	2	relu
3	Convolution	13x13x384	3x3	1	relu
4	Convolution	13x13x384	3x3	1	relu
5	Convolution	13x13x256	3x3	1	relu
	Maxpool	6x6x256	3x3	2	relu
6	FC	9216	-	-	relu
7	FC	4096	-	-	relu
8	FC	4096	-	-	relu
Output	FC	1000	-	-	Softmax

TABLE 2.2: Alexnet architecture. The dimensions of each layer are a result of performing the convolution/pooling operation using the corresponding filter size and padding.

starts to update its weights to better perform on the training set. This will lead to the model performing really well on the training set, but badly on the test set. This is the concept of overfitting. You can think of overfitting as the model is *memorizing* the features that allow it to perform well on the training set without learning the key generic features that allow it to perform well on the test set.

To solve the problem of overfitting, the authors of alexnet introduced data augmentation to increase the dataset size and add diversity and complexity to the training data. The alexnet architecture has 60M trainable parameters. To further reduce overfitting, they used a technique called drouput originally introduced by Hinton et al in 2012 [17]. With dropout, the output of neurons in a network is set to 0 with a probability of 0.5. Neurons that are dropped out do not contribute to the forward pass or the back-propagation. This way, every input follows a different path when input to the network. This forces the neurons to not rely on the presence of other particular neurons and thereby learn more robust features that are useful when other random neurons are present. Dropout is used in training only. During testing, all the neurons are used but their outputs are multiplied by 0.5 to account to the neurons that were dropped out during training. Although dropout increases the convergence time by a factor of 2, it is still a vital part to prevent overfitting and is widely used in

recent deep learning models.

2.3.2 VGG-16

Alexnet proved that deep learning can achieve state of the art performance in machine vision tasks. However, its architecture is rather complex. Simonyan et al introduced VGG16 [18] in 2014. They achieved first and second places in localization and classification tasks respectively in ILSVRC-2014 [4]. The intuition behind VGG-16 is that deeper networks extract more features and hence perform better. VGG-16 has a large number of parameters (around 138M). However, its simple architecture made it popular among researchers. VGG-16 uses uniform kernels for it's convolutional layers (3x3 kernles with a stride of 1), and uniform pooling layers (2x2 kernels with a stride of 2). The number of outputs for convolutional layers also uniformly increases. Table 2.3 shows the architecture of VGG-16. The number 16 refers to the number of weight layers. Another implementaion of VGG, VGG-19, adds 3 more convolutional layers.

Although VGG-16 has a large number of trainable parameters, it is considered a popular feature extractor for object detection models. This is because in such models the feature extractor is trimmed before the fully connected layers which substantially decreases the total number of trainable parameters.

2.3.3 Resnet

The general trend in CNN architecture evolutions is that networks are getting deeper and deeper. However, deeper networks are harder to train because of the famous vanishing gradient problem. As the gradient is back-propagated to earlier layers, the continuous multiplication might make its value negligible. This makes deeper networks prone to degradation. Resnets [19] introduces the concept of skipped connections which allow a network to take the activation of a layer output and add it to the input of any other layer in the model. This allows the training of very deep networks of hundreds or even thousands of layers. Resnets are built from residual

Layer		Dimensions	Kernel Size	Stride	Activation
Input	Image	224x224x3	-	-	-
1	Convolution	224x224x64	3x3	1	relu
2	Convolution	224x224x64	3x3	1	relu
	Maxpool	112x112x64	2x2	2	relu
3	Convolution	112x112x128	3x3	1	relu
4	Convolution	112x112x128	3x3	1	relu
	Maxpool	56x56x128	2x2	2	relu
5	Convolution	56x56x256	3x3	1	relu
6	Convolution	56x56x256	3x3	1	relu
7	Convolution	56x56x256	3x3	1	relu
	Maxpool	28x28x256	2x2	2	relu
8	Convolution	28x28x512	3x3	1	relu
9	Convolution	28x28x512	3x3	1	relu
10	Convolution	28x28x512	3x3	1	relu
	Maxpool	14x14x512	2x2	2	relu
11	Convolution	14x14x512	3x3	1	relu
12	Convolution	14x14x512	3x3	1	relu
13	Convolution	14x14x512	3x3	1	relu
	Maxpool	7x7x512	2x2	2	relu
14	FC	4096	-	-	relu
15	FC	4096	-	-	relu
16	FC	1000	-	-	Softmax

TABLE 2.3: VGG-16 architecture

blocks which are explained below.

Consider figure 2.5. It is composed of 2 layers where:

- $a^{[l]}$ represents the activation input to the first layer
- $a^{[l+1]}$ represents the activation output from the first layer and input to the second layer
- $a^{[l+2]}$ represents the activation output from the second layer

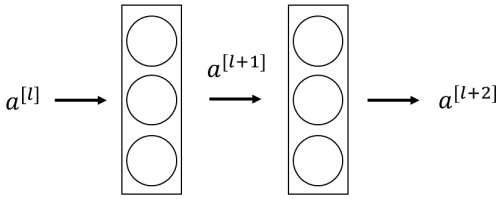

FIGURE 2.5: 2 Layer network

In a normal CNN workflow, $a^{[l+1]}$, and $a^{[l+2]}$ would be calculated as follows where W and b are respectively the weights and bias applied at a specific layer:

- $a^{[l+1]} = Relu(W^{[l+1]} \times a^{[l]} + b^{[l+1]})$
- $a^{[l+2]} = Relu(W^{[l+2]} \times a^{[l+1]} + b^{[l+2]})$

With residual blocks, it is possible to add the activation $a^{[l]}$ to the input to layer 2 of the network. Hence, $a^{[l+2]}$ would become:

- $a^{[l+2]} = Relu(W^{[l+2]} \times a^{[l+1]} + b^{[l+2]} + a^{[l]})$

We can say that we have made a *shortcut*, also called *skip connection*, over the first layer. $a^{[l]}$ is injected after the linear part (convolution operations and addition of bias), and before the non-linear part (applying Relu). This way, if the linear operation of the second layer (convolution of weights with input + bias) is negligible, the skip connection allows it to keep the same value of the input to the first layer $a^{[l]}$ before applying Relu.

2.3.4 ResNext

A variant of Resnet, named ResNext [20], was introduced by Xie et al. ResNext won 2nd place in ILSVRC 2016 [21] in the classification task. Resnext showed performance improvements over its Resnet counterpart as it achieved a 22.2% top-1 error

rate on the imagenet dataset while Resnet obtained 23.9. ResNext Blocks suggest that instead of having a simple residual block, we could have a number of paths with the same topology that the input goes through. The number of paths is called *cardinality*. The output of all paths is merged to create the final output of the ResNext Block. Experimental results in [20] showed that increasing the cardinality leads to an increase in accuracy. ResNext Blocks follow the intuition that having a high cardinality with smaller depth and width at each individual path would have roughly the same number of parameters as a regular residual block. The authors of ResNext show that their architecture adapts easily to new datasets because of its simple architecture and the fact that it is controlled by only one hyperparameter, the cardinality. Figure 2.6 shows a residual block and its equivalent resnext block.

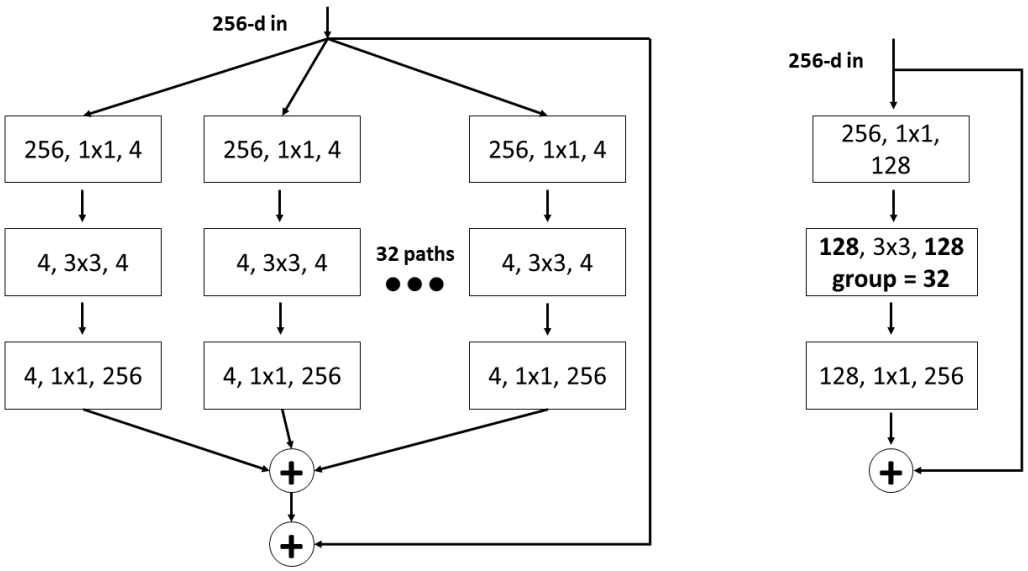

FIGURE 2.6: Equivalent Resnext (left) and residual (right) blocks. Resnext block has a cardinality of 32 and all paths have the same topology. The final outputs of each individual path are merged to create the final output with depth 256. A layer is shown as (Number of input channels, filter size, number of output channels).

2.3.5 Inception Networks

When designing a convolution layer in a CNN, the system developer needs to specify the size of the filter to be applied. This might be a difficult process as the best kernel size differs depending on the input. Larger filter sizes are preferred when objects of interest are scattered throughout the image, while smaller kernels perform better

when the objects are close to each other. Before inception networks, CNN developers would usually stack more and more convolutional layers with different filter sizes applied at each layer with the hopes of boosting up the performance. However, deeper models are more difficult to train as they are prone for overfitting. The inception network [22], also called GoogleNet, suggests a solution to this problem by applying filters of different sizes at each layer (1×1, 3×3, and 5×5), and applying padding to have a uniform output shape. The outputs of convolutions from different filters are stacked to form the final output. This makes the network more complex, but it also significantly enhances performance. GoogleNet is obviously a really deep network and as any other deep network, it is subject to overfitting. To address this problem, the authors apply softmax to the output of two intermediate hidden layers and computed an auxiliary loss for each of the two classifiers. The total loss is calculated as shown in equation 3.1.

$$Loss_{Total} = Loss_{Real} + 0.3 \times Loss_{Auxilliary1} + 0.3 \times Loss_{Auxilliary2} \tag{2.2}$$

This is the first version of inception networks, now referred to as Inception-V1. Enhancement were later on made to it leading to the emergence of Inception versions 2, 3, and 4.

In their paper titled *Rethinking the inception architecture for computer vision* [23], Szegedy et al introduced improvements to the initial inception model. The inception network was very popular after its release but two of the main issues plaguing it were representational bottleneck and computational complexity. The loss of information that occurs when an input is significantly resized due to a convolutional operation is referred to as a representational bottleneck. To address these issues, the authors proposed the replacement of 5×5 convolutions by two successive 3×3 convolutions (Figure 2.7). This leads to the reduction of the representational bottleneck. In addition, this factorization was found to reduce computational complexity as a 5×5 is 2.78 times more expensive than a 3×3 convolution.

Moreover, they proposed factorizing $n \times n$ convolutions to $1 \times n$ and $n \times 1$ convolutions. Although the two operations are equivalent in terms of extracted information, this factorization was found to be 33% more efficient than a single $n \times n$ convolution. For example, a 3×3 convolution would be replaced by a 1×3 convolution followed by a 3×1 convolution (Figure 2.8).

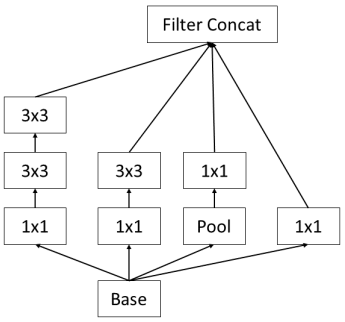

FIGURE 2.7: 5×5 convolutions of Inception-V1 are replaced by two successive 3×3 convolutions.

A third inception module (Figure 2.9) was also introduced that was inspired by the previous two modules. The idea was to make the filter banks wider instead of deeper to reduce the representational bottleneck as deeper modules performing excessive dimension reduction also lose more information. Such modules would be specially important in deeper networks as they allow the generation of more robust feature maps.

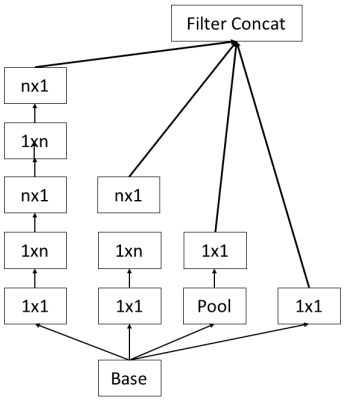

FIGURE 2.8: $n \times n$ convolutions are replaced by successive $1 \times n$ and $n \times 1$ convolutions.

A newer inception model was thereby developed using the three proposed inception modules. The network architecture is given in table 2.4

In addition to the 3 enhanced inception modules proposed, the authors applied batch normalization which adjusts the range in which activation outputs vary around (covariance shift). It scales and adjusts the output allowing for significantly faster training. Batch normalization allows the use of higher learning rates as it ensures that no layer output would go too high or too low. It also helps in reducing overfitting as it has a slight regularization effect.

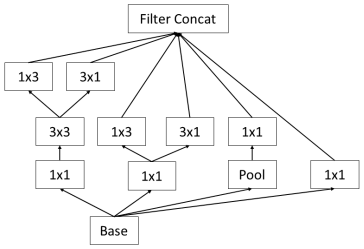

FIGURE 2.9: Wider instead of deeper filter banks [23]

Layer Type	Filter size/stride or remarks	Input size
Convolution	$3 \times 3/2$	$299 \times 299 \times 3$
Convolution	$3 \times 3/1$	$149 \times 149 \times 32$
Convolution (Padded)	$3 \times 3/1$	$147 \times 147 \times 32$
Maxpool	$3 \times 3/2$	$147 \times 147 \times 64$
Convolution	$3 \times 3/1$	$73 \times 73 \times 64$
Convolution	$3 \times 3/2$	$71 \times 71 \times 80$
Convolution	$3 \times 3/1$	$35 \times 35 \times 192$
3 \times Inception	As in figure 2.7	$35 \times 35 \times 288$
5 \times Inception	As in figure 2.8	$17 \times 17 \times 768$
2 \times Inception	As in figure 2.9	$8 \times 8 \times 1280$
Maxpool	8×8	$8 \times 8 \times 2048$
Fully Connected	Logits	$1 \times 1 \times 2048$ \times 2048
Fully Connected	Classifier	$1 \times 1 \times 2048$ \times 1000

TABLE 2.4: Enhanced inception network architecture [23]

The adjustment of activation values adds a certain noise to them, similar to what dropout does. Hence, using batch normalization, we can use less dropout. Although still necessary, using less dropout reduces the loss of information which is a desired outcome.

There was confusion over the definitions of versions 2 and 3 of the inception network. Szegedy et al clarified this confusion in their Inception-V4 paper [24]. Stating *"The Inception deep convolutional architecture was introduced as GoogLeNet in (Szegedy et al. 2015a), here named Inception-v1. Later the Inception architecture was refined in various ways, first by the introduction of batch normalization (Ioffe and Szegedy 2015) (Inception-v2). Later by additional factorization ideas in the third iteration (Szegedy et al. 2015b) which will be referred to as Inception-v3 in this report."* [24]

Inception-V2 and Inception-V3 proved that the proposed modifications were quite effective. They achieved 6.3% and 5.6% top-5 error rate respectively in the ILSRVC-2012 challenge [4].

Inception V4 was introduced in 2017 [24]. Little change was made to individual inception blocks, as the significant improvements were made to the network stem

(operations performed before inception blocks). Inception-V4 has a simpler, more uniform architecture and more inception blocks than Inception-V3. Inception-V4 proved to be an improvement over its predecessors as it achieved 5.0% top-5 error rate in the ILSRVC-2012 [4] challenge. The structure of Inception V4 and its stem are shown in figure 2.10 and the 3 types of inception blocks used are illustrated in figure 2.11.

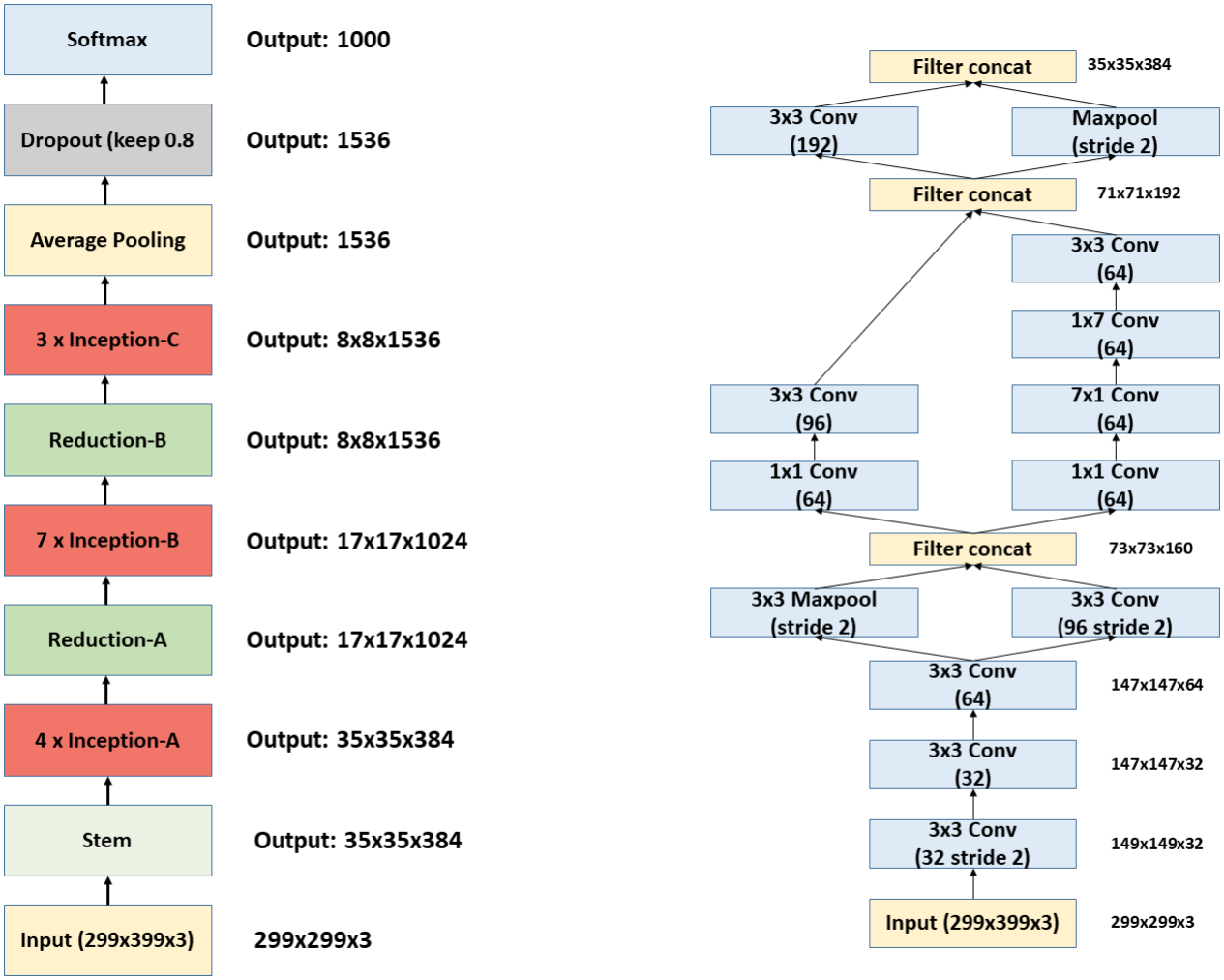

FIGURE 2.10: Inception-V4 structure (left) and its stem (right)

Resnets [19] proved to increase model accuracy without significantly affecting computation time. Hence, Szegedy et al combined inception and residual blocks in Inception-Resnets [24]. The authors tested with different designs and elaborated two of them:

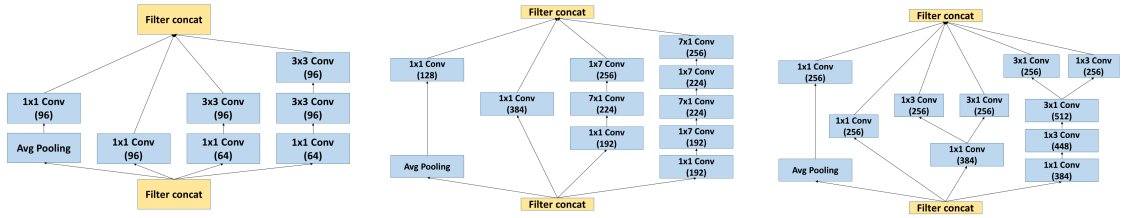

FIGURE 2.11: Inception-V4 modules A (left), B (center), and C (right)

- Inception-Resnet-V1: Roughly has the same computational cost as Inception-V3
- H2 = Inception-Resnet-V2: Roughly has the same computational cost as Inception-V4. Uses the same stem as Inception-V4

Both versions of Inception-Resnet use similar inception-residual blocks with a varying number of outputs. However, they have different stems. The structure of the overall model architecture of both versions of Inception-Resnet, and the stem of Inception-Resnet-V1 are shown in figure 2.12. The inception-resnet modules used for V1 are shown in figure 2.13, while those used for V2 are shown in figure 2.14.

The introduction of residual connections allowed for faster training of inception networks while not affecting performance. Inception-Resnet-V1 achieved 5.5% top-5 error rate on the ILSVRC 2012 [4] validation set while Inception-Resnet-V2 achieved 4.9% on the same set.

2.3.6 Mobilenets

Google introduced mobilenets[2] in 2017 as a model suitable for mobile and embedded systems applications. While in theory having deeper networks enhances the model performance, this comes at a heavy size and computational cost making it impossible to run real-time applications using deep learning models. Mobilenet addresses this issue by using depth-wise separable convolutions to reduce the number of parameters without compromising the quality of extracted features.

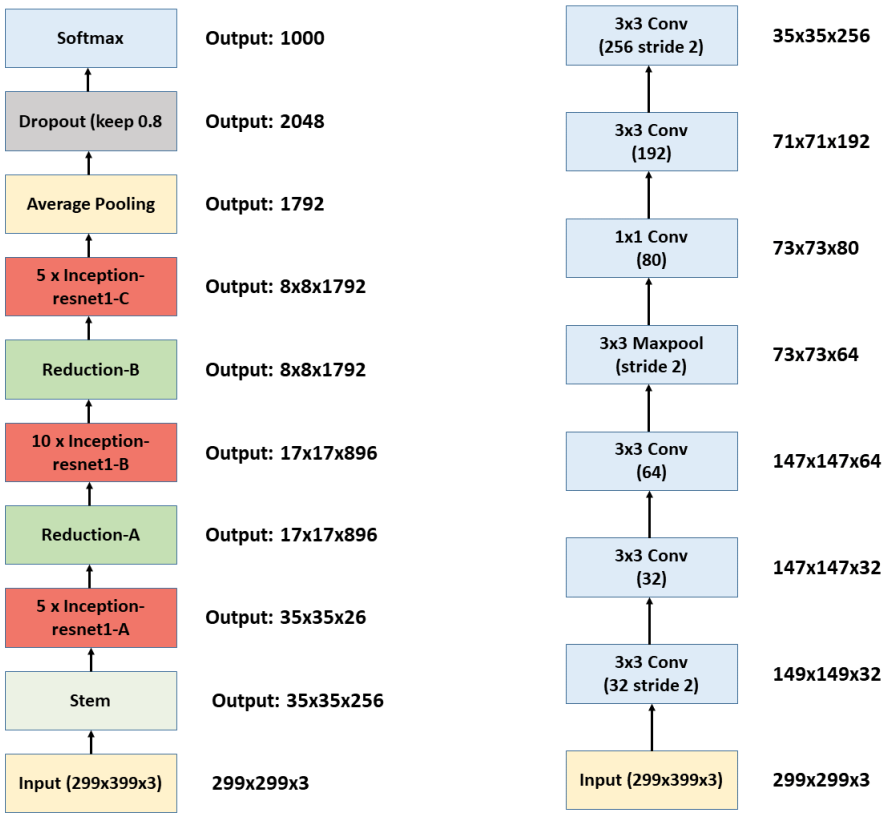

FIGURE 2.12: Structure of both versions of the Inception-Resnets models (left), and the stem of Inception-Resnet-V1 (right). Although both models share the same overall structure, the resnet-inception modules differ as shown in figures 2.13 and 2.14

A regular convolutional layer applies a set of N filters on an input volume with M channels. Each filter should also have M channels and outputs a feature map of depth 1. The outputs of all filters applied are concatinated to have an output volume of depth N. Regular convolutions filters and combines inputs to produce a new output in one step. Depthwise separable convolutions used in mobilenets factorize this process to two steps:

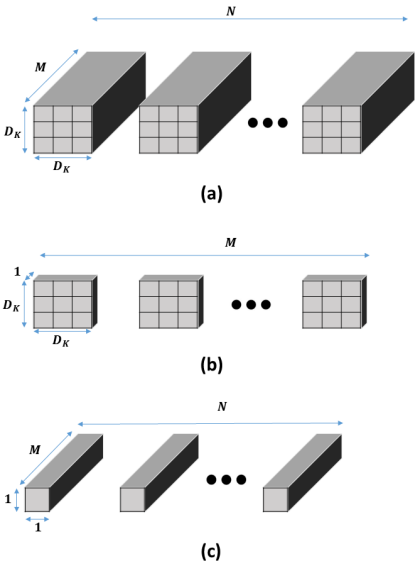

FIGURE 2.15: (a) shows a regular convolution with N filters which is factorized to a depthwise convolution (b) followed by a pointwise convolution (c)

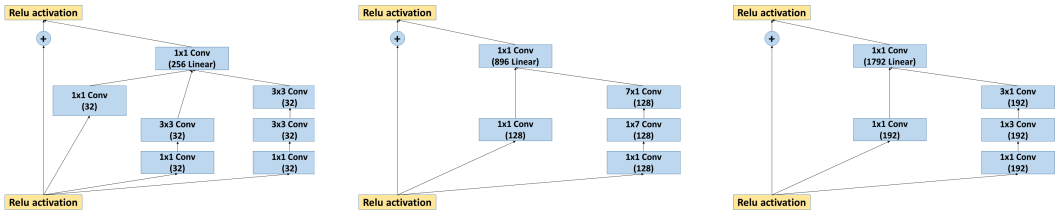

FIGURE 2.13: Inception-Resnet modules A (left), B (center), and C (right) used in Inception-Resnet-V1

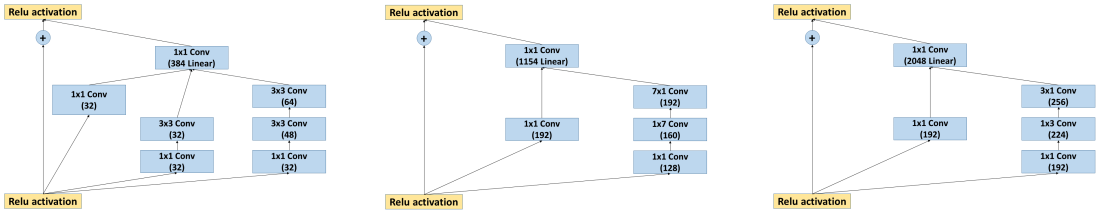

FIGURE 2.14: Inception-Resnet modules A (left), B (center), and C (right) used in Inception-Resnet-V2

- A depthwise convolution that applies
  a single filter to every input channel
- A pointwise convolution that applies
  a 1×1 convolution to combine the outputs of the depthwise convolution producing the output channel of a single filter. Outputs from all filters applied are then concatenated to form the final output volume.

Figure 2.15 shows how a regular convolution is factorized using depthwise separable convolutions.

Using depthwise separable convolutions to factorize regular convolutions leads to a reduction in computation of:

$$\frac{1}{N} + \frac{1}{D_K^2}$$

Mobilenets use 3×3 dephwise separable convolutions leading to 8 to 9 reduction in computational cost. The full structure of mobilenet is shown in table 2.5.

Layer Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$56 \times 56 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times \left\{ \frac{Convdw/s2}{Conv/s1} \right\}$	$\frac{3 \times 3 \times 512 dw}{1 \times 1 \times 512}$	$\frac{14 \times 14 \times 512}{14 \times 14 \times 512}$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

TABLE 2.5: Complete mobilenet architecture [2]

2.4 Conclusion

In this chapter, we provided an overview of traditional detection models and introduced convolutional neural networks. We explained some of the well known feature extractors for classification and their structure. In the next chapter, we will provide a literature review of the related work of our approach in the tasks of utilizing RGBD data, object detection, and keypoint detection.

Chapter 3

Related Work

We utilize depth information for generating object proposals in our approach. We also highlight experimental results utilizing both RGB and depth channels for human head and shoulder detection. In this chapter, we present a literature review of human detection using RGBD data, in addition to related work in the tasks of object and keypoint detection.

The structure of this chapter is as follows: Section 3.1 gives an overview of how depth sensors function. Different object detection models are described in Section 3.3. Object proposals are discussed in Section 3.4. Finally, work related to keypoint detection is elaborated in Section 3.5

3.1 Depth sensors

Depth sensors are used to capture depth information of a scene. Usually, a depth sensor is composed of two main components:

- Infrared projector
- Infrared camera

The infrared projector shoots a pattern of dots across the scene. Depth sensors usually have a diffractive optical element that distorts the dots generated by the infrared projector. While the infrared dots are invisible to the naked human eye, the infrared camera is capable of capturing them. The intensities of these infrared dots are stored in an image. In order to calculate the depth at each pixel, the captured infrared image is processed by the depth sensor processor to generate the final depth

image. The processor relies on the displacement of the dots to calculate the pixel intensities, representing distance to the sensor. Closer objects would have a dense distribution of infrared dots, while further objects would have a more distorted pattern.

We use an infrared depth sensor in our work. Other range measuring sensors, like LIDAR (Light Imaging Detection And Ranging), emit laser lights at a surface and measure the time the light takes to return to the source. Since the speed of light is known, the distance can be derived from equation below:

$$Distance = \frac{\text{Speed of Light } \times \text{Time of flight}}{2} \tag{3.1}$$

3.2 People detection utilizing RGBD data

Depth sensors introduced a new, robust channel that gives information about pixel depth in a scene. This information could be used for 3D scene understanding and reconstruction. There are several approaches in the literature that utilized the depth channel from an RGBD sensor for human detection.

3.2.1 People detection in RGB-D data

[25] propose the histogram of oriented depth (HOD) descriptor, which is inspired by the histogram of oriented gradients (HOG) [10]. HOD encodes the orientation of depth changes from depth frames in a similar way that HOG encodes gradient orientations from RGB images. The calculated orientation is stored in a 1D histogram and the aggregation of blocks of those histograms result in the HOD features. Those features are used to train a SVM [5]. In [25], the authors separately train 2 SVM, one on HOD features, and another on HOG features. During detection, information from both SVM is used to detect humans in a way that is robust to illumination

changes (HOD is unaffected by poor illumination), and depth degradation (HOD feature quality degrades as the distance between objects and the sensor increase. HOG features are solely used in such cases).

3.2.2 Real-time human detection and tracking in complex environments using single RGBD camera

In [26], the authors make use of the depth information in order to generate proposals for human detection. They use local maximum height pixels as plausible human head tops. Using this method, they reject all points that have a height lower than 0.6 meters or higher than 2 meters, as humans are expected to have heights within this range. This drastically reduces the search space as points close to the floor and ceiling are determined to be definitely not human head tops. A SVM is then trained with two sources of features. The first is a histogram of height difference, where the difference between the plausible human head top point and neighbouring pixels of the upper human body is calculated. This will give distinctive information as the human upper body has a distinctive shape from other objects. The second feature is a joint histogram of color and height, where the color and depth information are used to generate a discriminative representation of the head. Pixels that are within 0.2 meters below the possible head top point are considered to build this histogram where the HSV color statistics of those pixels are collected over 5 height intervals.

3.2.3 Robust 3D Human Detection in Complex Environments with Depth Camera

In [27], the authors generate candidate head top locations by taking the highest point in a connected region of pixels generated using depth information. They then generate bounding boxes from the candidate locations and encode a 3 channel representation of the bounding box using depth information only. The first channel represents the raw depth data. The second channel encodes the first and second order gradients of pixels using 8 templates that describe the relationship between 3 pixels. The third

channel encodes the height difference between the candidate heat top location and other pixels in the generated box. Alexnet [1] is then trained on this 3 channel representation to classify the proposals as heads or non-heads. In our implementation, we use the proposal generation algorithm used in [27] and optimize it for efficiency.

3.3 Object Detection

The input to classification models is a single image, and the output is a class label, or a probability associated with a certain class. Meaning that this input image belongs to this class. In object detection, we are concerned with obtaining the locations of objects, possibly of different classes, in a single image. Hence, from an object detection model, we need the following output:

- List of bounding boxes, specifying the x, y, width, and height values of all objects detected in the image.
- Class labels corresponding to the bounding boxes specifying to which class each detected object belongs to.
- Probabilities associated with each bounding box and label. Also referred to as confidence.

Deep learning models proved efficient in object detection tasks. An object detection model based on deep learning generally consists of two parts:

- The base network, which is a feature extractor. They are common CNN discussed in Section 2.3. Usually trimmed before the fully connected layers that are used for classification.
- An object detection framework (SSD [28], Faster-RCNN [29], Yolo [30]

Any feature extractor can be used as a base network for object detection. However, a few of the common feature extractors are:

- Inception [22]

- Mobilenet [2]
- ResNet [19]

In this section, we will introduce the famous object detection frameworks SSD [28], and Faster-RCNN [29]

3.3.1 Faster RCNN

Faster-RCNN [29] is a famous object detection framework. It is inspired by its predecessors R-CNN [31] and Fast-RCNN [32] and improves some of their biggest weaknesses: computation time. Faster-RCNN uses a feature extractor to generate high-level feature maps from an input image. A set of anchor boxes are then created for every location in the processed feature map. Anchors are fixed rectangles with different pre-set sizes and aspect ratios. The set of created boxes from anchors are then input to a region proposal network (RPN) that has two outputs:

- Probability that an anchor represents an object, also referred to as "*objectness score*". This is binary classification where an anchor can either be foreground (object) or background. Note that the RPN does not predict the class of an object, only whether or not it is actually an object.
- Four values representing the $x, y, width$, and $height$ offset of the anchor box with respect to the ground truth box. This is used to better fit the anchor box to get the final proposals.

Having the final list of proposals from RPN, Region of Interest pooling is applied to retrieve a fixed-size feature map from the preprocessed feature map that was output from the base network. This is done to avoid recomputing feature maps for all the proposals.

R-CNN is used as the last step in the Faster-RCNN workflow. Every extracted feature map is input to R-CNN that flattens the map and uses 2 fully connected layers with Relu activations. A couple more fully connected layers are used to:

- First FC layer to classify the object

- Second FC layer outputs four values to regress the final bounding box coordinates

Faster-RCNN achieved 73.2 mAP at 7 FPS on the Pascal VOC 2007 dataset [33]. The complete architecture of Faster-RCNN is shown in Figure 3.1.

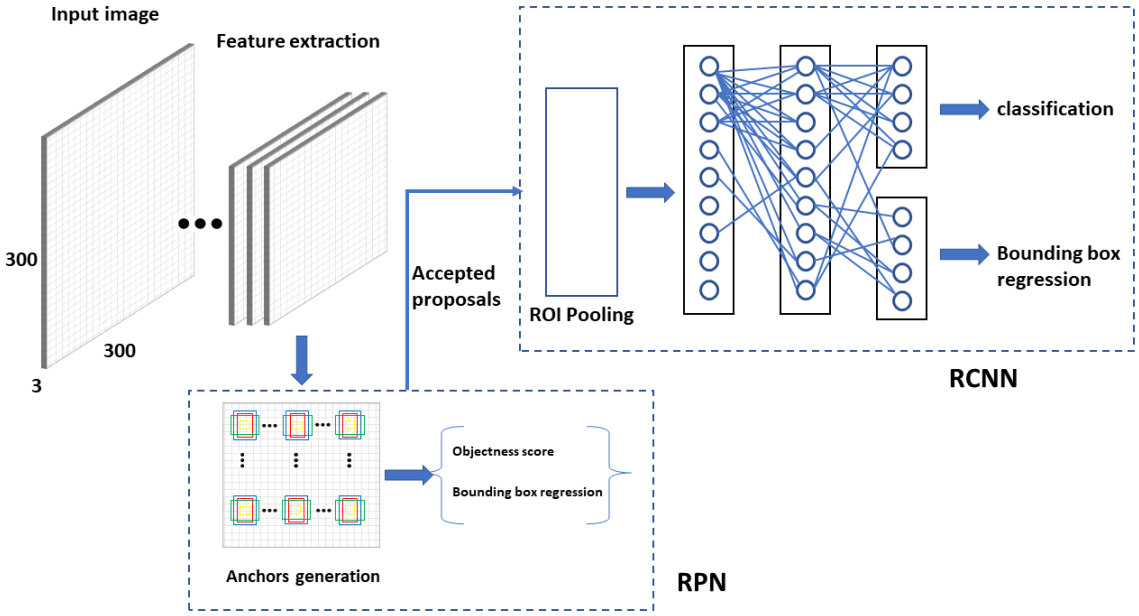

FIGURE 3.1: Faster-RCNN architecture

3.3.2 Single Shot Multibox Detector (SSD)

Although Faster-RCNN introduced substantial performance improvements over its predecessors [31] [29], it was still not feasible for real-time applications. This was mainly due to the fact that Faster-RCNN separates the processes of generating proposals and classifying them. SSD addresses this issue as it generates proposals and classifies them in one network pass (single shot). Similar to Faster-RCNN, SSD requires a feature extractor which is usually one of the common classification models trimmed before any classification layers. Convolutional feature layers are added to the end of the base network. These layers decrease in size to allow the detection of objects of varying sizes.

For each one of the added layers, a set of default boxes are associated to every location of it. The default boxes are of varying aspect ratios, so we have several anchor

boxes at every cell in the layer. For each default box, the offset to a ground truth box is calculated (4 values), in addition to class scores representing the probability that the default box belongs to each class. The default boxes are similar to the concept of anchor boxes in Faster-RCNN, but they are applied to different feature maps of different resolutions allowing for more robust detections. SSD achieved 74.3 mAP at 46 FPS on the Pascal VOC 2007 dataset [33]. An illustration of default boxes at multiple scales is shown in Figure 3.2, while the full structure of SSD is shown in Figure 3.3.

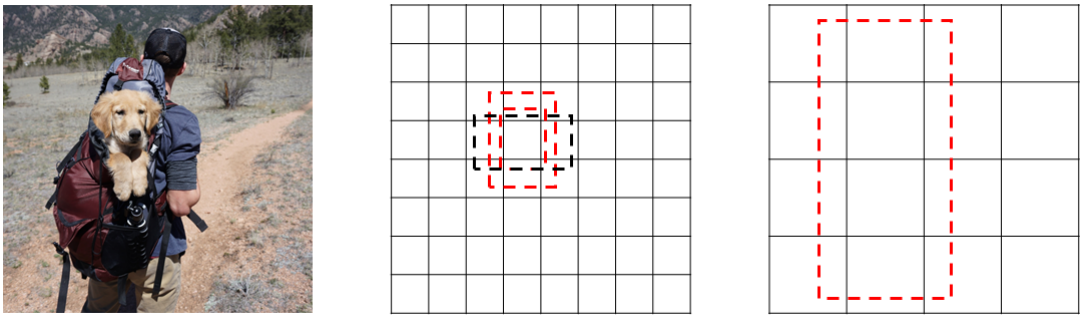

FIGURE 3.2: An image with two ground truth objects, a man and a dog (left). An 8×8 feature map where two of the default boxes are matched with the dog (middle), and a 4×4 feature map were a default box is matched to the man (right).

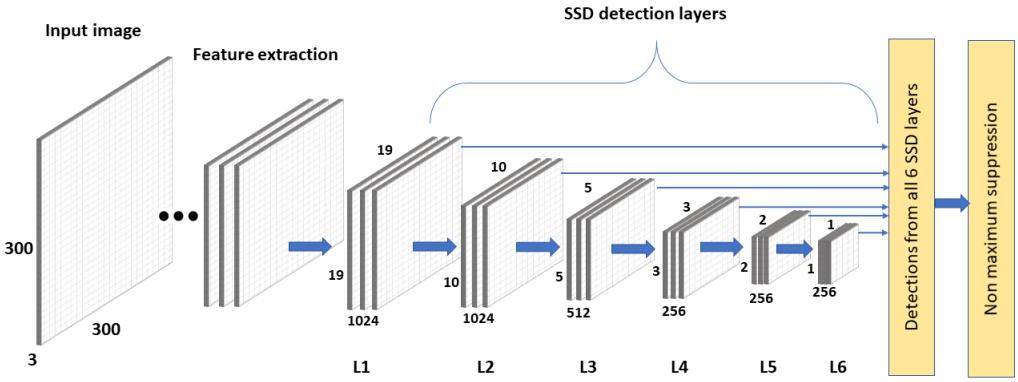

FIGURE 3.3: SSD structure

3.4 Object Proposals

One of the main problems with traditional methods for image analysis is the slow processing time. In order to detect objects of different sizes in different regions of an image, traditional methods needed to employ algorithms such as the sliding window approach and image pyramids. Because such algorithms need to be run sequentially, they severely affect processing times. This is one of the main reasons why traditional methods were not able to run in real-time speeds.

To a large extent, deep learning has offered a solution to this problem with the concept of object proposals for detection systems. Object proposals are regions of interest in an image. They represent bounding boxes in which a deep learning model suspects that an object *might* exist. Different approaches for proposal generation using CNN were discussed in Section 3.3. As our approach works on generating proposals using an image processing algorithm, we will present two non CNN proposal generation algorithms that function on RGB images. We then introduce the object proposal algorithm used in our work that utilizes depth data for fast generation of proposals.

3.4.1 Selective Search

Introduced in [34], the selective search algorithm is one of the well known image processing methods for generating region proposals. In fact, it was used in the original RCNN method [31] to generate region proposals that are then classified. The selective search algorithm relies on image segmentation, which is dividing an image to a set of disjoint partitions. They use the efficient graph-based segmentation method proposed in [35] to heavily segment the image (dividing it into many small partitions). Image segmentation partitions the image based on pixel intensities. Pixels that have close intensity values are considered to belong to the same partition. After segmenting the image, bounding boxes around segmented partitions are generated representing object proposals. This is the first stage of the selective search algorithm. Since the generated proposals should include objects of different scales

and aspect ratios, segmented partitions are grouped with their neighbours based on similarity. Bounding boxes are then created around the newly merged partitions to generate bigger object proposals for bigger objects. The process of combining neighbouring partitions to create bigger proposals continues until we end up with one proposal around the entire image.

3.4.2 Edge boxes

Zitnick et al proposed the edge boxes algorithm [36], which is another approach for object proposals using edges. The advantage of using edges is that they are efficient to compute and they offer a good representation of object boundaries. The idea is that the number of contours that are fully contained in a box gives an indication of the likeliness of an object existing in the box. At first, edges are computed over the entire image. A sliding window approach is utilized to evaluate boxes at all locations, sizes, and aspect ratios. At every possible location, the bounding box is scored based on the sum of magnitudes of all edges of contours that are fully contained in the bounding box minus the magnitudes of edges belonging to contours that intersect with the box, but are not fully enclosed inside of it. The contours are created from edge groups, which are grouping of edges based on affinity. Although there is a huge number of boxes to be evaluated from the sliding window approach, the efficient computation of edges made the edge boxes algorithm a reliable and fast method for proposal generation, as it runs at 4 FPS which was acceptable at the time.

3.4.3 Candidate Headtop Locations (CHL)

Even with state of the art deep learning approaches for object detection, the number of proposals to be evaluated was still large (around 8000 for SSD [28] and 300 for Faster-RCNN [29] after applying NMS). To address this problem, [27] introduced an intuitive way for generating proposals from depth images. The preliminary version of the algorithm was introduced in [37]. The intuition was that the head region is less likely to be occluded in images. Also, it was found that pixels that belong to

the same person have continuous depth values and usually vary within a specific range. Therefore, scanning the depth image from the top-left corner to the bottom right corner, projecting every pixel to the 3D-plane and connecting pixels that have a eucledian distance less than a threshold would allow for the fast generation of connected regions in an image. With this approach, every human in the scene would be a separate connected region. The top pixel of every connected region is taken as a candidate human head-top location. For each CHL, a bounding box is created as the proposal around a human head. The width of the windows is $3 \times R$ and the height is $4 \times R$ where $R = r \times k/d$ and r resembles the average radius of a human head in the real world (15cm), k is a constant factor calculated from camera intrinsics, and d is the depth value of the CHL pixel. A 3 channel representation is generated from the depth information of the proposal and Alexnet [1] is used to train for the classification of heads based on this 3 channel representation. The CHL algorithm produces an average of 23 proposals per image.

3.5 Keypoint Detection

Deep learning models output probabilities for classification tasks and bounding boxes for detection tasks. However, CNN can also be used for the task of keypoint detection. Keypoint detection is the task of localizing locations of interest within an image represented by the x and y coordinates of these locations. A popular application of keypoint detection is facial recognition where it is required to detect the locations of human eyes, nose, and mouth boundaries (left and right). Detecting those keypoints would be beneficial in driver assistant systems to detect if a driver falls asleep, for example.

Another application for keypoint detection is human pose estimation, as detecting the locations of shoulders, elbows, hands, knees, and feet would assist in determining the pose a human is taking and thereby predict the action a human is performing.

In this section, we will discuss three methods used in the literature for keypoint detection using deep learning.

3.5.1 R-CNNs for Pose Estimation and Action Detection

Gkioxari et al [38] take inspiration from R-CNN to build a CNN for the tasks of person detection, pose estimation, and action detection. Like R-CNN, their model takes object proposals as input. They generate proposals using multiscale combinatorial grouping [39]. Their model has 3 fully connected layers, for the following tasks:

- **Person detection:** Output the probability that a proposal belongs to a person.
- **Pose estimation:** Authors defined a set of keypoints to detect that will allow for accurate pose estimation. The keypoints are of the human nose, and the right and left shoulders, elbows, wrists, hips, knees, and ankles. As each keypoint is characterized by its x and y coordinates, this fully connected layer outputs $13 \times 2 = 26$ values.
- **Action classification:** A fully connected layer to classify the action a person is performing.

They calculate the average precision of keypoint(APK) based on the distance between the detected keypoint and its corresponding ground truth value. If the distance between the detected keypoint and the groundtruth is less than $0.2 \times H$, where H is the height of torso of the corresponding ground truth, the keypoint detection is considered correct. They achieved 2.5% higher mAP than the previous state of the art, k-poselets [40].

3.5.2 Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks

The CNN used in our work is inspired by this work [41]. The authors propose the use of a cascade of 3 CNN for the task of facial landmark (keypoint) detection. The first CNN is a shallow and fast model that quickly produces candidate windows of

faces (proposal network, P-Net) and their bounding box regressions. Highly overlapping windows are merged using nonmaximum suppression (NMS). The output of this model is input to a more complex model that rejects many of the proposals as non-faces, further refines bounding box regressions, and applies NMS (refinement network, R-Net). Finally, the last CNN further refines the remaining proposals by classifying them as faces and non-faces, in addition to predicting the locations of 5 facial keypoints (eyes,nose, and mouth left and right positions). In addition, the authors introduce online hard example mining. Instead of manual labelling of hard examples, the authors sort the facial classification losses in the forward propagation and take the top 70% of the samples as hard samples. The gradients are calculated only for those hard samples in the back-propagation. Their approach achieved a state of the art mean error rate of 6.9%.

The structure of their cascaded network is shown in Figure 3.4.

Our model for classification and regression is inspired by this paper's O-Net, the last CNN. As we produce the proposals using the candidate head-top locations algorithm [27], we do not need P-Net and R-Net.

3.5.3 OpenPose

OpenPose [8] is a real-time method for pose estimation proposed by Cao et al in 2017. The computational cost of using openpose is invariant to the number of people present in an image. The method consists of 2 stages. The first stage consists of a feature extractor (first 10 layers of VGG-16) while the second stage is composed of a 2-branch multi-stage CNN. The first branch generates confidence maps which are 2D representations of the likelihood that a pixel belongs to a body part. It is a grayscale image that has high values where the confidence is high. The second branch generates part affinity fields (PAF), which are 2D vector fields representing the degree of association between different detected keypoints. Those generated PAF will be used to connect detected keypoints to generate the skeleton for each detected person. The multi-stage approach of each branch allows the refinement of keypoints and PAF at every stage.

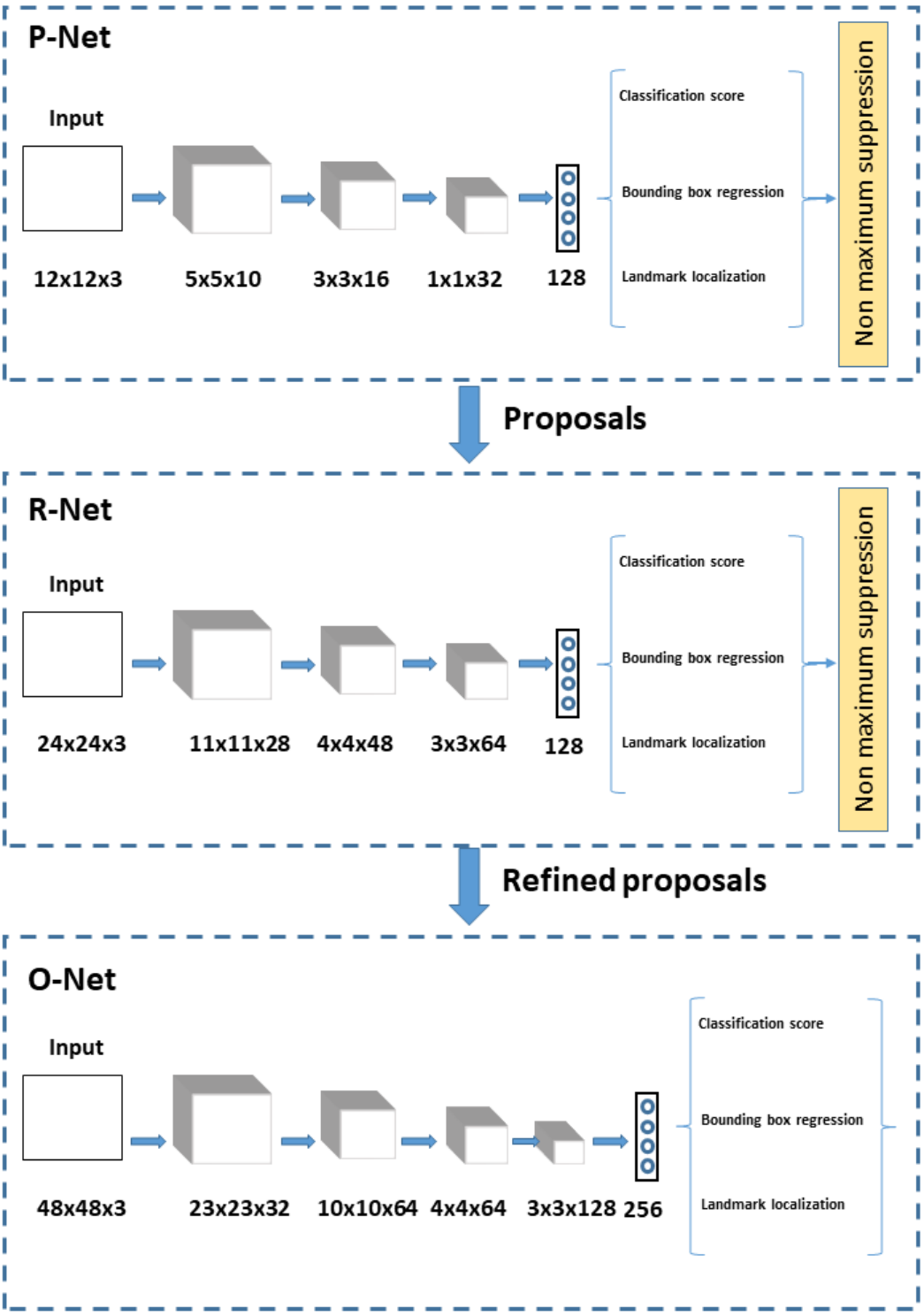

FIGURE 3.4: Structure of the cascaded network for facial landmark detection.

Since OpenPose is considered the state of the art in human pose estimation, we compare our shoulder detection precision to that of OpenPose.

To our best knowledge, there are no approaches that perform human head detection and shoulder keypoint localization using one model. In addition, existing approaches tend to have high computational requirements. For those reasons, we develop a now model that accomplishes those two tasks and focus on optimizing it to run on embedded systems.

3.6 Conclusion

In this chapter, we provided a literature review of approaches to human detection using RGBD data, in addition to well known methods for object and keypoint detection. In the next chapter, we will discuss the methodology that we have followed in our work. We will also discuss our model architectures and configurations.

Chapter 4

Methodology and System Description

We have discussed in the previous chapter that CNN proved efficient in different tasks of machine vision. They introduced a lot of improvement over traditional methods and they are currently considered the state of the art. For that reason, we have decided to adopt a CNN approach in our work. We want to build a system that is capable of detecting humans in a scene that is both efficient in terms of computation time and size, and accurate in terms of precision of the detections. For each detected person we want to detected the exact location of the head, in addition to to the locations of the shoulder keypoints. In order to achieve this goal, the scene is captured using a 3D camera that captures both the visual and depth information. This chapter will describe our system architecture and the methodologies used in our work.

This chapter is organized as follows. In Section 4.1, we give information about the dataset used for our work. We explain the concept of precision and recall in Section 4.2. Section 4.3 describes our adopted method for generating object proposals. Section 4.4 defines the CNN models we developed and tested for our system. We explain in Section 4.5 the training parameters that were used. Finally, Section 4.6 describes the work done for head detection using pre-exisitng methods.

Specification	Value
Depth range	0.6 - 8.0m (Optimal 0.6 - 5.0m)
Depth Image Size	640×480
RGB Image Size	640×480
Field of View	$60°horiz \times 49.5°vert.(73°diagonal)$
Power	USB 2.0
Software	Astra SDK or OpenNI 2 or 3rd Party SDK

TABLE 4.1: Orbbec Astra specifications

4.1 Dataset

The dataset used for our work was captured by the VIVA Research Lab. It was taken in 4 different indoor locations at a restaurant. Every captured frame has an RGB image and a corresponding depth image. The camera used is the Orbbec Astra[1], whose specifications are given in Table 4.1.

At every restaurant, the camera is placed in an elevated position that covers most of the customers waiting to be served. The Orbbec camera is connected to a Nvidia Jetson TX2[2] to capture the images. RGB images and the corresponding normalized depth images from the 4 locations are shown in Figures 4.1 and 4.2 respectively.

FIGURE 4.1: 4 sample color images from the dataset.

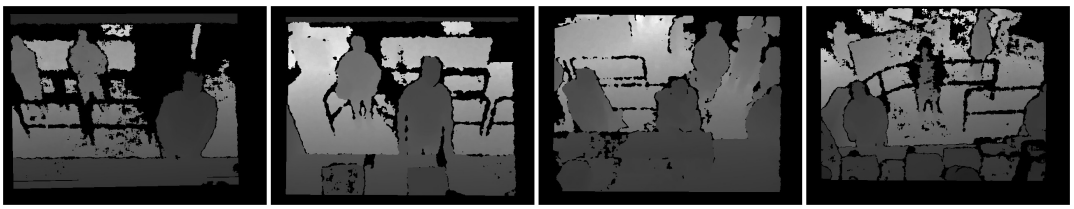

FIGURE 4.2: Corresponding normalized depth images from the dataset.

[1]https://orbbec3d.com/product-astra/
[2]https://developer.nvidia.com/embedded/buy/jetson-tx2

In total, we have 1610 images. 1449 images were randomly selected for training, and 162 were selected for testing. However, since the input to our models is 48×48 proposals, the number of samples used for testing is the total number of heads in all test images, and the same applies for training images. Hence, the total number of train samples is 3212 positive samples and 28921 negative samples, and the total number of test samples is 357 positive samples and 3361 negative samples. Data augmentation applied to increase the size of the positive training samples is discussed in Section 4.3.

4.2 Precision and Recall

To test the quality of our models in head detection, we use the precision and recall measures which are defined below:

- Recall is the total number of correctly detected heads over the total number of heads present in the testing set. It is a measurement of how good a model is in detecting all ground truth heads. Recall is calculated using Equation 4.1 below.

$$Recall = \frac{TP}{TP + FN} \tag{4.1}$$

  Where:

  - TP (true positives) represent the total number of correctly detected heads
  - FN (false negatives) represent the total number of missed heads

- Precision is a measurement of how accurate a model is in detection. It is used to know how many of our detections correspond to actual ground truth heads. Precision is calculated using Equation 4.2 below.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

  Where FP (false positives) represent the total number of detections that do not correspond to ground truth heads.

4.3 Object Proposals

While classification CNN do not require object proposals, any CNN model that works on object detection requires a method for generating object proposals, which are candidate locations in an image where an object *might* exist. For our work, computation and size requirements are of critical interest. For common object detection frameworks like Faster-RCNN [29] and SSD [28], proposals are generated after features have been extracted from an input image through a base CNN. Hence, the quality of the generated proposals relies heavily on the performance of the base network. A proposal generator should ideally generate proposals that include all objects of interest in an image. As the generated proposals are classified by the detector in later stages, it is common for the generated proposals to contain a lot of false positives. The ideal proposal generator would produce comprehensive proposals from an image while keeping the number of false positives to a minimum.

Usually, deeper and more sophisticated CNN extract better features. However, this comes with a heftier computation cost as the more parameters a CNN has, the more size and computation power it requires. The CHL algorithm introduced in [27] introduces a fast and efficient way for generating object proposals from depth images. We have decided to use the CHL algorithm [27] to generate proposals as a pre-processing step in our system architecture. That is, proposals will be extracted from depth frames using the CHL algorithm, and those proposals will be input to a small CNN for head and shoulder detection. Using the CHL algorithm for object proposals allows us to design smaller, more efficient feature extractors. The pseudocode for the CHL algorithm is given in Algorithm 1.

```
Regions = empty list;
Proposals = empty list;
for j in Image rows do
    for i in Image columns do
        if [j, i] has not been processed yet then
            region = new connected region;
            queue = create new empty queue;
            add [j, i] to region;
            add [j, i] to queue;
            while there are elements not processed in queue do
                m,n = current queue element column value, row value;
                if 3D distance of Image[m,n] and Image[m, n − 1] < 150 then
                    add [m, n − 1] to region;
                    add [m, n − 1] to queue;
                end
                if 3D distance of Image[m,n] and Image[m, n + 1] < 150 then
                    add [m, n + 1] to region;
                    add [m, n + 1] to queue;
                end
                if 3D distance of Image[m,n] and Image[m + 1, n] < 150 then
                    add [m + 1, n] to region;
                    add [m + 1, n] to queue;
                end
            end
            add region to Regions
        end
    end
end
for region in Regions do
    if number of pixels in region > 400 then
        add coordinates of top point of region to Proposals
    end
end
```

Algorithm 1: The CHL algorithm pseudo-code

The CHL algorithm processes a depth image to generate candidate head-top locations. The image is scanned from the top left corner to the bottom right corner. The idea is that pixels that belong to the same object would have continuous depth values and vary in a specific range. Each pixel is projected to the 3D space and the euclidean distance between the pixel and its neighbouring pixels is calculated. If the distance is within a certain range, the connected region is expanded and the neighbours of the newly added pixel are processed next to check if they also belong to the same object. This continues until no more pixels are found that belong to this object. The algorithm then moves to the next pixel in the queue to start searching for a new object until no more pixels are left unprocessed. This generates a number of connected regions. In order to reduce the number of proposals, connected regions that have a low number of pixels are discarded. In our implementation, regions that are composed of less than 400 pixels are discarded. The top point (having the lowest y coordinate) of every accepted region is chosen as the candidate head top location. After generating the candidate head top locations, bounding boxes should be generated from them in order to be fed to the CNN. In the original CHL algorithm, the bounding box was a rectangle of aspect ratio $3:4$. For our application, we found that the bounding box needed to have a bigger width in order to assure that it contains the shoulder keypoints. Hence, we slightly changed the bounding box generation such that it has equal width and height.

We have introduced a few changes to the CHL algorithm that allowed us to improve its speed by 3 times. The improvements are explained below:

- **Removing the pre-generation of the 3D point cloud:** The implementation of the CHL algorithm in [27] required two passes over all image pixels. In the first pass, the x, y, and $depth$ values of a pixel are used to generate a pointcloud. In the second pass, the 3D euclidean distance between neighbouring pixels in the generated pointcloud is calculated and connected regions are generated accordingly. In our implementation, we combine the processes of 3D euclidean distance calculation and connected regions generation. We only have to pass over image pixels once where we project the pixels being analyzed to the 3D

space and calculate their 3D euclidean distance that is used to generated connected regions. This enhancement reduces the computation time as the access to pixel values is happening once instead of twice. In addition, we do not need to store the generated pointcloud in memory.

- **Optimized pixel access:** Instead of the regular pixel access that was used in the original CHL algorithm for retrieving depth at a specific pixel, which requires specifying the x and y coordinates of the required pixel and the retrieval of the corresponding value from memory, we use pointer access to values to reduce that overhead.

In addition, we use Cython [42] to create a bridge between Python and C to remove the overhead caused by the dynamic binding in Python. The enhanced CHL algorithm functions at 115 frames per second. Also, the fact that the CHL algorithm operates entirely on the CPU makes it a very good option as the entire GPU memory would be reserved for the CNN.

To determine whether two pixels belong to the same object, the CHL algorithm relies on the eucledian distance between the two pixels in the 3D space. If the eucledian distance is less than a threshold T, the pixels are considered to belong to the same object. Having a too low or too high value of T reduces the recall of the algorithm. In this context, recall is referred to as the percentage of ground truth heads the CHL algorithm is successfully able to propose as possible head top locations. Also, increasing the value of T reduces the average number of proposals per image. Figure 4.4 shows the effect of increasing the value of T on the recall and average number of proposals per image of the CHL algorithm. It should be noted that as the value of T decreases, the number of connected regions increases and the size of those regions decreases. This leads to more regions being rejected. So there is a trade off that exists between lowering the value of T and the number of accepted regions. Similarly, increasing the value of T would decrease the number of regions and increase their size, which will also affect the recall. Based on the experimental results, we found that setting the value of T to $150mm$ gives the best recall (96.16) while maintaining a relatively low average number of proposals per image (19.88). Figure 4.3 shows

sample results of applying the CHL algorithm on our dataset where red circles denote candidate heat top locations output by the CHL algorithm.

FIGURE 4.3: Results of the CHL algorithm on our dataset. Red circles denote candidate head top locations.

The input to the CNN is the set of bounding box proposals. To generate the training data, we apply the CHL algorithm to our train images and extract the bounding boxes for each proposal. Every proposed bounding box that has intersection over union (IOU) greater than 40% with a ground truth head is considered a positive proposal, it is a negative proposal otherwise. From our dataset of 1449 train images, this gives us 3212 positive training samples and 28921 negative samples. Sample positive and negatives proposals are shown in Figures 4.5 and 4.6 respectively.

In order to increase the number of positive samples, we introduce augmentation to the positive samples by applying 5 rotations to each sample $(-10°, -5°, 0°, 5°, 10°)$. Figure 4.7 shows the augmentations applied to a positive sample. This increases the number of positive train samples to 16060.

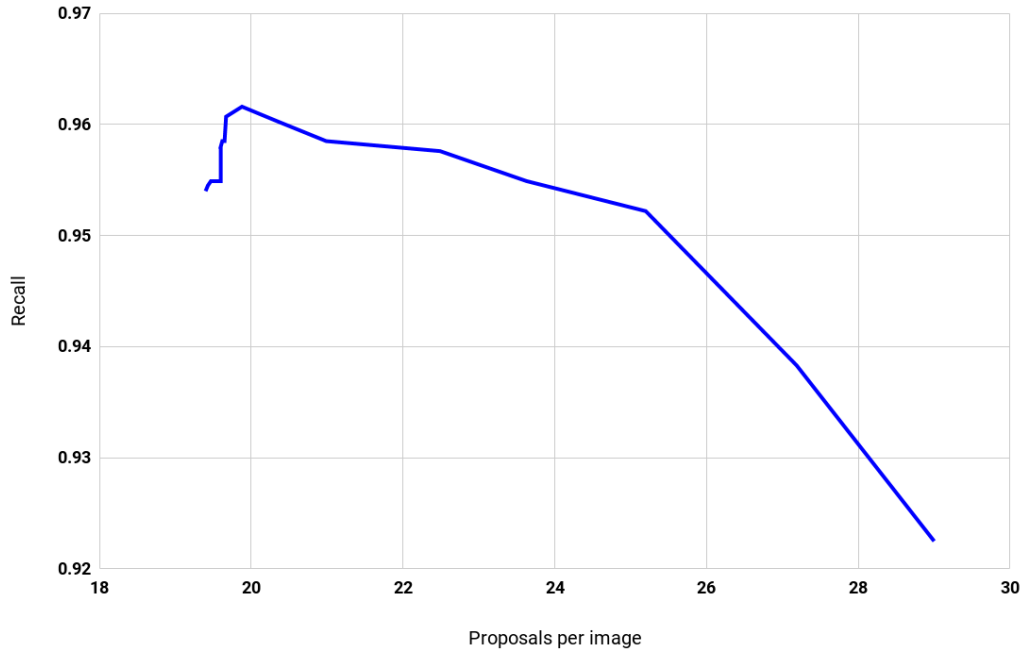

FIGURE 4.4: Recall against the average number of proposals per image as the value of T increases.

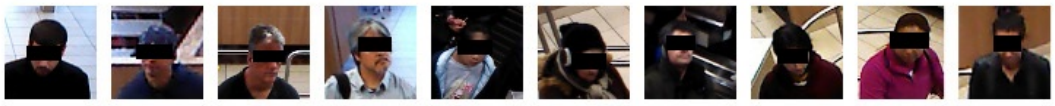

FIGURE 4.5: Sample train positive samples.

4.4 Feature Extractors

After generating the proposals, [27] use Alexnet [1] to classify proposals as heads and background where the input is a 3 channel representation generated from the depth image. For our work, we have decided not to use a pre-defined feature extractor. Feature extractors built for classification or as backbones to object detection frameworks do not have any information over the possible areas of interest in an image. The input, which is usually a 3 channel image, is processed through their layers to generate features that increase in quality as the depth and complexity of the network increases. This is why usually, the more sophisticated a CNN is, the better performance it has. Since we already have proposals, there is no need to use a complex, deep CNN.

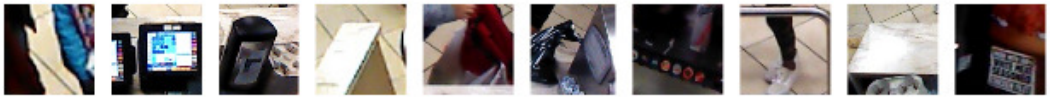

FIGURE 4.6: Sample train negative samples.

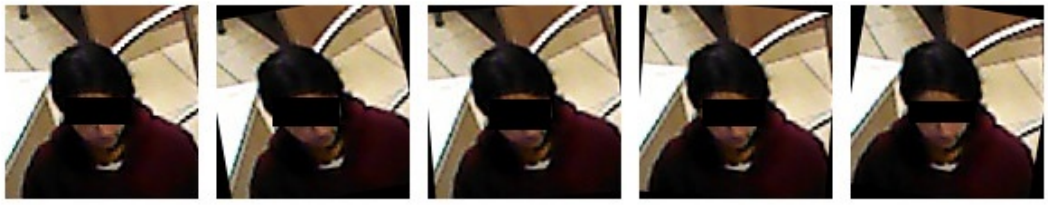

FIGURE 4.7: Augmentations applied to a positive sample.

Our feature extractor was inspired by the work done in [41], that was described in Section 3.5.2 of Chapter 3. They generate object proposals from an image using a small proposals CNN (PNet), then refine those proposals with a refinement CNN (RNet), and finally classify the refined proposals and generate detections using a small detection CNN (ONet).

Since we already have the proposals generated using the CHL algorithm [27], we do not need PNet and RNet, so we designed and trained three variations of CNN for our system and tested their performance on our dataset. The 3 architectures will be referred to in the remaining of this thesis as CNN4, CNN5, and CNN6 and their architectures are given in Tables 4.2, 4.3, and 4.4 respectively.

The 3 models share a similar architecture that is composed of convolutional and pooling layers. Relu is used as the activation function of these layers. The output of the last convolutional layer is flattened and input to a fully connected layer (FC1). 3 different fully connected layers (FC2, FC3, and FC4) are connected to FC1 and have the following functionalities:

- FC2: Has 2 outputs, used for binary classification of the input. It outputs 1 if the proposal is a person, and 0 otherwise.

Layer	**Dimensions**	**Kernel Size**	**Stride**	**Activation**
Image	$48 \times 48 \times 3$	-	-	-
Convolution	$46 \times 46 \times 32$	3	1	relu
Maxpool	$23 \times 23 \times 32$	3	2	relu
Convolution	$21 \times 21 \times 64$	3	1	relu
Maxpool	$10 \times 10 \times 64$	3	2	relu
Convolution	$8 \times 8 \times 64$	3	1	relu
Maxpool	$4 \times 4 \times 64$	2	2	relu
Convolution	$3 \times 3 \times 128$	2	1	relu
FC1	1×256	-	-	relu
FC2	1×2	-	-	Softmax
FC3	1×4	-	-	Softmax
FC4	1×4	-	-	Softmax

TABLE 4.2: Architecture of CNN4

Layer	**Dimensions**	**Kernel Size**	**Stride**	**Activation**
Image	$48 \times 48 \times 3$	-	-	-
Convolution	$46 \times 46 \times 32$	3	1	relu
Maxpool	$23 \times 23 \times 32$	3	2	relu
Convolution	$21 \times 21 \times 64$	3	1	relu
Maxpool	$10 \times 10 \times 64$	3	2	relu
Convolution	$8 \times 8 \times 64$	3	1	relu
Maxpool	$4 \times 4 \times 64$	2	2	relu
Convolution	$3 \times 3 \times 128$	2	1	relu
Convolution	$3 \times 3 \times 128$	2	1	relu
FC1	1×256	-	-	relu
FC2	1×2	-	-	Softmax
FC3	1×4	-	-	Softmax
FC4	1×4	-	-	Softmax

TABLE 4.3: Architecture of CNN5

Layer	Dimensions	Kernel Size	Stride	Activation
Image	$48 \times 48 \times 3$	-	-	-
Convolution	$46 \times 46 \times 32$	3	1	relu
Maxpool	$23 \times 23 \times 32$	3	2	relu
Convolution	$21 \times 21 \times 64$	3	1	relu
Maxpool	$10 \times 10 \times 64$	3	2	relu
Convolution	$8 \times 8 \times 64$	3	1	relu
Convolution	$8 \times 8 \times 64$	3	1	relu
Maxpool	$4 \times 4 \times 64$	2	2	relu
Convolution	$3 \times 3 \times 128$	2	1	relu
Convolution	$3 \times 3 \times 128$	2	1	relu
FC1	1×256	-	-	relu
FC2	1×2	-	-	Softmax
FC3	1×4	-	-	Softmax
FC4	1×4	-	-	Softmax

TABLE 4.4: Architecture of CNN6

- FC3: Has 4 outputs X_{min}, Y_{min}, X_{max}, Y_{max}, the bounding box coordinates of the head location in the proposal.

- FC4: Has 4 outputs X_{LS}, Y_{LS}, X_{RS}, Y_{RS}, the coordinates of the left and right shoulders of the person in the proposal.

The 3 layers have a softmax activation function that normalizes the outputs of each layer to a probability distribution between 0 and 1.

The difference between the 3 different CNN models is with the number of convolutional layers in different models. CNN4 has 4, CNN5 has 5, and CNN6 has 6 convolutional layers.

They key difference between our work and the work done in [27] is that we also predict the exact locations of the head inside a proposal, while the work in [27] only classifies the input as head or background. In addition, we predict the locations of the shoulder keypoints.

4.4.1 Head Detection

For the task of detecting the exact location of the head, we use a fully connected layer with softmax activation to predict the coordinates of the head bounding box. We have decided to adopt this approach instead of using one of the well known object detection frameworks like SSD [28] or Faster RCNN [29]. The main reason behind this decision is that we do not require our CNN to generate proposals for detection as proposals are generated through the CHL algorithm. Our work is closer to an RCNN approach where the region proposal network (RPN) is replaced with the CHL approach for proposals. In a similar approach to SSD, we also perform bounding box regression. The reasons why we don't use SSD or Faster-RCNN to detect heads from proposals are given below:

- SSD [28] receives the processed feature map output from a feature extractor, and processes it over 6 layers. The SSD layers gradually decrease the feature map dimensions. At every layer, a set of default boxes of different aspect ratios are generated at every location of the feature map to detect objects. This reduction in dimensions of the detection layers, and the generation of the default boxes with different aspect ratios allow the detection of objects with different shapes and sizes, making SSD an excellent framework for object detection. However, this is not a good approach for our case. Our input image has small dimensions (48×48) with the last feature map having a width and height of 3×3. This makes the 6 layers of SSD that decrease in size not useful. In addition, we only have one possible head in an input. So generating default boxes would not be necessary.

- Faster RCNN [29] generates a feature map using its base network, creates anchor boxes at each location of this feature map, and then uses RPN to to refine the proposals and keep only the ones that have a high probability of representing an object. The refined proposals are then input to R-CNN that classifies the proposals and generates a bounding box for each proposal. For our work, we do not require the entire functionality of RPN or the region of interest pooling.

4.4.2 Shoulder Detection

For detecting the shoulder locations of a person, we use a fully connected layer with softmax activation that outputs 4 values representing the x and y coordinates of the shoulders. It was inspired by the work done in [41] that was described in Section 3.5.2 of Chapter 3. However, since we do not need to generate and refine proposals, we do not implement the proposal and refinement networks. Our model architecture is close to their detection network (ONet). A complete overview of our system is given in Figure 4.8. Our input image is also of size 48×48, and the same principle is applied in the fully connected layers. We have a fully connected layer for classification, another for bounding box regression, and a third for shoulder keypoint detection. The difference is with the structure of the convolutional layers as we experimented with different architectures as shown in Tables 4.2, 4.3, and 4.4. The other parameters that we used in our system for training are explained in Section 4.5 below.

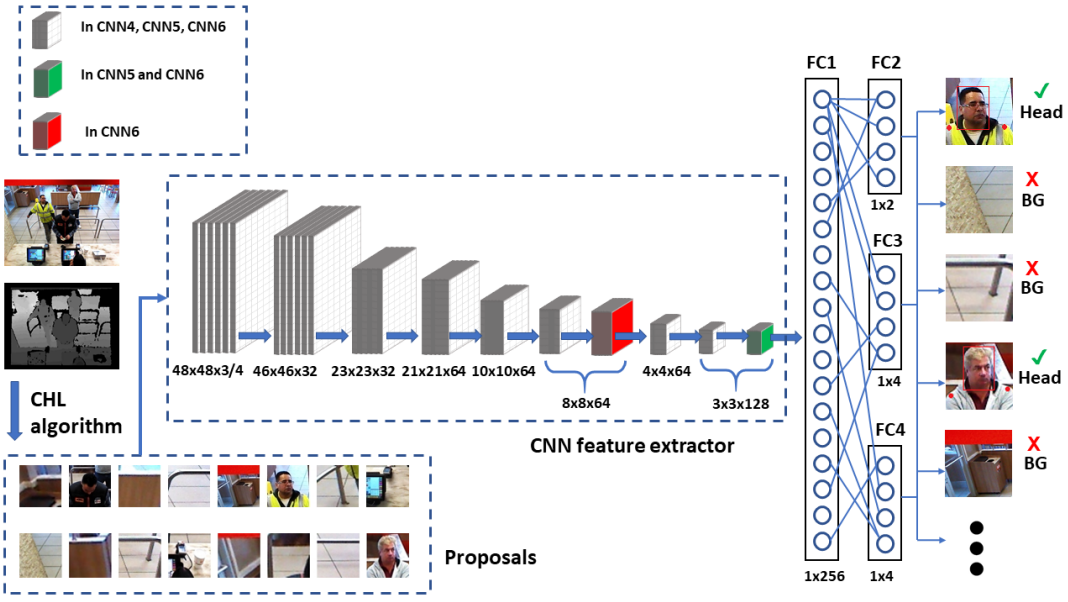

FIGURE 4.8: Complete overview of our system. Parameters of the convolutional layers are given in Tables 4.2, 4.3, and 4.4

4.5 Training

We built a framework for our model using Tensorflow [43]. The training was carried out on a workstation with Nvidia GTX 1080Ti graphics card [3] that has a compute capability of 6.2.

We train our models on two types of input:

- 3 channel input of RGB images
- 4 channel input of RGB+D images to see if the inclusion of depth information enhances the performance of the network

In this section, we will introduce the different training parameters used.

4.5.1 Batch size

Batch size is the number of samples input to the network at each training step. The loss gradient would be calculated based on the predictions of all samples inside a batch. For our implementation, we train 2000 epochs using a batch size of 200. Since our proposals would contain more negative samples than positive ones, we set the number of negative samples in a batch to 133 ($\frac{4}{6} \times 200$), and the number of positive samples to 67 ($\frac{2}{6} \times 200$). This way, the loss of the network is influenced more by negative samples, reducing false positives.

4.5.2 Loss function

Our network has 3 outputs from 3 fully connected layers. FC2 classifies the input as head or not head while FC3 and FC4 output the predicted coordinates of the head bounding box and the shoulder keypoint location respectively. A loss function needs to be defined for each one of the 3 tasks. A loss function is a measurement of how *bad* the network is in predicting a certain output based on the ground truth. Gradient descent (explained in 4.5.4) works on minimizing a loss function to improve the performance of a network.

[3] https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/

We use binary cross-entropy [44] loss as the loss function of FC2 as we only have 2 classes (1 for head, 0 for background). Binary cross-entropy loss is calculated using Equation 4.3.

$$Binary_{loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i) \tag{4.3}$$

Where N is the batch size, y_i is the ground truth label, and $\hat{y}_i$ is the predicted label.

FC3 and FC4 predict coordinates defining the head bounding box and the locations of shoulder keypoints respectively. As the measure of the quality of the predictions of those layers depends on the distance between the predicted points and the ground truth ones, we use the mean squared error [30] loss that is calculated using Equation 4.4.

$$MSE_{loss} = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N} \tag{4.4}$$

Where N is the batch size, y_i is the ground truth value, and $\hat{y}$ is the predicted value.

In order to reduce overfitting, we add regularization [45] to our total loss. The concept of regularization is that weights with large values might make it harder for the model to generalize even though it fits the training data. Regularization adds a value to the total loss that penalizes for large weights. We use L2 regularization in our work that is added to the total loss. L2 is calculated using Equation 4.5.

$$L2 = \sum_{j=1}^{m} \left\| w_j \right\|^2 \cdot \frac{\lambda}{2N} \tag{4.5}$$

Where:

- m: the number of layers
- w_j: the weight matrix of the j^{th} layer

- N: the batch size
- λ: the regularization parameter (default 0.01)

The total loss of the network is thereby calculated using Equation 4.6.

$$Total_{loss} = Binary_{loss} + MSE_{head} + MSE_{shoulders} + L2_{loss} \tag{4.6}$$

4.5.3 Learning rate

In a neural network, the learning rate controls by how much the weights are adjusted based on the loss gradient. If we have a too high learning rate, the training time would be lower but we risk missing any local minimas to the loss. If we have a too low learning rate, the training time would increase, although we have a better chance of not missing a local minima of the loss. The following formula 4.7 illustrates the functionality of the learning rate where Lr represents the set learning rate.

$$Weight_{new} = Weight_{existing} - (Lr \times Gradient) \tag{4.7}$$

Learning rate decay is used to speed up the training process by slowly reducing the learning rate over time. At the beginning of the training process, it is feasible to have a high learning rate as the weights need to be rapidly changed. After a certain time the weights need to be *fine tuned* and their values shouldn't change significantly. At this stage, we reduce the learning rate to be able converge to a minimum loss.

For our implementation, we use an initial learning rate of 0.001, and we multiply the learning rate by 0.1 every 500 epochs. An epoch is one complete cycle in which all the training data was input to the CNN.

4.5.4 Gradient descent with momentum

Gradient descent [46] is the core concept that allows artificial neural networks to *learn*. It is the calculation of the total loss of a minibatch using a given function based on the predictions and the corresponding ground truths of each sample in

the minibatch. The back-propagation of each sample calculates the updates to the values of all weights in the neural network in order to improve the accuracy of the network in predicting this specific sample. The back-propagation of all samples in the minibatch are then averaged in order to calculate the final updates to the weights.

Momentum [47] is an algorithm usually used with gradient descent that helps in speeding up the convergence of the network. At each step, we take into consideration the gradients of the previous step when calculating the new weights. We define a parameter that accumulates a weighted average of the past gradient, then use this parameter when calculating the new weights. The equations below (4.8 and 4.9) define the calculation of the accumulator and its use in calculating the new weights.

$$Accumulator = (Accumulator_{old} \times Momentum) + gradient \tag{4.8}$$

$$Weight_{new} = Weight_{existing} - (Lr \times Accumulator) \tag{4.9}$$

Momentum allows the network to reach a minimum loss faster through minimizing significant oscillations in the loss in wrong directions. We use gradient descent with momentum in our work with a *Momentum* value of 0.9.

4.6 Detection using different pre-existing methods

We compare our work with [27] by generating the 3 channel representations from depth information (DMH standing for Depth map, multi-order depth template, and height difference map) and training mobilenet [2] for classification only. We do not train the network for bounding box regression or shoulder keypoint detection as the authors of [27] only perform classification on the proposals. Figures 4.9 and 4.10 show positive and negative DMH samples respectively.

We also test the performance of OpenPose [8] for the task of shoulder keypoint detection on our test images.

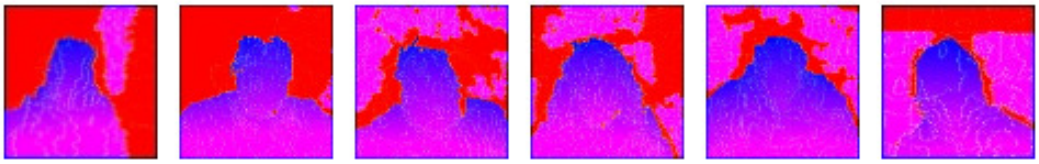

FIGURE 4.9: Sample DMH positive samples.

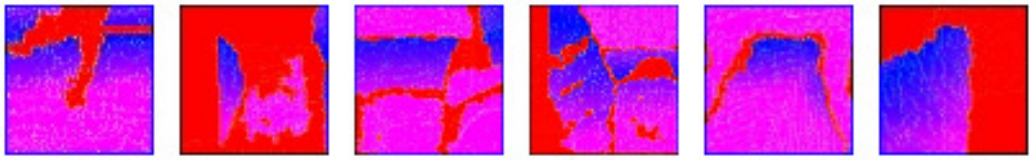

FIGURE 4.10: Sample DMH negative samples.

The DMH representation is composed of 3 channels:

- Depth map: the normalized raw depth data
- Multi-order depth template: encoding of the second-order gradients with the depth and gradient information based on 8 templates [27]
- Height difference map: encodes the height difference between the head top point and all other pixels in the proposal

In addition, we use the tensorflow object detection API [48] to train different models for object detection on our dataset. This API is an open source framework that simplifies the process of training different object detection models. It has implementations of SSD [28] and Faster RCNN [29], in addition to a wide variety of feature extractors that can be used as based models for the object detection frameworks.

We train the following models on our dataset:

- SSD with base network Inception V2
- SSD with base network Inception V3

- SSD with base network mobilenet

We chose those models based on the results given by the authors of the API given in [48] showing that those models provide the best mean average precision when used as features extractors of SSD.

Table 4.5 shows the different training configurations used to train models using SSD for detection.

Parameter	**Value**
Input dimensions	300×300
Number of SSD layers	6
Default boxes minimum scale	0.2
Default boxes maximum scale	0.95
Default boxes aspect rations	$0.33, 0.5, 1.0, 2.0, 3.0$
Batch size	16
Initial learning rate	0.004
Learning rate decay steps	30000
Momentum	0.9

TABLE 4.5: Important training parameters of SSD networks

4.7 Conclusion

In this chapter, we provided a comprehensive explanation of the dataset that we have experimented on in addition to the enhancements we did to the CHL [27] algorithm. We discussed the architecture of the different models we have built in addition to the training parameters we have employed. We also discuss the different pre-existing methods that we implemented to compare our work with. In the next chapter, we will provide the experimentation results and thoroughly analyse them.

Chapter 5

Experimentation and Results

This chapter elaborates our experimental results for the tasks of head and shoulder detection. We explain the measures we used for evaluating different models and elaborate on the testing configurations. We provide a comprehensive analysis of the results and comparison between the different tested models.

This chapter is organized as follows. In Section 5.1, we explain the experimentation results of head detection precision-recall of different models. Section 5.2 shows the experimentation results for shoulder precision. Sections 5.3 and 5.4 discuss the average IOU of our approach compared to that of [27] and the benchmarking experiments respectively. In Section 5.6, we show how our system tackles the problem of occlusion. Finally, we discuss the weaknesses of our method and failure cases in Section 5.5.

5.1 Head detection

When measuring the efficiency of a method in object detection, it is important to take both precision and recall into consideration. Having a high recall means our method is really good in detecting a certain object, but we need to know the associated precision to know how many false positives we are predicting when we have this high recall. It can be seen that there is a trade off between a model recall and precision. To generate the precision-recall curve, we calculate the precision and recall of different models on different confidence thresholds ranging from 0.05 to 0.9. As we increase the confidence of a model, we are decreasing the recall but increasing the precision, and vice versa.

In addition, we need to define a rule to follow in order to classify a detection as either a true positive or a false positive. Intersection over union (IOU) is typically used to measure the quality of a detection. It is defined as the ratio of the overlapping area of a detected box with the ground truth box and the union of the areas of both boxes.

In our experimentation, we calculate two precision-recall curves:

- Precision recall curve of our models compared to some state of the art methods for object detection. In those experiments, we set an IOU threshold of 0.5 in order to consider a detection to be correct. That is, if our regressed detected bounding box has more than 0.5 IOU with the ground truth box, the detection is considered a true positive. Otherwise, it is considered a false positive. Those results are shown in Figure 5.1.

- Precision recall curve of our models compared to mobilenet trained on the DMH representation proposed in [27]. In those experiments, the proposal box is evaluated instead of the regressed detection that is output by our models. Those experiments are necessary to compare with the work done in [27] that does not regress the head bounding box. It only classifies a proposed box as either a head or background based on a confidence threshold. In addition, Figure 5.5 shows that proposal boxes have a relatively low average IOU with the ground truth (0.28). Hence, we disregard the IOU of the proposal box with the ground truth similar to what the authors of [27] do. If a proposal box classified as a head contains a ground truth head, the proposal box is considered a true positive, regardless of the IOU value. It is considered a false positive otherwise. A proposal box is considered to contain a ground truth if more than 50% of the ground truth box is contained in the proposal box. The results of those experiments are shown in Figures 5.2 and 5.3.

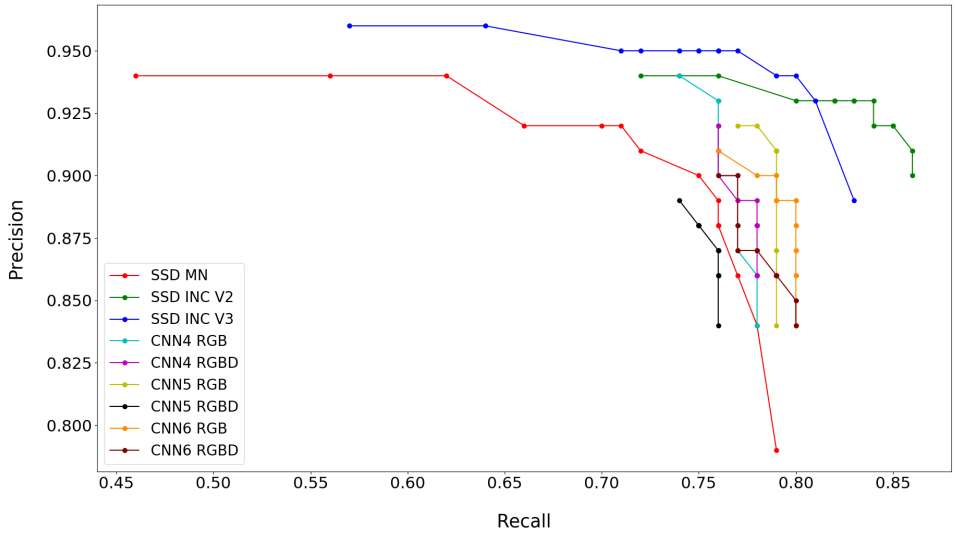

FIGURE 5.1: Precision-Recall curve applying 0.5 IOU criterion using confidence thresholds between 0.05 and 0.9

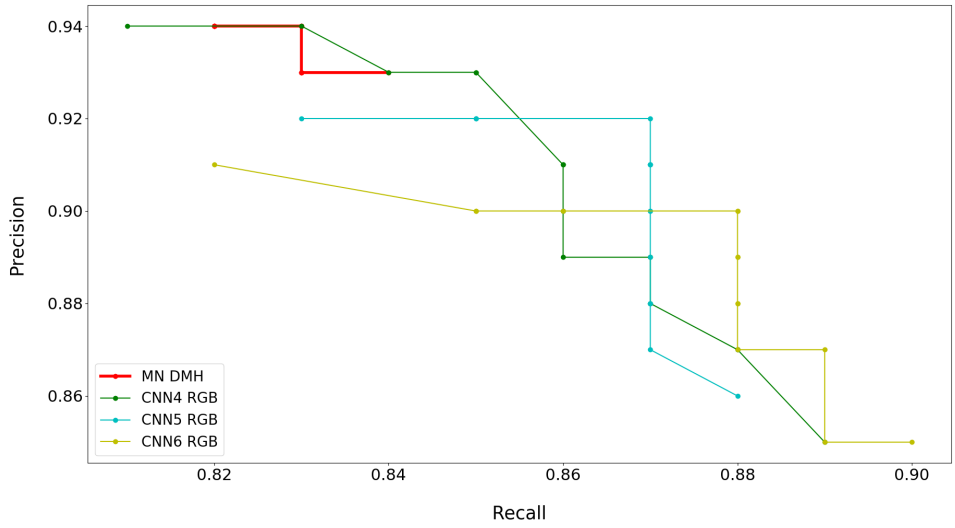

FIGURE 5.2: Precision-Recall curve using the ground-truth inclusion criterion using confidence thresholds between 0.05 and 0.9 for our RGB models

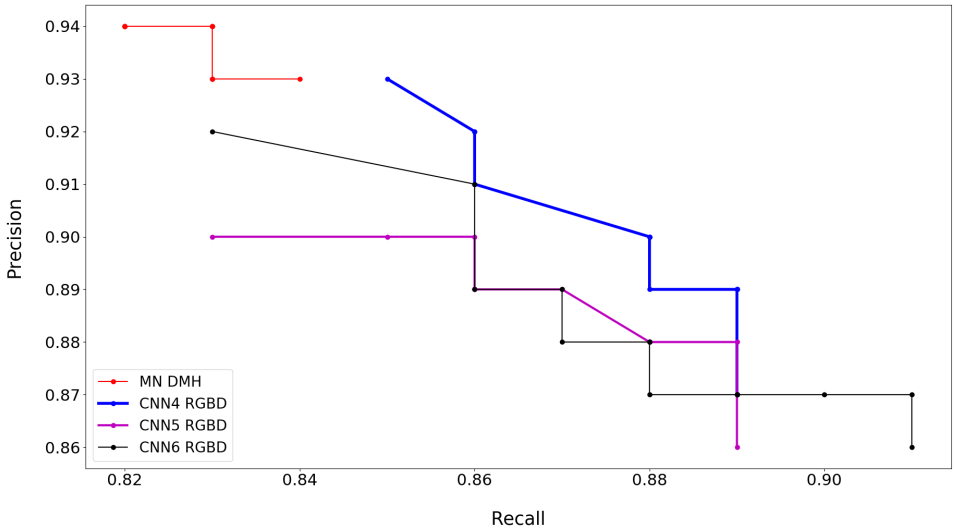

FIGURE 5.3: Precision-Recall curve using the ground-truth inclusion criterion using confidence thresholds between 0.05 and 0.9 for our RGBD models

Figure 5.1 shows that the best method in terms of precision-recall is SSD-Inception-V3 followed by SSD-Inception-V2. This was expected as the inception networks are among the best feature extractors and they work really well with SSD. However, this impressive performance comes at a hefty computational cost. This is elaborated in Section 5.4. We can see that our models perform relatively better than SSD mobilenet, which is the fastest feature extractor usually used as a backbone for SSD.

The figure shows that our models that receive a RGB input perform better than the models that operate on RGBD input. This can be explained by the depth channel adding confusion to the head classification task. The depth channel gives information that allows the differentiation of objects from background due to the distance disparity between an object and the background. While this differentiation proves to be beneficial for the precision of shoulder detection (discussed in Section 5.2), it adds confusion to the classification task as depth does not give information that helps in differentiating a human head from any other object.

We can also see that deeper models perform better. CNN6 achieves the best recall (0.8) at the same precision (0.88) of CNN5 and CNN4 (recalls of 0.79 and 0.77 respectively). This is also an expected result as deeper models extract better features that lead to better classification and bounding box detection accuracy. We see that the performance gain from adding more convolutions is not very significant, which allows us to use the smaller models while still having an acceptable performance in terms of precision-recall.

Figures 5.2 and 5.3 respectively compare the performance of our RGB and RGBD models with mobilenet trained on the 3 channel representation proposed in [27] (DMH). Our experimentation results show that [27] achieves high precision, but its recall is lower than the other compared models. This can be explained by the fact that DMH does not give a representation of the human head and shoulders region that is good enough to be distinguishable from other proposed objects. While it is impressive that the DMH representation is capable of defining the head and shoulders region through depth information alone, this representation is not as powerful

as the RGB representation.

The results also solidify the analysis done on Figure 5.1 as we can see that adding the depth channel adds confusion to the classification and leads to more false positives. In addition, it is also seen that deeper models perform better as CNN6 achieves the best recall (0.88) at the same precision (0.9) of CNN5 and CNN4 (recalls of 0.87 and 0.86 respectively).

5.2 Shoulder detection

For measuring the precision of our shoulder detections, we use the PCKh measure proposed in [9] which is inspired by the PCK measure proposed in [49]. PCK stands for *"Probability of Correct Keypoints"*. The PCKh measure (h stands for head) determines a keypoint to be correctly predicted if the euclidean distance between the predicted and the ground truth points is less than a specific value, denoted as d calculated according to formula 5.1 below:

$$d = \alpha \times max(h, w) \tag{5.1}$$

Where:

- α: a threshold that has a value between 0 and 1
- h: height of the head bounding box
- w: width of the head bounding box

In the original implementation of PCKh [9], α is set to 0.5. In our implementation, we calculate the precision of shoulder detection of our different models and openpose using different α values ranging between 0.1 and 0.9. For each value of α, the percentage of shoulder detections satisfying the PCKh equation given in Equation 5.1 is calculated. The results of calculating the PCKh precision using different values of α on our testing images are shown in Figure 5.4.

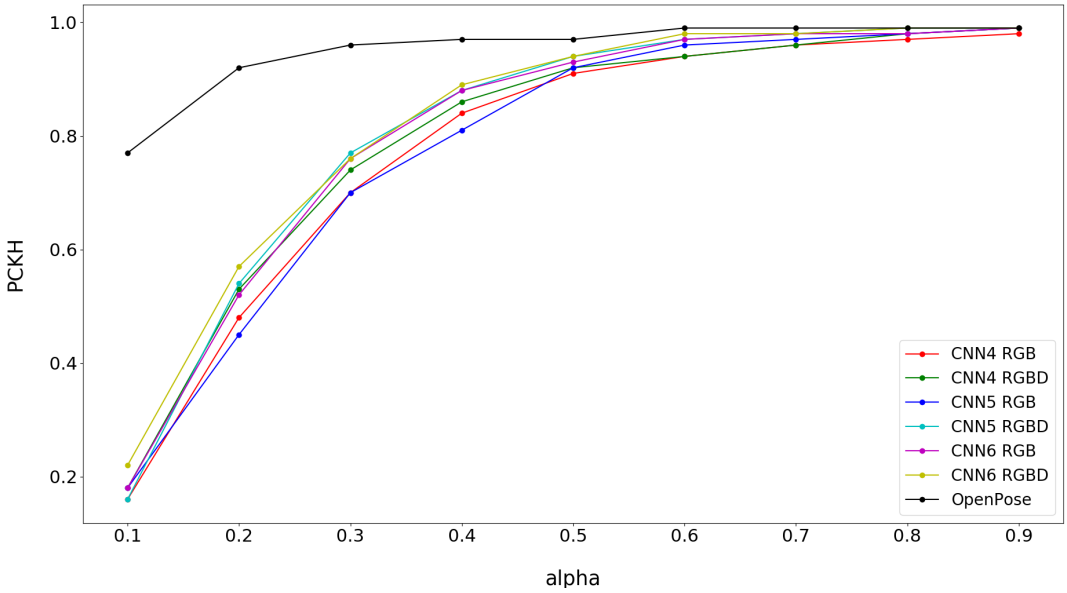

FIGURE 5.4: PCKh of shoulder detection of different tested models

The results in Figure 5.4 show that OpenPose is by far the best model in terms of shoulder detection precision. However, this excellent performance comes at a heavy computational cost (discussed in Section 5.4). The results clearly show that adding the depth channel always enhances the accuracy of shoulder detection. This can be seen through comparing models trained on RGB with the same models trained on RGBD. This can be explained by the distance disparity between an object and its background that is exploited by the depth channel. This disparity makes it easier for the model to correctly predict the keypoint locations of a proposal that has been classified as a head.

Figure 5.4 also shows that deeper models, in general, perform better in terms of shoulder precision. This is also expected as deeper models extract better features and are more efficient in keypoint detection.

5.3 Bounding box IOU

Our approach includes the generation of the bounding box of the exact location of the head within a proposal. This allows us to have a higher IOU with the ground

truth bounding box. Figure 5.5 shows the IOU of detected bounding boxes with the ground truth of different models. It is clear that our approach enhances the IOU of detections by more than twice. This is because the work done in [27] does not attempt to regress the bounding box location. The IOU is calculated based on the overlap between the proposal and the ground truth, while for our methods the IOU is calculated from the overlap of the detected bounding box and the corresponding ground truth. Figure 5.6 shows sample detection results from using CNN4 trained on RGB images.

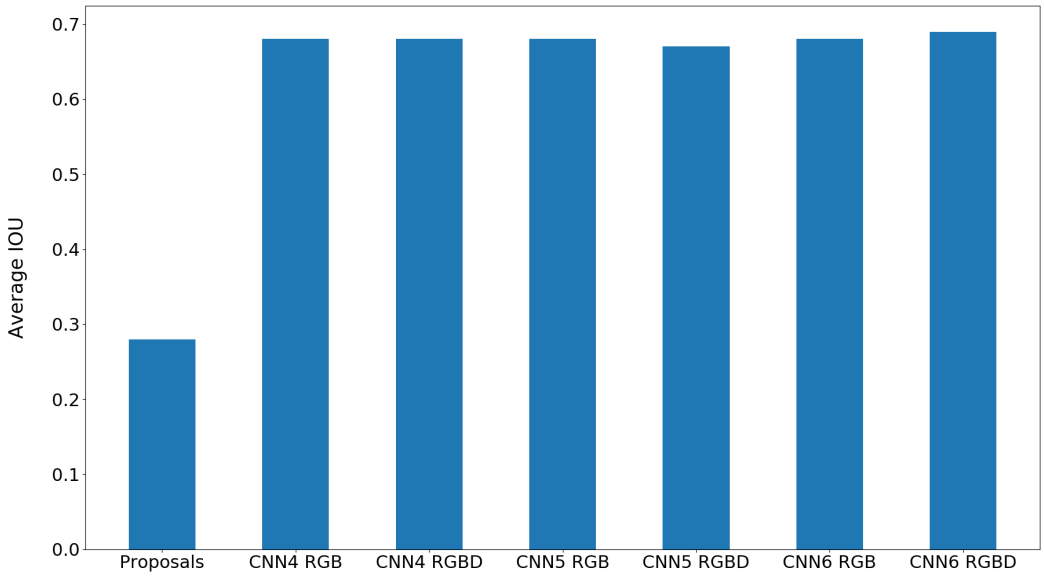

FIGURE 5.5: IOU of detected bounding box with the ground truth of different models

5.4 Computational cost

We have seen in the previous two sections that deeper and more complex models perform better in both precision-recall and shoulder precision. The main reason is that deeper models generate better features that allow them to perform better. However, this obviously comes at a heavier computational cost. As the size of the model increases, its computational cost will increase due to the increased number of computations to be performed.

FIGURE 5.6: 4 sample detection results using CNN4 RGB. Green represents ground truth data, red rectangles are regressed head detections, blue rectangles are proposal boxes, and yellow circles are predicted shoulder keypoint locations.

GPU	NVIDIA GeForce GTX 1080Ti
CPU	Intel Core i7-7820X 3.6 GHz, 8MB L2 cache
Memory	32 GB DDR4
Storage	2 TB
USB type	3.0
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN
OS	Ubuntu 16.04

TABLE 5.1: Razor workstation Specifications

GPU	NVIDIA Pascal, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM A57/2 MB L2
Memory	8 GB 128 bit LPDDR4
Storage	32 GB
USB type	USB 3.0
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN
Dimensions	50 mm x 87 mm
OS	Ubuntu 16.04

TABLE 5.2: Jetson TX2 Specifications

To evaluate the computational cost of the different tested models, we benchmarked them on two platforms:

- Razor workstation: An advanced workstation with powerful specifications
- Jetson TX2: An embedded system platform developed by NVIDIA [1]. Since we are building a model that should ideally run on embedded systems with low power consumption, we tested the models using the Jetson TX2.

The specifications of the Razor workstation and the Jetson TX2 are given in Tables 5.1 and 5.2 respectively. Inference benchmarking (in milliseconds) of the different models on the GPU and CPU of the Razor workstation and the Jetson TX2 are shown in Tables 5.3 and 5.4 respectively.

The columns in the benchmarking resemble the following:

- Proposals: The time for generating object proposals using the enhanced CHL algorithm from depth images.
- Inference: The inference time in milliseconds of the different models without any pre or post processing.

[1] https://developer.nvidia.com/embedded/buy/jetson-tx2

	Proposals	Inference	Extra	Total time (ms)	FPS
CNN4 RGB	8.7	3.1	0	11.8	**84.75**
CNN4 RGBD	8.7	3.1	0	11.8	**84.75**
CNN5 RGB	8.7	3.1	0	11.8	**84.75**
CNN5 RGBD	8.7	3.3	0	12	83.33
CNN6 RGB	8.7	3.5	0	12.2	81.97
CNN6 RGBD	8.7	3.7	0	12.4	80.65
MN DMH	8.7	6.8	12	27.5	36.36
SSD MN	NA	26	0	26	38.46
SSD INC V2	NA	30	0	30	33.33
SSD INC V3	NA	39	0	39	25.64
OpenPose	NA	250	0	250	4

	Proposals	Inference	Extra	Total time (ms)	FPS
CNN4 RGB	8.7	110	0	118.7	**8.42**
CNN4 RGBD	8.7	110	0	118.7	**8.42**
CNN5 RGB	8.7	116	0	124.7	8.02
CNN5 RGBD	8.7	117	0	125.7	7.96
CNN6 RGB	8.7	127	0	135.7	7.37
CNN6 RGBD	8.7	127	0	135.7	7.37
MN DMH	8.7	286	12	306.7	3.26
SSD MN	NA	210	0	210	4.76
SSD INC V2	NA	370	0	370	2.7
SSD INC V3	NA	580	0	580	1.72

TABLE 5.3: Processing times of different models on Razor machine GPU (top) and CPU (bottom).

- Extra: Any extra processing to be made that is necessary for the functioning of a method. Mobilenet trained on the 3 channel DMH representation [27] requires extra time to generate the DMH channels. DMH generation was also optimized using Cython [42] to optimize it as much as possible.

- Total time: The total processing time of the method. The summation of the 3 previous values

- FPS: Method running frames per second calculated using the following formula 5.2:

$$FPS = \frac{1000}{TotalTime(ms)} \tag{5.2}$$

Table 5.3 shows that the time for proposal generation is constant, as the proposal generation step is independent of the inference step of a model. Proposal boxes are generated as a preprocessing step, generated proposals are input to different models to test the inference speed. The proposals generation algorithm runs at around 115

	Proposals	Inference	Extra	Total time (ms)	FPS
CNN4 RGB	44	30	0	74	**13.51**
CNN4 RGBD	44	31	0	75	13.33
CNN5 RGB	44	32	0	76	13.16
CNN5 RGBD	44	32	0	76	13.16
CNN6 RGB	44	33	0	77	12.99
CNN6 RGBD	44	33	0	77	12.99
MN DMH	44	50	50	144	6.94
SSD MN	NA	77	0	77	12.99
SSD INC V2	NA	98	0	98	10.2
SSD INC V3	NA	130	0	130	7.69
OpenPose	NA	5500	0	5500	0.18

	Proposals	Inference	Extra	Total time (ms)	FPS
CNN4 RGB	44	300	0	344	**2.91**
CNN4 RGBD	44	280	0	344	**2.91**
CNN5 RGB	44	310	0	354	2.82
CNN5 RGBD	44	290	0	334	2.99
CNN6 RGB	44	330	0	374	2.67
CNN6 RGBD	44	320	0	364	2.75
MN DMH	44	645	50	739	1.35
SSD MN	NA	500	0	500	2
SSD INC V2	NA	1190	0	1190	0.84
SSD INC V3	NA	1850	0	1840	0.54

TABLE 5.4: Processing times of different models on Jetson TX2 GPU (top) and CPU (bottom).

FPS. The processing time of the CHL algorithm is constant whether we're testing on the Razor GPU or CPU as it runs solely on the CPU. The proposals algorithm can't be parallelized as it functions on sequential analysis of pixels from the top left corner to the bottom right corner. However, this is not an entirely a negative effect as this keeps the system GPU free to only run inference.

The results show that the models that we developed outperform all other compared models by a comfortable margin on both GPU and CPU. On GPU, our slowest model CNN6 trained on RGBD data, runs at 80.65 FPS. It is more than two times faster than the closest competitor, SSD with base network mobilenet which runs at 38.46 FPS. On CPU, the analytical results are the same as CNN6 trained on RGBD runs at 7.37 FPS where the closest competitor SSD with base network mobilenet runs at 4.76 FPS. We can see that our models are more affected when running on the CPU compared to SSD Mobilenet. This is due to the fact that mobilenet is optimized to

train on low end devices.

We can see that deeper networks are more computationally expensive and have a lower frame rate. In addition, the type of input affects the processing time. Models trained on RGBD input are slower than the same models trained on RGB input. These two effects are expected as adding more convolutions increases the computational cost and affects speed. Adding an extra input channel also has the same effect of increasing the computational cost. However, the results show that the difference in processing times isn't significant between our different models (around 4 FPS between our fastest and slowest models on GPU and around $1FPS$ on CPU). This is primarily due to the small input size as all our proposals are resized to 48×48.

Mobilenet trained on the DMH representation [27] runs at 38.46 FPS on GPU and 4.76 FPS on CPU. The main overhead in this method comes from the generation of the DMH representation. Although the idea of relying solely on depth information for human head detection is interesting, the precision-recall performance of this approach (discussed in Section 5.1), and the added overhead from the generation of the DMH representation make this approach not feasible for reliably detecting humans in different environments. It should also be noted that relying solely on the depth information introduces problems in outdoor environments as off the shelf infra-red depth cameras function poorly when subject to sunlight. In addition, depth cameras usually have a range limitation. Most commonly used depth cameras today have a maximum range of around 7 meters, with the accuracy of the depth measurement degrading as the object is further from the sensor. More powerful depth sensors using LIDAR technology can be used, but they are much more expensive than regular depth cameras.

For object detection models running using SSD, mobilenet proves the best feature extractor to be used as backbone for SSD in terms of processing speed followed by inception V2 and inception V3. OpenPose runs at only 4 FPS on the razor GPU, which makes it unsuitable for embedded systems.

Table 5.4 shows the benchmarking results on the Jetson TX2. The analytical conclusions derived from benchmarking on the Razor machine are verified by the Jetson benchmarking. Namely, adding more convolutions or adding the depth channel to the input increases the computational cost and reduces the frame rate. Our methods are still faster than compared models. The interesting phenomenon is that SSD object detection models are not affected as much as our models, specially on the Jetson GPU. SSD with base network mobilenet runs at the same frame rate as our slowest model (CNN6 trained on RGBD input). This can be explained through analysing how SSD works. SSD generates proposals and classifies them in a single pass. In addition, the number of default boxes evaluated by SSD does not rely on the number of people in the image. For our methods, the batch size is not constant. The proposals generated from an image are batched and passed to our models for classification, bounding box regression, and keypoint detection. However, our smaller models (CNN4 and CNN5) are still faster than the fastest SSD method (SSD with base network mobilenet). In addition, it should be noted that the Jetson TX2 has modest CPU power, which significantly affects the generation of proposals. It should also be noted that while SSD models detect head locations only, our models detect head locations in addition to shoulder keypoints.

5.5 Drawbacks and failure cases

Analysing the results of our methods, we notice a significant drawback: bounding box regression and keypoint detection can't be separated from the proposal classification. As discussed in Section 4.4, all our models have 3 fully connected layers with softmax activations:

- FC2: classifies the proposal as head or background
- FC3: generates head bounding box coordinates
- FC4: predicts shoulder keypoint locations

We rely on the classification result of FC2 in order to generate the regressed bounding and predict the shoulder keypoints. Only when FC2 classifies the proposal as a positive are the head bounding box and shoulder keypoints generated. This raises two problems:

- In cases of false positives, both the head bounding box and the shoulder keypoints will be generated, where there is no human present.
- In cases of false negatives, neither the head bounding box nor the shoulder keypoints will be generated.

In addition, two shoulder keypoints will always be generated, even if a part of the person is not visible or occluded and only one shoulder is visible.

Finally, although our models can perform well when trained on RGB images only, the proposals algorithm operates solely on depth images. This makes it reliant on the quality of the depth images and adds range constraints. Not to forget that off the shelf depth sensors perform poorly in outdoor environments.

Figure 5.7 shows sample failure cases using CNN4 RGB.

5.6 Occlusion

We have previously discussed in Chapter 1 that occlusion is one of the most challenging issues to object and keypoint detection. Our apporach that utilizes the CHL algorithm for generating object proposals solves the problem of occlusion to a decent extent. Setting the distance threshold in order to consider two pixels to belong to the same object to 150mm means that our system would fail to propose the presence of an occluded person only if this person is within 15cm in 3D space from the person occluding him/her.

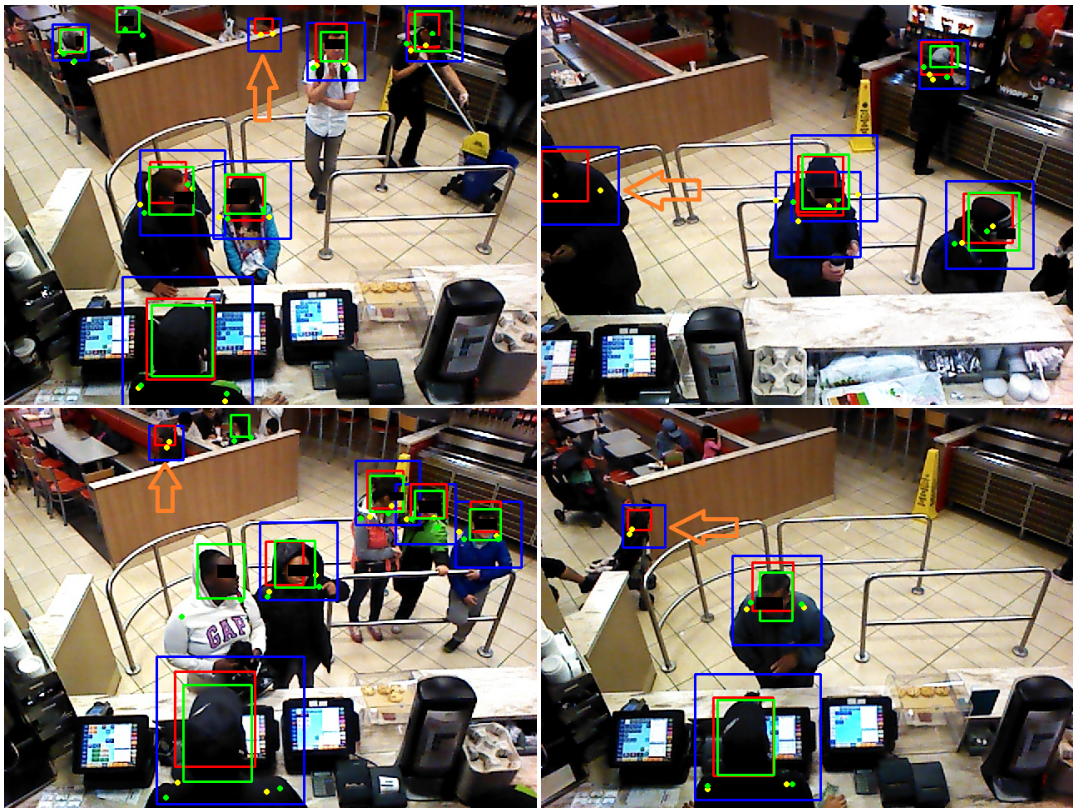

FIGURE 5.7: 4 sample failure cases using CNN4 RGB. Green represents ground truth data, red rectangles are regressed head detections, blue rectangles are proposal boxes, and yellow circles are predicted shoulder keypoint locations.

5.7 Conclusion

In this chapter, we elaborated our experimentation results for head detection, shoulder detection, and bounding box IOU. We provided a comprehensive analysis of the results of different models. We also studied the computational cost of all models on two platforms and highlighted the drawbacks of our approach. We concluded that our approach performs better than compared models in terms of speed/accuracy. We highlighted the effect of adding extra convolutions and adding the depth channel as an extra channel to the input. In the next chapter, we will summarize the work done in this thesis and the results obtained. We will also discuss future extensions and improvements to our approach.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have provided a comprehensive review of CNN and famous deep learning models for classification, head detection, and keypoint detection. We have introduced a new approach for detecting human heads and shoulders using a combination of image processing and deep learning techniques. We show the effect that utilizing depth information has on the precision of head and shoulder detection. In addition, we discussed three different architectures of our model and analyse the trade off between speed and accuracy of the different architectures. We compare our approach to several state of the art methods in head and shoulder detection.

The analysis of our experimentation results shows that in general, deeper CNN are able to perform better as they extract more sophisticated features. However, this comes with an increase in computational cost. We also show that adding depth as an extra channel to the input enhances the precision of shoulder detection due to the distance disparity between objects and background in depth data. However, the depth channel slightly affects head classification as it adds confusion since it does not provide strong distinctive information that helps in differentiating a human head from any other object. We show that our approach is able to detect heads with higher IOUs than other approaches [27] while still being smaller and faster.

6.2 Future Work

Our work utilizes traditional image processing techniques for generating proposals and a CNN to detect human heads and shoulder keypoints. In the future, our work can be expanded to detect more body parts to improve the pose estimation task. For instance, we can construct another pipeline to detect elbows and hands of a proposal that's been determined to be a human head by our current approach. A proposal box could be generated based on the detected shoulder keypoints that includes elbow and hand locations. The new proposal box would be input to another CNN trained to predict the locations of elbow and hand keypoint locations.

In addition, further enhancements to the object proposals algorithm can also be investigated to make it run faster on embedded devices. A possible solution would be to introduce multi-threading by dividing the images to a number of regions where each region runs the CHL algorithm on a separate thread. Optimizing the proposals algorithm so it can run on GPU might also be a future field of investigation.

Finally, I would like to say that although deep learning has allowed us to achieve state of the art performances in different machine vision problems, it should not be treated as a black box. I think it is imperative for new researches in machine vision to learn and understand traditional approaches in machine vision so they can understand the problems and challenges that deep learning has solved. In addition, deep learning algorithms still have lots of room for improvement and I think we are still far from reaching the ideal deep learning model.

Appendix A

Link of Scripts for all our experimentations

We provide below the links for the codes that were developed for the work performed in this thesis.

1. **CNN model architectures - Training and evaluation:** The scripts defining CNN4, CNN5, and CNN6, in addition to training and evaluation scripts can be found in the following link https://gitlab.com/wassimelahmar/chl_onet

2. **Different processing scripts:** Different scripts used in preprocessing of the data, generating the DMH representations and implementaion of the CHL algorithm in addition to other testing scripts can be downloaded from the following link: https://gitlab.com/wassimelahmar/chl_work

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint arXiv:1704.04861*, 2017.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[5] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm", in *Icml*, Citeseer, vol. 96, 1996, pp. 148–156.

[7] T. K. Ho, "Random decision forests", in *Document analysis and recognition, 1995., proceedings of the third international conference on*, IEEE, vol. 1, 1995, pp. 278–282.

[8] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7291–7299.

[9] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis", in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, 2014, pp. 3686–3693.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection", in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 886–893.

[11] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 8, pp. 1532–1545, 2014.

[12] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, IEEE, vol. 1, 2001, pp. I–I.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning", in *International Workshop on Artificial Neural Networks*, Springer, 1995, pp. 195–201.

[15] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition", in *International conference on artificial neural networks*, Springer, 2010, pp. 92–101.

[16] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biol. Cybernetic 36*, 1980.

[17] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *arXiv preprint arXiv:1207.0580*, 2012.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks", in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, IEEE, 2017, pp. 5987–5995.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[24] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning.", in *AAAI*, vol. 4, 2017, p. 12.

[25] L. Spinello and K. O. Arras, "People detection in rgb-d data", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2011, pp. 3838–3843.

[26] J. Liu, Y. Liu, Y. Cui, and Y. Q. Chen, "Real-time human detection and tracking in complex environments using single rgbd camera", in *2013 IEEE International Conference on Image Processing*, ieee, 2013, pp. 3088–3092.

[27] L. Tian, M. Li, Y. Hao, J. Liu, G. Zhang, and Y. Q. Chen, "Robust 3d human detection in complex environments with depth camera", *IEEE Transactions on Multimedia*, 2018.

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector", in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in *Advances in neural information processing systems*, 2015, pp. 91–99.

[30] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[31] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[32] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[33] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*.

[34] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition", *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[35] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation", *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[36] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges", in *European conference on computer vision*, Springer, 2014, pp. 391–405.

[37] L. Tian, G. Zhang, M. Li, J. Liu, and Y. Q. Chen, "Reliably detecting humans in crowded and dynamic environments using rgb-d camera", in *Multimedia and Expo (ICME), 2016 IEEE International Conference on*, IEEE, 2016, pp. 1–6.

[38] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik, "R-cnns for pose estimation and action detection", *arXiv preprint arXiv:1406.5212*, 2014.

[39] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, "Multiscale combinatorial grouping", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 328–335.

[40] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik, "Using k-poselets for detecting people and localizing their keypoints", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3582–3589.

[41] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks", *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[42] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, "Cython: The best of both worlds", *Computing in Science Engineering*, vol. 13, no. 2, pp. 31 –39, 2011, ISSN: 1521-9615.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[44] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method", *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[45] C. Cortes, M. Mohri, and A. Rostamizadeh, "L 2 regularization for learning kernels", in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, AUAI Press, 2009, pp. 109–116.

[46] L. Bottou, "Large-scale machine learning with stochastic gradient descent", in *Proceedings of COMPSTAT'2010*, Springer, 2010, pp. 177–186.

[47] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.", *ICML (3)*, vol. 28, no. 1139-1147, p. 5, 2013.

[48] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors", in *IEEE CVPR*, vol. 4, 2017.

[49] Y. Yang and D. Ramanan, "Articulated human detection with flexible mixtures of parts", *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2878–2890, 2013.