Aggregated Learning: An Information Theoretic Framework to Learning with Neural Networks

by

Masoumeh Soflaei Shahrbabak

Thesis submitted in partial fulfillment of the requirements
for the Ph.D. degree in
Electrical and Computer Engineering

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

Abstract

Deep learning techniques have achieved profound success in many challenging real-world applications, including image recognition, speech recognition, and machine translation. This success has increased the demand for developing deep neural networks and more effective learning approaches.

The aim of this thesis is to consider the problem of learning a neural network classifier and to propose a novel approach to solve this problem under the Information Bottleneck (IB) principle. Based on the IB principle, we associate with the classification problem a representation learning problem, which we call "IB learning". A careful investigation shows there is an unconventional quantization problem that is closely related to IB learning. We formulate this problem and call it "IB quantization". We show that IB learning is, in fact, equivalent to the IB quantization problem. The classical results in rate-distortion theory then suggest that IB learning can benefit from a vector quantization approach, namely, simultaneously learning the representations of multiple input objects. Such an approach assisted with some variational techniques, result in a novel learning framework that we call "Aggregated Learning (AgrLearn)", for classification with neural network models. In this framework, several objects are jointly classified by a single neural network. In other words, AgrLearn can simultaneously optimize against multiple data samples which is different from standard neural networks. In this learning framework, two classes are introduced, "deterministic AgrLearn (dAgrLearn)" and "probabilistic AgrLearn (pAgrLearn)".

We verify the effectiveness of this framework through extensive experiments on standard image recognition tasks. We show the performance of this framework over a real world natural language processing (NLP) task, sentiment analysis. We also compare the effectiveness of this framework with other available frameworks for the IB learning problem.

Acknowledgements

To attain knowledge add things every day. To attain wisdom remove things every day.

Tao Te Ching

I would like to express my gratitude to my supervisor Professor Yongyi Mao, for his guidance, patience, and caring. Without his support, encouragement, and especially his guidance, it would not have been possible to complete this work. After I read his article "life after Peter" I wished I could be Yongyi's student. Dream came through! Maybe I should write an article and call it "life after Yongyi". I cannot thank him enough for his support, guidance, patience.....

I would like to thank my committee members, Professor Xiaodan Zhu, Professor James R. Green, Professor Jochen Lang, and Professor Paula Branco for their time, attention to detail, and encouragement.

My sincere thanks go to the team at Beihang University for their help and support; especially Professor Richong Zhang, for allowing me to use Beihang resources, Zhiyuan Hu, and students at Beihang University for their help and support at all hours. Without their support it would not have been possible to accomplish the huge amount of simulations carried out during this short period of time. 衷心感谢大家对我的支持和帮助。 I would also like to thank Dr. Harry Guo for his guidance in the experimental part and Dr. Ali Al-Bashabsheh for his contributions in the proofs of the theorems in the paper.

I am so grateful to Dr. Baily Seshagiri who helped, encouraged, supported me, and generously proofread my work.

In life, there are always people who make success possible and rewarding. My lovely friends who stayed with me in happy and sad days during this journey. The ones who helped, and cajoled me when I needed it the most. The colleagues who were there always for me, especially Guillaume and Ziqiao. The ones who called me Big sister! The ones who

taught me Chinese! and Chenjie, I will never forget the taste of original green tea, thanks for that.

Many thanks to all the people who showed up in my life during this journey and taught me a lesson and guided me towards the right path! I truly appreciate your help.

Above all, my deepest gratitude goes to my lovely parents, who made all of this possible, for their endless encouragement and support, my success is yours. And many thanks to my sister and my brother for their encouragement, support, and always being there for me. Thank you now and always.

Dedication

This Ph.d. thesis is dedicated to my loving parents: my mother, Salma Bagheri and my father, Hassan Soflaei. Their support, encouragement, and constant love have sustained me throughout my life.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

AgrLearn	Aggregated Learning
BC Learning	Between Class Learning
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
dAgrLearn	Deterministic Aggregated Learning
DL	Deep Learning
ERM	Empirical Risk Minimization
FN	False Negative
FP	False Positive
IB	Information Bottleneck
KL	Kullback-Leibler
LLN	Law of Large Numbers
LSTM	Long Short-Term Memory
MAP	Maximum a Posterior
MI	Mutual Information
MINE	Mutual Information Neural Estimator
ML	Machine Learning
MLP	Multi-Layer Perceptron
MPQA	Multi-Perspective Question Answering
NLP	Natural Language Processing

pAgrLearn	Probabilistic Aggregated Learning
ReLU	Rectified Linear Unit
ResNet	Residual Network
RIR	Relative Improvement Rate
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVHN	Google Street View House Numbers
TN	True Negative
TP	True Positive
VGG	Visual Geometry Group
VIB	Deep Variational Information Bottleneck
WideResNet	Wide Residual Network

Chapter 1

Introduction

1.1 Overview, Motivation and Contributions

The revival of neural networks in the paradigm of deep learning [52] has stimulated intense interest in understanding the networking of deep neural networks, e.g., [69, 84]. Among various efforts, an information-theoretic approach known as Information Bottleneck (IB) [74] stands out as a fundamental tool to theorize the learning of deep neural networks [15, 65, 69].

The IB principle was introduced by Tishby et al. [74]. They introduced a variable T that is defined as a compressed representation of object X. The IB principle was proposed to extract or represent only relevant information instead of extracting all of the information. Therefore the main goal in this principle is to minimize the amount of information about object X while preserving, as much as possible, the information about class label Y. The conflict between these two requirements can be formulated as a constrained optimization problem in which one requirement is implemented as the objective function and another requirement as the constraint [74, 64, 66]. In this thesis, we call the problem of learning representation T, *IB learning*.

Initial attempts to solve the IB learning problem were based on assuming known distributions for object X and class label Y. In [66] the IB learning problem with unknown

distributions for discrete random variables was investigated. But, the proposed formulation in [66] was not applicable to real world problems. As a result, Alemi et al. [4] proposed using unknown distributions with continuous random variables. They introduced a framework known as Deep Variational Information Bottleneck (VIB). Based on the VIB framework, a lower bound was constructed for the IB learning problem, and the performance of this framework was experimentally established. Subsequently, different techniques were proposed to estimate the lower bound established by VIB [1, 2, 23]. More recently, methods have been proposed for a general formulation of the IB learning problem [47, 48].

In this work, we investigate the IB learning problem to solve real world problems using perspectives from rate-distortion theory and the information theory for quantizing signals. The key observation that has inspired this work is that the optimization formulation of IB learning resembles greatly the rate-distortion function in rate-distortion theory [67]. A careful examination of this observation indeed reveals that, conceptually, there is an unconventional quantization problem that is closely related to IB learning. To that end, we formulate this problem, which we refer to as *IB quantization*. We prove that the objective of IB quantization, namely, designing quantizers that achieve the rate-distortion limit, is equivalent to the objective of IB learning. This result establishes an equivalence between the two problems.

In rate-distortion theory, it is well known that scalar quantizers, which quantize signals one at a time, are in general inferior to vector quantizers, which quantize multiple signals simultaneously. The discovered equivalence between IB learning and IB quantization then suggests that IB learning may benefit from a "vector quantization" approach, in which the representations of multiple inputs are learned jointly. Exploiting variational techniques, we show that such a vector quantization approach to IB learning naturally results in a novel framework for learning neural network classifiers. We call this framework *Aggregated Learning (AgrLearn)*.

In AgrLearn framework, n random training objects are aggregated into a single amalgamated object and passed to the model; the model predicts the soft labels for all n examples jointly. The training of the AgrLearn framework is carried out by solving an

optimization problem, derived from a variational relaxation of the IB learning problem and an approximation of mutual information.

In this learning framework, based on the type of mapping from object X to variable T, two classes are introduced. Deterministic mapping results in "deterministic AgrLearn (dAgrLearn)", and probabilistic mapping results in "probabilistic AgrLearn (pAgrLearn)".

We conduct experiments, applying AgrLearn framework to the current art of deep learning architectures for image recognition to show the significant gain that the framework brings in classification accuracy. Moreover, we show the robustness of this framework when dealing with the problem of data scarcity, and also how it is compatible with regularization techniques such as MixUp [85]. In addition to this, we use this framework to solve a real world problem called sentiment analysis.

Our main contributions to the field of IB Learning and Deep Neural Networks, as shown in this thesis, are as follows:

We have:

1. Established an equivalency between the IB learning problem and the IB quantization problem, and formulated the IB quantization problem.

2. Proposed a novel neural network learning framework, which we call *Aggregated Learning (AgrLearn)*. This framework is inspired by vector quantization. Generally, vector quantizers by providing a richer family of encoders and decoders perform better than scalar quantizers. This was a motivation for us to use this quantization technique for solving IB learning problem instead of using standard neural networks which are equivalent to scalar quantizers.

3. Established two classes of the AgrLearn framework: *deterministic AgrLearn (dAgrLearn)* and *probabilistic AgrLearn (pAgrLearn)*.

4. Established empirically that the AgrLearn framework improves the classification performance of recent deep models. This investigation is done on image recognition. In addition, we have investigated the performance of the AgrLearn framework in comparison with the VIB framework.

5. Established empirically how well the AgrLearn framework can perform when there is data scarcity in neural network learning. Finally, we investigate the performance of the AgrLearn framework by adding regularization techniques such as MixUp.

6. Established empirically the performance of the AgrLearn framework on a real world natural language processing task, sentiment analysis.

1.2 Outline

This thesis is organized as follows:

Chapter 2 presents a review of machine learning algorithms, models, and concepts that are used in this thesis. Information bottleneck principle and a literature review of solutions to IB problem are presented in Chapter 3. Chapter 4 illustrates the equivalence between the IB learning problem and the IB quantization problem, theories, and their proofs are provided in this chapter. After establishing the equivalence between IB learning and IB quantization, in Chapter 5 a variational approach to the IB learning problem is demonstrated. Using the variational approach, the novel neural network learning framework, AgrLearn, is introduced in this chapter which is the main contribution of this thesis. Two classes of the AgrLearn framework that we call deterministic AgrLearn (dAgrLearn) and probabilistic AgrLearn (pAgrLearn) are introduced in this chapter. In Chapter 6 we investigate the application of AgrLearn framework to image recognition and natural language processing, specifically sentiment analysis. Finally, the conclusions reached in this thesis as well as suggestions for future research are presented in Chapter 7.

1.3 Notation

Throughout this study, vector notations are denoted by lowercase letters, and matrix notations by uppercase, boldface letters. A vector x has elements $(x[1], x[2], \cdots, x[m])$.

A capitalized letter denotes a random variable, e.g., X, and a value it may take is shown

by its lowercase. The distribution of a random variable is showing by p with a subscript indicating the random variable, e.g., $p_X(x)$. The symbol $\mathbb{E}$ denotes expectation.

1.4 List of Publications

- Journal Papers

    1. M. Soflaei, H. Guo, A. Al-Bashabsheh, Y. Mao, R. Zhang, "Novel Approach to Learning with Neural Networks by Using Vector-Quantization", in preparation to submit to IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).

- Conference Papers

    1. M. Soflaei, H. Guo, A. Al-Bashabsheh, Y. Mao, R. Zhang, "Aggregated Learning: A Vector-Quantization Approach to Learning Neural Network Classifiers", AAAI Conference on Artificial Intelligence (AAAI-2020), New York, 2020.

Chapter 2

Background

2.1 Machine Learning

Machine learning (ML) is a sub-field of Artificial Intelligence (AI). ML is defined as a computer program to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E [56]. In this chapter, we review some algorithms, models, and concepts in ML which are used in the entire thesis.

2.1.1 Basic Forms of Machine Learning Algorithms

ML algorithms are often classified into three major categories, based on the nature of learning: supervised learning, unsupervised learning, and reinforcement learning.

In *supervised learning*, the example inputs that are labeled with their desired outputs are provided to the system. The algorithm will learn by comparing the actual output with the desired ones to find the error and modifies the model accordingly. Then the system can predict the output for new (unseen) inputs.

The training dataset in supervised learning contains pairs of input and desired output. In other words, for each input, there is a desired output. Considering the input variable X

in space $\mathcal{X}$ and the desired output variable Y in space $\mathcal{Y}$, the dataset can be defined as:

$$\mathcal{D} := \{(X_1, Y_1), \cdots, (X_N, Y_N)\}$$

and the goal of supervised learning is to approximate the mapping function $F : \mathcal{X} \to \mathcal{Y}$ so as to predict the output variable Y for new input variable X. Therefore, the supervised learning model is defined as:

$$Y := F(X; \theta) \tag{2.1}$$

where θ represents the set of model parameters. Classification and regression are two examples of this type of ML.

- In the classification method inputs are divided into two or more classes and the model needs to be trained in such a way that it can assign the new (unseen) inputs to one of the classes. For X and Y as the observed samples and class labels respectively, the objective of classification is to develop a classifier, which assigns a class label Y for the new (unseen) samples. In other words, the classification method consists of two steps: first building a model based on the available objects and class labels, then using the trained model to predict labels for the unseen objects with the model.

- In the regression method as well as in the classification method, the objective of learning is to approximate the mapping function which maps input variable X to output variable Y, as accurately as possible such that it can predict the output variable for the new (unseen) input data. The only difference between these two methods is that the output variable in the regression is a numerical variable whereas in the classification method it is a categorical variable.

In *unsupervised learning*, the input data is provided without their corresponding labels. The goal is to model the underlying structure or distribution in the data in order to learn more about the data. Clustering, and density estimation are two examples of unsupervised learning.

- In clustering, data is divided into a number of groups in which the data within a group are more similar to each other than to data in other groups. In other words,

data within each group are very similar to each other, while they are different from data in other groups. K-means algorithm [6] is an example of clustering.

Reinforcement learning is based on rewarding and penalizing. These algorithms are designed to reward certain actions and penalize others. The machine is then able to make the best decision and continually learns from past failures. The agents are trained to learn a state-dependent strategy (or policy) in order to optimize reward. A chess-playing neural network is an example of reinforcement learning.

This thesis focuses on *supervised classification* training algorithms, and further references to learning algorithms will refer only to supervised classification algorithms unless otherwise specified.

2.1.2 Maximum Likelihood Principle

The *maximum likelihood principle* is a method for obtaining the optimum set of parameters that define a model. We should mention, in supervised learning, a model is defined as a restricted family $\mathcal{H}$ of hypotheses about how Y depends on X.

Assume we have a training dataset $\mathcal{D}$ with model parameters θ. The maximum likelihood of the model parameters θ is:

$$\theta^{\mathrm{ML}} = \arg\max_{\theta} \; p(\mathcal{D}|\theta). \tag{2.2}$$

It is more convenient to work with the natural logarithm of likelihood, because of the monotonically increasing property of logarithm. Hence, maximizing a function is equal to maximizing the log of that function. Therefore, maximum likelihood can be written as:

$$\theta^{\mathrm{ML}} = \arg\max_{\theta} \; \log p(\mathcal{D}|\theta). \tag{2.3}$$

This is called log-likelihood.

The Maximum likelihood principle can also be considered as a special case of the Maximum a Posterior (MAP) principle. MAP is an estimation method based on a posterior

distribution to find the set of model parameters that are most likely to explain the observed experience. This approach is based on posterior distribution and Bayes' rule as follows:

$$\begin{aligned}
\theta^{\mathrm{MAP}} &= \arg\max_{\theta} \; p(\theta|\mathcal{D}) \\
&= \arg\max_{\theta} \; \log p(\theta|\mathcal{D}) \\
&\overset{\text{(i)}}{=} \arg\max_{\theta} \; \log \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\
&= \arg\max_{\theta} \; (\log p(\mathcal{D}|\theta) + \log p(\theta))
\end{aligned} \tag{2.4}$$

where (i) follows from Bayes' rule. Without having prior knowledge about the distribution of the model parameters θ, there is a naïve assumption that all model parameters are equi-probable, therefore $\log p(\theta)$ can be considered as constant for all θ. Hence, the maximum a posterior estimate reduces to

$$\begin{aligned}
\theta^{\mathrm{MAP}} &= \arg\max_{\theta} \; \log p(\mathcal{D}|\theta) \\
&= \theta^{\mathrm{ML}}
\end{aligned} \tag{2.5}$$

which is equivalent to the maximum likelihood in equation (2.3).

In the context of classification, one common approach for finding the model is to apply the maximum likelihood principle. On the other hand, the performance of the classification model can be measured by the cross-entropy loss. Therefore, the maximum likelihood principle corresponds to the minimization of cross-entropy loss, where cross-entropy loss is defined as

$$\ell_{\mathrm{CE}}(q, p) := -\sum_{y \in \mathcal{Y}} q_Y(y) \log p_Y(y) \tag{2.6}$$

where p denotes the probability of the observed label and q is the probability of the true label.

We use a binary classification to prove the relationship between maximum likelihood and cross-entropy loss. Given a model $\mathcal{H}$, dataset $\mathcal{D}$, finding the best distribution $\widehat{p}_{Y|X}(y|x)$ in $\mathcal{H}$ based on $\mathcal{D}$ is the goal, where for each input X the label Y either is 1 or 0, therefore

we have

$$
\begin{aligned}
\widehat{p}^{\mathrm{ML}}_{Y|X}(y|x) &:= \arg\max_{p_{Y|X}\in\mathcal{H}} \; p(\mathcal{D}|p_{Y|X}(y|x)) \\
&\overset{\text{(i)}}{=} \arg\max_{p_{Y|X}\in\mathcal{H}} \; \prod_{i=1}^{N} p_{Y|X}(y_i|x_i) \\
&= \arg\max_{p_{Y|X}\in\mathcal{H}} \; \log \prod_{i=1}^{N} p_{Y|X}(y_i|x_i) \\
&= \arg\min_{p_{Y|X}\in\mathcal{H}} \; \sum_{i=1}^{N} \{-\log p_{Y|X}(y_i|x_i)\} \\
&\overset{\text{(ii)}}{=} \arg\min_{p_{Y|X}\in\mathcal{H}} \; \sum_{i=1}^{N} \{-\mathbb{I}\{y_i=1\}\log p_{Y|X}(1|x_i) - \mathbb{I}\{y_i=0\}\log p_{Y|X}(0|x_i)\} \\
&= \arg\min_{p_{Y|X}\in\mathcal{H}} \; \sum_{i=1}^{N} \ell_{\mathrm{CE}}(\mathbb{I}\{y_i=.\}p_{Y|X}(.|x_i))
\end{aligned}
\tag{2.7}
$$

where (i) by assuming (x_i, y_i)'s are i.i.d. samples from dataset $\mathcal{D}$, and (ii) $\mathbb{I}$ is an indicator function. This shows the maximum likelihood principle is equivalent to the minimization of cross-entropy loss.

2.1.3 Empirical Risk Minimization

As mentioned in Section 2.1.1, the goal in supervised learning is to find a mapping function $F : \mathcal{X} \to \mathcal{Y}$ that maps input object X to class label Y. For this purpose, the penalty for the differences between predictions $F(X;\theta)$, and actual targets Y is defined by the loss function ℓ. Here goal is to find parameters that minimizes this penalty. Minimization over the average of the loss function ℓ gives rise to the population risk:

$$
\theta^* = \arg\min_{\theta} \; \mathbb{E}_{(X,Y)\sim\mathcal{D}} \ell((X,Y);\theta), \tag{2.8}
$$

but as the distribution $\mathcal{D}$ is unknown in most practical situations, the empirical distribution $p_{XY}(x, y)$ is assumed, hence the Empirical Risk Minimization (ERM) principle is defined

as:

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell((X_i, Y_i); \theta)$$

$$= \arg\min_{\theta} \mathcal{L}(\theta) \tag{2.9}$$

where overall loss is defined as

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell((X_i, Y_i); \theta). \tag{2.10}$$

We should mention that the solutions to ERM are equivalent to the classification solution or regression solution based on the loss function. Cross-entropy loss and Hinge loss [41] can be used for classification solution; mean squared error loss and Huber loss [38, 55] can be used for regression solution.

2.2 Neural Networks and Deep Learning

Neural networks are processing systems, that are built to model the mammalian cerebral cortex on a smaller scale. Feed forward networks contain multiple layers. In each layer, there are a number of connected nodes that have an activation function. The first layer of these networks, called the input layer, receives input data, and the last layer of these networks is the output layer. The layers between the input and output layers are called hidden layers. A neural network with more than one hidden layer is called a "deep" network. A neural network and a mathematical model of a neuron in that network is shown in Figure 2.1 where x_i defines inputs, w_i denotes weights, b is bias and σ is a non-linear activation function. A network is called a fully connected neural network or Multi-Layer Perceptron (MLP) if all the neurons are connected to each node of the preceding layer. A fully connected neural network is depicted in Figure 2.1.

These networks support forward and backward passes for predicting the output. If x is defined as the input vector, and y is defined as the class label (output) vector, the output of neurons in layer l of the network is

$$y^{(l+1)} = \sigma\Big(\mathbf{W}^{(l+1)} y^{(l)} + b^{(l+1)}\Big) \tag{2.11}$$

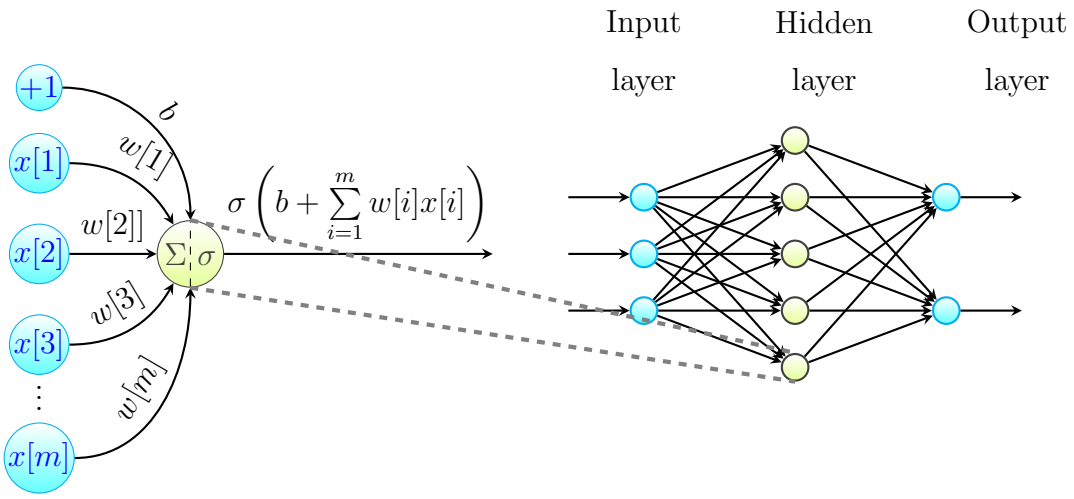


Figure 2.1: A neural network and mathematical model of a neuron.

where $y^0 = x$. The feed-forward pass for Lth hidden layer of a standard neural network is defined as:

$$F(x;\theta) = \text{Softmax}\Big(\sigma\Big(\mathbf{W}^{(L)}\cdots\sigma\big(\mathbf{W}^{(1)}x + b^{(1)}\big)\cdots + b^{(L)}\Big)\Big) \tag{2.12}$$

where a Softmax is used at the last layer of the network for predicting the labels, and $\mathbf{W}^{(L)}$ is a weight matrix for the Lth hidden layer of the network.

After each forward pass, the network calculates the loss function between the predicted class labels, and the true class labels and estimates how far the predicted class labels deviate from the actual ones. Based on that estimation, the network adjusts the weights to bring the predicted class labels closer to the true class labels, and thus achieves a more accurate prediction. This process is called "backward propagation" or backward pass. One of the techniques for the backward pass is the "Stochastic Gradient Descent" (SGD). SGD is an iterative method for updating weight values by using the value of the gradient. More precisely, in SGD gradient is replaced by its stochastic estimation using a single data point:

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta\nabla_\theta\mathcal{L}(\theta^{\text{old}}) \tag{2.13}$$

where η denotes the step size, $\mathcal{L}$ is a measure of loss between the predicted class labels and true class labels, and the gradient $\nabla_\theta\mathcal{L}(\theta^{\text{old}})$ is defined as

$$\nabla_\theta\mathcal{L}(\theta^{\text{old}}) = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta[\ell((X_i, Y_i);\theta^{\text{old}})]. \tag{2.14}$$

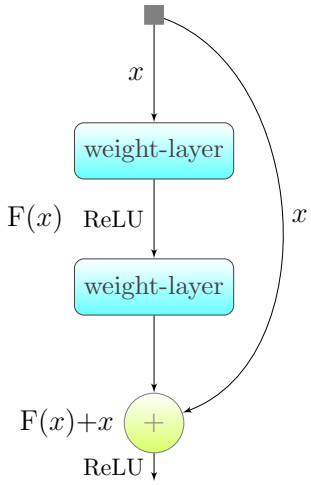


Figure 2.2: Residual Learning.

As we mentioned earlier in this section, increasing the number of hidden layers beyond one gives rise to, what is known as, a "deep neural network". These types of networks have had a huge impact on many challenging real world applications such as speech recognition [33, 26], image recognition [50, 31], natural language processing, etc.

Convolutional Neural Network (CNN) [53, 83] and Recurrent Neural Network (RNN) [19, 12, 63] are two types of these networks, which are used in computer vision and language modeling, respectively.

- In CNNs, each layer has a convolution (filtering) part and a pooling (sub-sampling) part. At the end of these layers there is a regression layer, such as a logistic regression phase, which predicts the label of the input that passes through the network. In other words, the output is a combination of all these layers, where each layer applies a different filter. They are, therefore, very efficient in image recognition and classification. In CNNs, increasing the depth of the network helps to improve recognition and classification accuracy. Because of this property, deeper networks such as Visual Geometry Group (VGG) [71], Residual Network (ResNet) [30] have been introduced.

  In VGG structure, each block contains two convolution layers with Rectified Linear Unit (ReLU) activation functions [42, 57] and a max-pooling layer. At the end of the network, there are several fully connected layers and finally a soft-max layer for

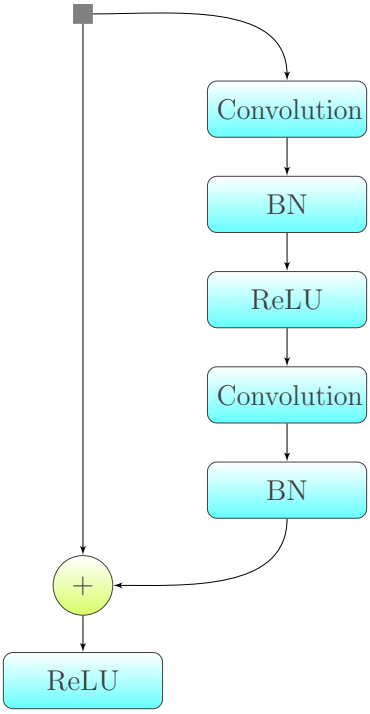


Figure 2.3: Basic diagram of the Residual block for the ResNet.

classification. Based on the number of layers there are different VGG models such as VGG-11, VGG-16, VGG-19 in which there are 11, 16, and 19 layers respectively.

ResNet is an ultra-deep network. This network is based on the idea of residual connection. In the residual connection, the output of each layer is summed up with the original input (Figure 2.2). The advantage of this network is, that it does not suffer from the vanishing gradient problem which occurs in deep neural networks during backward propagation. During backward propagation, the calculated gradient with respect to the weights gets smaller and smaller as we keep moving through the network. As a result, the weights do not get updated fast enough for the earlier layers; which in turn causes the learning process to slow down. This problem is solved by using residual connection in the ResNet structure. The residual block for the ResNet is shown in Figure 2.3. Each block contains two convolution layers with a ReLU activation functions and two Batch Normalization (BN) layers. BN increases the performance and stability of the network by fixing means and variances of each layer's inputs [40].

Table 2.1: Structure of ResNet-18 and ResNet-34

Layer name	output size	18-Layer	34-Layer
conv1	112× 112	7×7, 64,stride2	
conv2	56 × 56	3 × 3 max pool, stride 2	
		$\begin{bmatrix} 3\times 3 & 64 \\ 3\times 3 & 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3 & 64 \\ 3\times 3 & 64 \end{bmatrix} \times 3$
conv3	28 × 28	$\begin{bmatrix} 3\times 3 & 128 \\ 3\times 3 & 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3 & 128 \\ 3\times 3 & 128 \end{bmatrix} \times 4$
conv4	14 × 14	$\begin{bmatrix} 3\times 3 & 256 \\ 3\times 3 & 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3 & 256 \\ 3\times 3 & 256 \end{bmatrix} \times 6$
conv5	7 × 7	$\begin{bmatrix} 3\times 3 & 512 \\ 3\times 3 & 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times 3 & 512 \\ 3\times 3 & 512 \end{bmatrix} \times 3$
	1 × 1	average pool, fully-connected layer and soft-max	

There are different types of ResNets based on the number of layers, such as 18, 34, 50, 101, and even 1202. For example, ResNet-34 has 33 convolutional layers and 1 fully connected layer at the end of the network. The structure of ResNet-18, and ResNet-34 is shown in Table 2.1.

One issue with very deep residual networks is that a gradient might not go through all the residual block weights which may cause only a few blocks to learn useful information or many blocks to learn a smaller amount of information. This problem is called diminishing feature reuse [22, 73, 37]. This problem slows down the training process. To address this problem, the Wide Residual Networks (WideResNet) has been proposed [82]. In this network, the depth of the residual network is decreased but its width is increased. The structure of this network is shown in Table 2.2.

- RNNs can be used to process sequential information. These networks are used predominantly for natural language processing. They have some loops within them, which provide them with memory or the capability to store information for a transient

Table 2.2: Structure of Wide Residual Networks. k denotes the width of network, and B denotes the number of blocks in group

group name	output size	block type
conv1	32×32	$[3 \times 3, 16]$
conv2	32×32	$\begin{bmatrix} 3 \times 3 & 16 \times k \\ 3 \times 3 & 16 \times k \end{bmatrix} \times B$
conv3	16×16	$\begin{bmatrix} 3 \times 3 & 32 \times k \\ 3 \times 3 & 32 \times k \end{bmatrix} \times B$
conv4	8×8	$\begin{bmatrix} 3 \times 3 & 64 \times k \\ 3 \times 3 & 64 \times k \end{bmatrix} \times B$
avg-pool, fully-connected layer	1×1	$[8 \times 8]$

duration in order to predict the output. Because of this ability to store information, these networks are able to learn sequences. Long Short-Term Memory (LSTM) is one of the RNN architectures [34]. Each LSTM network has a cell with the memory part and regulators which are made up of input gates, output gates and a forget gate. The cell checks the dependency between elements in the input sequence. The forget gate controls which value remains in the cell. The LSTM architecture is shown in Figure 2.4.

Let x_t denote the input vector, h the hidden state, i the input gate, o the output gate, f the forget gate and c the cell state. $\mathbf{W}$, $\mathbf{U}$ and b are the weight matrices and the bias vector parameters which need to be learned during training. Therefore the

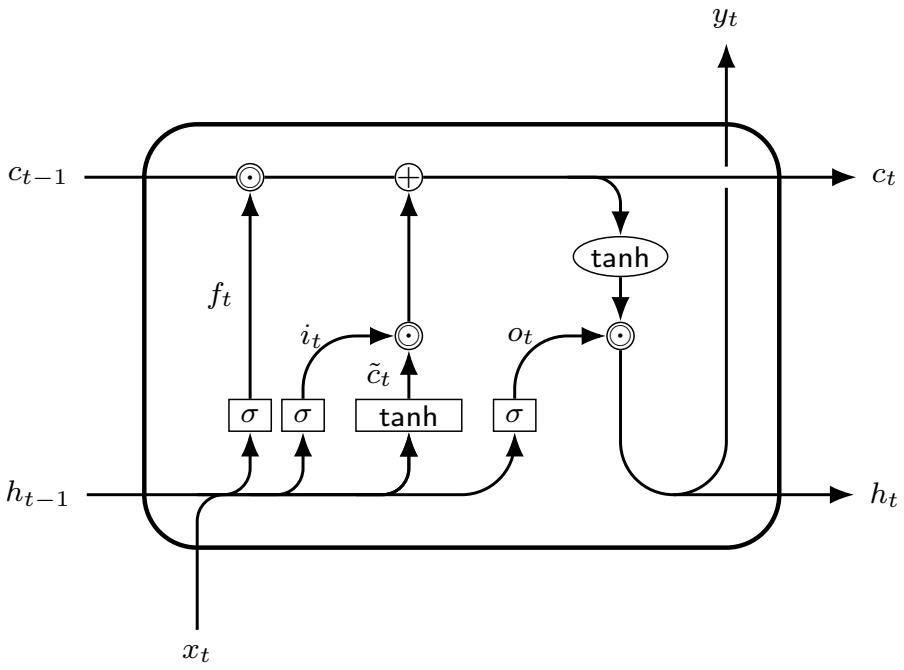


Figure 2.4: LSTM network structure.

forward pass of an LSTM unit is:

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f) \\
\tilde{c}_t &= \tanh(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c) \\
i_t &= \sigma(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) \\
c_t &= i_t \odot \tilde{c}_t + f_t \odot c_{t-1} \\
o_t &= \sigma(\mathbf{W}_0 x_t + \mathbf{U}_0 h_{t-1} + b_0) \\
h_t &= y_t = o_t \odot \tanh(c_t)
\end{aligned}
\tag{2.15}
$$

where the initial values are $c_0 = 0$ and $h_0 = 0$ and $\odot$ denotes element-wise product. σ denotes sigmoid function and tanh denotes hyperbolic tangent function. The subscript t indexes the time step.

In addition to the LSTM, CNNs can also be used for NLP tasks [44]. The difference between CNNs for image processing and NLP tasks is that in NLP tasks, the CNNs

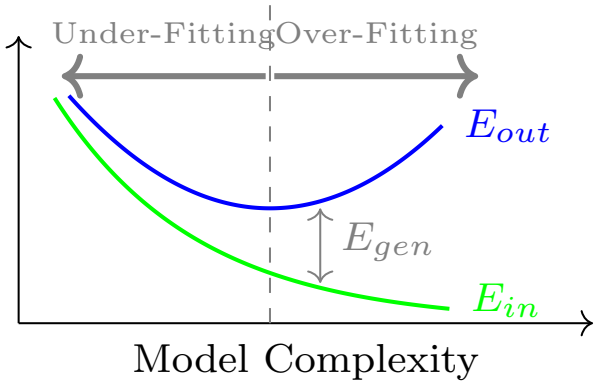


Figure 2.5: Training and test errors vs model complexity, where E_{gen} denotes generalization gap.

filters slide over the rows of sentence matrices, whereas for image processing they slide over patches of images. In the sentence matrices, each row corresponds to a word, in other words, each row is a vector of the word. These vectors can be word embeddings or one-hot vectors which index the word into a vocabulary. After the convolutional layers, max-pooling is applied over the output and finally, logistic regression is applied to classify the sentence.

2.3 Generalization, Overfitting, and Underfitting

Generalization refers to how well the concepts are learned by a model for the data that does not participate in training. The difference between a model's performance on training data and its performance on unseen data drawn from the same distribution is called the *generalization gap*. Figure 2.5 depicts the relationship between model complexity and model errors which are in-sample error or training error (E_{in}) and out-sample error or test error (E_{out}). Training error refers to the average error of the model over the training samples, and test error refers to the average error of the model over testing samples. Figure 2.5 shows that as the model complexity increases, the ability of a model to learn arbitrary patterns improves therefore training error decreases. On the other hand, test error decreases up to a certain point (at which an ideal model is reached) and then starts to increase. In this

situation, the model is called overfitted [24]. Usually, overfitting occurs when the model learns the training data too well. It captures even the noise embedded in the data. In contrast to overfitting, underfitting happens when the model cannot fit the data well enough i.e., the machine learning model is not complex enough to capture the relationship between datasets and class labels. The algorithm can learn quite fast but it is not flexible enough to predict a complex problem.

2.4 Techniques to Improve Generalization

Regularization is one of the methods used to improve generalization in deep learning networks. Guo et al. [28] group regularization into two categories: data-independent, and data-dependent. The difference between these two categories is in the type of constraints imposed on the model i.e., those that exploit the structure of data and those that do not exploit the structure of data (e.g., the distribution). Weight decay [29], and dropout [72] are two examples of data-independent regularization. Data augmentation schemes [53, 70, 68], and adversarial training schemes [25] are some examples of data-dependent regularization.

2.4.1 Weight Decay and Dropout

Weight decay and *dropout* are two standard regularization techniques.

- In weight decay, the weights of the network are multiplied by a factor less than 1, after each update [29]. This helps to prevent weights from growing too large. $L2$ is one type of this technique. In $L2$ the regularization term $\lambda\|\theta\|_2^2$ is added to the loss function as a penalty term and controls the complexity of the model:

$$\widehat{\theta} = \arg\min_{\theta} \left(\ell((X, Y); \theta) + \lambda\|\theta\|_2^2\right) \tag{2.16}$$

  where $\ell((X, Y); \theta)$ denotes the loss function.

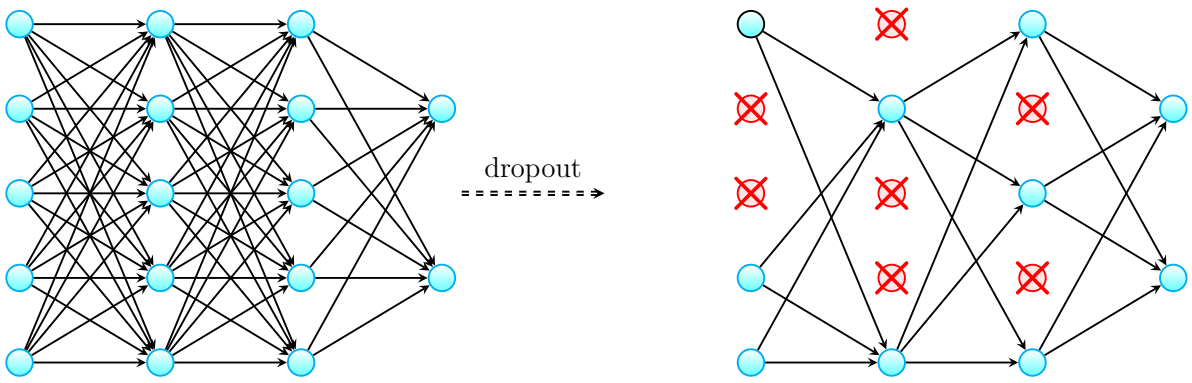


Figure 2.6: The effects of applying dropout with $p = 0.5$ to a deep multilayer perceptron.

- Dropout refers to the technique of randomly ignoring some neurons during training of the model [72]. In other words, some neurons are not considered in the forward and backward passes (Figure 2.6). Let us express a neural network for mapping input vector x to output vector y with L-hidden layers as follows:

$$
\begin{aligned}
x^{(1)} &= f_1(x^{(0)}) \\
&\vdots \\
x^{(L)} &= f_L(x^{(L-1)}) \\
y &= g(x^{(L)})
\end{aligned}
\tag{2.17}
$$

  where $x^{(l)}$ denotes the vector represented by layer l, and $x^{(0)} = x$. f_l denotes the mapping from layer $l-1$ to layer l and g denotes the mapping from layer L to y. For each layer of the network we choose a parameter $p^{(l)}$ between $(0, 1]$ and we delete each node in layer l with probability $1 - p^{(l)}$. More precisely, we create a random vector $z^{(l)}$ having the same dimension as the input vector and each element of it is drawn i.i.d from a Bernoulli distribution with parameter $p^{(l)}$. By applying dropout,

the expression of the network will change to:

$$
\begin{aligned}
x^{(1)} &= f_1(z^{(0)} \odot x^{(0)}) \\
&\vdots \\
x^{(L)} &= f_L(z^{(L-1)} \odot x^{(L-1)}) \\
y &= g(z^{(L)} \odot x^{(L)})
\end{aligned} \tag{2.18}
$$

where $\odot$ denotes element-wise product.

2.4.2 Data Augmentation

The data augmentation technique adds a type of mutation to the original training samples [70, 68]. As a result, the network will see new training samples in each iteration. This will help the network to encounter fresh samples each time which improves learning. The original labels of these input samples do not change with this transformation. The most popular transformations in image applications include flips, rotations, crops, and scaling the image.

2.4.3 MixUp

One of the effective data augmentation approaches for improving the accuracy of deep classification models is *MixUp* [85], which can also be considered as a data-dependent regularization scheme.

MixUp encourages the model to behave linearly in between training examples. This linear behaviour reduces the amount of undesirable oscillations when predicting outside of the training examples. The linear combination between two input samples and their corresponding target outputs after using MixUp is:

$$X^{\text{Mix}} = \lambda X + (1 - \lambda)\widehat{X} \tag{2.19}$$

$$Y^{\text{Mix}} = \lambda Y + (1 - \lambda)\widehat{Y} \tag{2.20}$$

where $\lambda \sim Beta(\alpha, \alpha)$, for $\alpha \in (0, \infty)$. The loss function for MixUp is:

$$\ell^{\text{Mix}} = \ell\big((\lambda X + (1-\lambda)\widehat{X}, \lambda Y + (1-\lambda)\widehat{Y}); \theta\big). \tag{2.21}$$

2.4.4 Between Class Learning

Between Class (BC) Learning technique is another effective data augmentation approach to improve the accuracy of deep classification models [75, 76]. In BC Learning technique images are generated by mixing two images belonging to different classes with a random ratio.

Consider two input images, X and $\widehat{X}$ with standard deviations σ_1, and σ_2 respectively. In BC Learning, a combination of these two images is obtained by:

$$X^{\text{BC}} = \frac{p(X - \mu_1) + (1-p)(\widehat{X} - \mu_2)}{\sqrt{p^2 + (1-p)^2}} \tag{2.22}$$

where μ_1 and μ_2 denotes mean values, $p = \dfrac{1}{1 + \dfrac{\sigma_1}{\sigma_2}.\dfrac{1-\lambda}{\lambda}}$ with a random ratio λ selected from the uniform distribution $U(0, 1)$ and its corresponding label is linearly determined as follows:

$$Y^{\text{BC}} = \lambda Y + (1-\lambda)\widehat{Y}. \tag{2.23}$$

2.5 Information Theoretic Concepts

In this section we will explain certain concepts of information theory. One of them is entropy. Entropy is a measure of the uncertainty of a random variable [14]. The entropy of a discrete random variable Y distributed according to $p_Y(y)$ is defined as follows

$$\begin{aligned} H(Y) &:= -\sum_{y \in \mathcal{Y}} p_Y(y) \log p_Y(y) \\ &= \mathbb{E}_{y \sim p_Y(y)} \log \frac{1}{p_Y(y)}. \end{aligned} \tag{2.24}$$

The log is to the base 2 and entropy is expressed in bits. $H(Y)$ is greater than or equal to 0.

If $(X, Y) \sim p_{XY}(x, y)$ is a joint distribution then the conditional entropy of Y given X is defined as

$$\begin{aligned} H(Y|X) &:= \sum_{x \in \mathcal{X}} p_X(x) H(Y|X = x) \\ &= -\sum_{x \in \mathcal{X}} p_X(x) \sum_{y \in \mathcal{Y}} p_{Y|X}(y|x) \log p_{Y|X}(y|x) \\ &= -\sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p_{XY}(x, y) \log p_{Y|X}(y|x) \\ &= \mathbb{E}_{(x,y) \sim p_{XY}(x,y)} \log \frac{1}{p_{Y|X}(y|x)} \end{aligned} \tag{2.25}$$

where $H(Y|X)$ denotes the uncertainty remaining in Y after observing X [14]. However, $H(Y|X) \neq H(X|Y)$, but $H(X) - H(X|Y) = H(Y) - H(Y|X)$. The joint entropy is also defined as

$$\begin{aligned} H(X, Y) &:= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log p_{XY}(x, y) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log(p_X(x) p_{Y|X}(y|x)) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log p_X(x) - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log p_{Y|X}(y|x) \\ &= -\sum_{x \in \mathcal{X}} p_X(x) \log p_X(x) - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_{XY}(x, y) \log p_{Y|X}(y|x) \\ &= H(X) + H(Y|X). \end{aligned} \tag{2.26}$$

For n random variable $X_1, X_2, \cdots, X_n$ the chain rule holds [14]:

$$\begin{aligned}
H(X_1, X_2, \cdots, X_n) &= -\sum_{x_1,\cdots,x_n} p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) \log p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) \\
&\overset{\text{(i)}}{=} -\sum_{x_1,\cdots,x_n} p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) \log \prod_{i=1}^{n} p_{X_i|X_{i-1}\cdots,X_1}(x_i|x_{i-1}\cdots, x_1) \\
&= -\sum_{x_1,\cdots,x_n} \sum_{i=1}^{n} p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) \log p_{X_i|X_{i-1}\cdots,X_1}(x_i|x_{i-1}\cdots, x_1) \\
&= -\sum_{i=1}^{n} \sum_{x_1,\cdots,x_n} p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) \log p_{X_i|X_{i-1}\cdots,X_1}(x_i|x_{i-1}\cdots, x_1) \\
&= -\sum_{i=1}^{n} \sum_{x_1,\cdots,x_i} p_{X_1,\cdots,X_i}(x_1, \cdots, x_i) \log p_{X_i|X_{i-1}\cdots,X_1}(x_i|x_{i-1}\cdots, x_1) \\
&= \sum_{i=1}^{n} H(X_i|X_{i-1}, \cdots, X_1)
\end{aligned} \tag{2.27}$$

where (i) by considering $p_{X_1,\cdots,X_n}(x_1, \cdots, x_n) = \prod_{i=1}^{n} p_{X_i|X_{i-1}\cdots,X_1}(x_i|x_{i-1}\cdots, x_1)$.

The *relative entropy* or *Kullback-Leibler (KL) divergence* is a measure of the inefficiency in assuming that the distribution is $q_X(x)$ when the true distribution is $p_X(x)$ [14]. If $p_X(x)$ and $q_X(x)$ are two distributions, the relative entropy or KL-divergence of $p_X(x)$ with respect to $q_X(x)$ is defined as

$$\begin{aligned}
\mathrm{KL}(p_X(x)\|q_X(x)) &:= \sum_{x\in\mathcal{X}} p_X(x) \log \frac{p_X(x)}{q_X(x)} \\
&= \mathbb{E}_{x\sim p_X(x)} \log \frac{p_X(x)}{q_X(x)}.
\end{aligned} \tag{2.28}$$

The KL-divergence is always nonnegative, and is zero if and only if $p_X(x) = q_X(x)$ [14] . KL-divergence is not a distance measure therefore we have:

$$\mathrm{KL}(p_X(x)\|q_X(x)) \neq \mathrm{KL}(q_X(x)\|p_X(x)). \tag{2.29}$$

For a given random variable X with distribution $p_X(x)$, if $p_{Y|X}(y|x)$ and $q_{Y|X}(y|x)$ are two conditional distributions, each defining a pair of random variables (X, Y), the conditional KL-divergence is defined as:

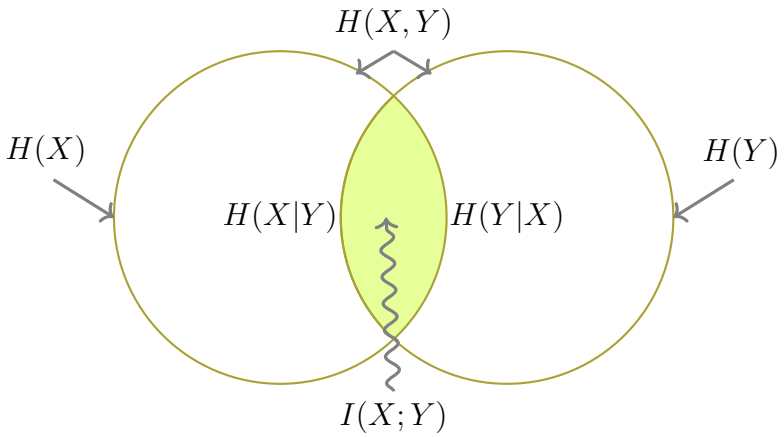


Figure 2.7: The Venn diagram depicting the relationship between mutual information and entropies. The intersection of the circles is the mutual information $I(X;Y)$.

$$\mathrm{KL}(p_{Y|X}(y|x)\|q_{Y|X}(y|x)) := \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y)\log\frac{p_{Y|X}(y|x)}{q_{Y|X}(y|x)}. \tag{2.30}$$

The *mutual information* shows how much uncertainty in one random variable will be resolved by observing another random variable [14]. If X and Y are two discrete random variables, distributed according to $p_{XY}(x,y)$, with marginal distributions $p_X(x) = \sum_{y\in\mathcal{Y}} p_{XY}(x,y)$, and $p_Y(y) = \sum_{x\in\mathcal{X}} p_{XY}(x,y)$, the mutual information between X and Y is defined as

$$\begin{aligned} I(X;Y) &:= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p_{XY}(x,y)\log\frac{p_{XY}(x,y)}{p_X(x)p_Y(y)} \\ &= \mathrm{KL}\big(p_{XY}(x,y)\|p_X(x)p_Y(y)\big). \end{aligned} \tag{2.31}$$

Intuitively, $I(X;Y)$ measures the level of dependence between X and Y. If X and Y are completely independent, $I(X;Y) = 0$. In other words, the mutual information vanishes for fully independent variables. Mutual information is always nonnegative.

The Venn diagram in Figure 2.7 depicts the following relationships between mutual information and entropies of two random variables X and Y:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{2.32}$$

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{2.33}$$

$$I(X;Y) = I(Y;X) \tag{2.34}$$

$$I(X;X) = H(X) \tag{2.35}$$

where

$$\begin{aligned} I(X;Y) &:= \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y) \log \frac{p_{XY}(x,y)}{p_X(x)p_Y(y)} \\ &= \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y) \log \frac{p_{X|Y}(x|y)}{p_X(x)} \\ &= -\sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y) \log p_X(x) + \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y) \log p_{X|Y}(x|y) \\ &= -\sum_{x\in\mathcal{X}} p_X(x) \log p_X(x) + \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y) \log p_{X|Y}(x|y) \\ &= H(X) - H(X|Y), \end{aligned} \tag{2.36}$$

since $H(X) - H(X|Y) = H(Y) - H(Y|X)$, therefore

$$I(X;Y) = H(Y) - H(Y|X), \tag{2.37}$$

because $H(X,Y) = H(X) + H(Y|X)$ we have

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) \\ &= H(Y) - (H(X,Y) - H(X)) \\ &= H(X) + H(Y) - H(X,Y), \end{aligned} \tag{2.38}$$

since $H(X|X) = 0$ we have

$$\begin{aligned} I(X;X) &= H(X) - H(X|X) \\ &= H(X). \end{aligned} \tag{2.39}$$

The chain rule of information is defined as

$$\begin{aligned} I(X_1, X_2, \cdots, X_n; Y) &= H(X_1, X_2, \cdots, X_n) - H(X_1, X_2, \cdots, X_n|Y) \\ &= \sum_{i=1}^{n} H(X_i|X_{i-1}, \cdots, X_1) - \sum_{i=1}^{n} H(X_i|X_{i-1}, \cdots, X_1, Y) \\ &= \sum_{i=1}^{n} I(X_i, Y|X_{i-1}, \cdots, X_1). \end{aligned} \tag{2.40}$$

If X — Y — Z form a Markov chain, then

$$I(X;Y) \geq I(X;Z) \tag{2.41}$$

$$I(X;Y) \geq I(X;Y|Z). \tag{2.42}$$

If the encoder describes the source sequence X^n by an index $f_n(X^n) \in \{1, 2, \ldots, 2^{nR}\}$, and the decoder describes X^n by an estimate $\widehat{X}^n$ then a *distortion function* (distortion measure) is a mapping

$$d : \mathcal{X} \times \widehat{\mathcal{X}} \to \mathcal{R}^+ \tag{2.43}$$

where the distortion $d(x, \widehat{x})$ is a measure of the cost of representing the symbol x by the symbol $\widehat{x}$ [14].

A $(n, 2^{nR})$-rate distortion code consists of an encoding function

$$f_n : \mathcal{X}^n \to \{1, 2, \ldots, 2^{nR}\} \tag{2.44}$$

and a decoding function

$$g_n : \{1, 2, \ldots, 2^{nR}\} \to \widehat{\mathcal{X}}^n \tag{2.45}$$

and the average distortion for the $(n, 2^{nR})$ code is

$$\begin{aligned} D &= \mathbb{E}\ d(X^n, g_n(f_n(X^n))) \\ &= \sum_{x^n \sim \mathcal{X}^n} p_{X^n}(x^n) d(x^n, g_n(f_n(x^n))) \end{aligned} \tag{2.46}$$

where R denotes *rate*. The rate is the average number of bits per message. A rate distortion pair (R, D) is said to be achievable if there exists a sequence of $(n, 2^{nR})$-rate distortion codes (f_n, g_n) with

$$\lim_{n \to \infty} \mathbb{E}\ d(X^n, g_n(f_n(X^n)) \leq D \tag{2.47}$$

There is a tradeoff between the rate and the average distortion, the larger the rate, the smaller the achievable distortion. This tradeoff is characterized by the rate-distortion function $R(D)$ [14]. The rate-distortion function is defined as the infimum of all achievable rates R under a given constraint on the average distortion D as follows:

$$R(D) := \min_{p_{\widehat{X}|X}(\widehat{x}|x):\sum_{(x,\widehat{x})} p_X(x)p_{\widehat{X}|X}(\widehat{x}|x)d(x,\widehat{x})\leq D} I(X;\widehat{X}). \tag{2.48}$$

In rate-distortion calculation for a given source $p_X(x)$, different choices of distortion measures will give different results, meaning that the distortion measure is itself a part of the problem setup. Therefore, the rate-distortion function always depends on the choice of distortion measure. Moreover, defining an appropriate distortion measure is not always possible in practical applications such as image and natural language processing. The Information Bottleneck (IB) principle was introduced to overcome these problems. We will explain this principle in the next chapter.

2.6 Summary

In this chapter, we gave brief summaries of machine learning algorithms, models, and concepts that we will use in the next chapters. We explained the basic concepts of information theory such as entropy, KL-divergence, and mutual information. We also explained the concept of rate-distortion function.

Chapter 3

Related Work

3.1 Introduction

In this chapter, we explain the information bottleneck principle in Section 3.2. Then in Section 3.3, we survey the solutions to the information bottleneck problem. We explain Mutual Information Neural Estimator in Section 3.4, and finally, we summarize this chapter in Section 3.5.

3.2 Information Bottleneck

The Information Bottleneck (IB) method is a technique in information theory introduced by Tishby et al. [74]. The main idea in IB is to extract the most relevant and meaningful information from the input data. A natural approach for extracting the relevant information is by lossy source compression. In this approach consider X as input with probability distribution $p_X(x)$, and Y as class label. A new variable T is introduced, that is a compressed representation of object X which still preserves information about Y. One is interested in finding a stochastic mapping from input X to representation T which is characterized by a conditional probability distribution function $p_{T|X}(t|x)$. One can determine the effectiveness of this compression by two factors: the rate, which is the average number of bits per

message, and the distortion function, which measures the cost of wrong representation. This requires a solution to the coding equations for $X \to T$ and $T \to Y$. They are solved by using a re-estimation method that generalizes the Blahut-Arimoto algorithm. In this approach one is interested in simultaneously satisfying both

$$p_T(t) = \sum_{x \in \mathcal{X}} p_X(x) p_{T|X}(t|x) \tag{3.1}$$

and

$$p_{T|X}(t|x) = \frac{p_T(t)}{Z(x,\beta)} \exp[-\beta d(x,t)] \tag{3.2}$$

where $Z(x,\beta)$ is a normalization function, β is a Lagrange multiplier, and d denotes a distortion function. The natural approach in order to achieve this goal as introduced in Theorem 3.2.1 , is by alternating the iteration between them until they reach convergence,

Theorem 3.2.1. *[74] Equation (3.1) and equation (3.2) are satisfied simultaneously at the minimum of the functional,*

$$\begin{aligned} \mathcal{F} &= -\langle \log Z(x,\beta) \rangle_{p_X(x)} \\ &= I(X;T) + \beta \langle d(x,t) \rangle_{p_{XT}(x,t)} \end{aligned} \tag{3.3}$$

where the minimization is done independently over the convex sets of the normalized distributions, $p_T(t)$ and $p_{XT}(x,t)$,

$$\min_{p_T(t)} \min_{p_{T|X}(t|x)} \mathcal{F}[p_T(t); p_{T|X}(t|x)] \tag{3.4}$$

These independent conditions correspond precisely to alternating iterations of equation (3.1) and equation (3.2). Denoting by i the iteration step,

$$p_T^{i+1}(t) = \sum_{x \in \mathcal{X}} p_X(x) p_{T|X}^i(t|x) \tag{3.5}$$

$$p_{T|X}^i(t|x) = \frac{p_T^i(t)}{Z^i(x,\beta)} \exp[-\beta d(x,t)] \tag{3.6}$$

where the normalization function $Z^i(x,\beta)$ is evaluated for every i in equation (3.6). Furthermore, these iterations converge to a unique minimum of $\mathcal{F}$ in the convex sets of the two distributions.

Measuring the distortion term is rarely possible and therefore Tishby et al. [74, 64, 66] proposed a direct approach to the solution of the IB problem. In this approach, the main goal is to capture a tradeoff between minimality of the representation of X, achieved by minimizing $I(X;T)$, and sufficiency of information on Y, achieved by constraining the value of $I(Y;T)$.

The IB Markovian can be written as:

$$T—X—Y$$

which shows $I(T;Y) \leq I(X;Y)$. It is obvious that T cannot have more information than X since it is only dependent on X. Therefore, we have

$$\begin{aligned} p_T(t) &= \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} p_{XYT}(x,y,t) \\ &= \sum_{x\in\mathcal{X}} p_X(x) p_{T|X}(t|x) \end{aligned} \tag{3.7}$$

$$p_{Y|T}(y|t) = \frac{1}{p_T(t)} \sum_{x\in\mathcal{X}} p_{XYT}(x,y,t). \tag{3.8}$$

There is always a tradeoff between compression of the representation, and preservation of the relevant or meaningful information. Tishby et al. addressed the following optimization problem for this tradeoff [74, 64]:

$$\mathcal{L}_{\text{IB}}[p_{T|X}(t|x)] := I(X;T) - \beta I(Y;T) \tag{3.9}$$

where β is a Lagrange multiplier that controls the trade-off and the free parameters correspond to the stochastic mapping with respect to the mapping $p_{T|X}(t|x)$. The optimal solution for equation (3.9) is presented in Theorem 3.2.2 as follows:

Theorem 3.2.2. *[74] The optimal solution that minimizes (3.9) satisfies the equation*

$$p_{T|X}(t|x) = \frac{p_T(t)}{Z(x,\beta)} \exp[-\beta \sum_{y\in\mathcal{Y}} p_{Y|X}(y|x) \log \frac{p_{Y|X}(y|x)}{p_{Y|T}(y|t)}] \tag{3.10}$$

where $Z(x,\beta) = \sum_{t\in\mathcal{T}} p_T(t) \exp[-\beta KL[p_{Y|X}(y|x) \| p_{Y|T}(y|t)]]$ *is a normalization function and*

$$p_{Y|T}(y|t) = \frac{1}{p_T(t)} \sum_{x\in\mathcal{X}} p_{Y|X}(y|x) p_{T|X}(t|x) p_X(x). \tag{3.11}$$

The trade-off between compression of the representation and learning is shown in Figure 3 of [69]. This Figure shows the interesting phenomena that at first $I(Y;T)$ increases (called the fitting phase), then at some point a phase transition occurs and $I(X;T)$ starts to decrease (called the compression phase) [69]. In other words, in the first phase of training the layers' information regarding the labels increases, which means $I(Y;T)$ increases, and in the second phase of training the layers' information regarding $I(X;T)$ decreases and the layers lose irrelevant information until convergence.

3.3 Solutions to Information Bottleneck Problem

The proposed approach for solving the IB optimization problem for both supervised and unsupervised learning in [74, 64, 10, 20] depends on assuming the knowledge of the joint distribution of X and Y. However in practice, this joint distribution is unknown, and one assumes that i.i.d samples are drawn from the assumed joint probability distribution, and those samples are used to create a maximum-likelihood estimate of the empirical distribution of the sample. Shamir et al. in [66] investigated how the IB framework can be generalized when X and Y are two discrete random variables and an empirical joint distribution of X and Y can be used to solve the IB optimization problem.

Subsequently, the IB principle was also applied to the case of continuous random variables. For this purpose Alemi et al. in [4] introduced a variational approximation to the IB. This method helps to parameterize the IB and make it applicable to deep neural networks. By using this method the authors were able to construct a lower bound for the IB objective, which gave rise to a framework which is called *Deep Variational Information Bottleneck* (VIB).

In VIB, the authors use a variational approximation for calculating $I(Y;T)$ and $I(X;T)$ in the IB objective function

$$\min_{p_{T|X}(t|x)} I(X;T) - \beta I(Y;T) \tag{3.12}$$

or equivalently

$$\max_{p_{T|X}(t|x)} I(Y;T) - \widehat{\beta} I(X;T) \tag{3.13}$$

The lower bound that is achieved by this method is:

$$\begin{aligned} I(Y;T) - \widehat{\beta} I(X;T) &\geq \int p_X(x) p_{Y|X}(y|x) p_{T|X}(t|x) \log q_{Y|T}(y|t) \; dxdydt \\ &\quad - \widehat{\beta} \int p_X(x) p_{T|X}(t|x) \log \frac{p_{T|X}(t|x)}{r_T(t)} \; dxdt \\ &= \mathcal{L} \end{aligned} \tag{3.14}$$

where $q_{Y|T}(y|t)$ and $r_T(t)$ are variational approximation for $p_{Y|T}(y|t)$ and $p_T(t)$, respectively. $p_{XY}(x,y) = p_X(x)p_{Y|X}(y|x)$ can be approximated by using the empirical distribution $p_{XY}(x,y) = \frac{1}{N}\sum_{n=1}^{N} \delta_{x_n}(x)\delta_{y_n}(y)$. Therefore in practice $\mathcal{L}$ can be calculated as follows:

$$\mathcal{L} \approx \frac{1}{N} \sum_{n=1}^{N} \Big[\int p_{T|X_n}(t|x_n) \log q_{Y_n|T}(y_n|t) \; dt - \widehat{\beta} p_{T|X_n}(t|x_n) \log \frac{p_{T|X_n}(t|x_n)}{r_T(t)} \Big] \tag{3.15}$$

where $p_{T|X}(t|x)$ can be obtained by using an encoder to form a Gaussian distribution $\mathcal{N}(t|f_e^{\mu}(x), f_e^{\Sigma}(x))$, where f_e is an MLP network, with K-dimensional mean μ of t and $K \times K$ is the covariance matrix Σ as output. The sampling makes backward propagation difficult. To overcome this problem the Reparameterization Trick [46] is applied such that $t = f(x, \epsilon)$ where ϵ is a Gaussian random variable. The second term in equation (3.15) is KL-divergence between $p_{T|X}(t|x)$ and $r_T(t)$ which can be considered as a regularization term, thus putting everything together, we minimize the following objective function

$$\mathcal{J} = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{\epsilon \sim p(\epsilon)}[-\log q_{Y_n|T}(y_n|f(x_n,\epsilon))] + \widehat{\beta} \mathrm{KL}[p_{T|X_n}(t|x_n)||r_T(t)]. \tag{3.16}$$

In [4] the authors experimentally established the performance of this framework. Achille et al. [2] efficiently approximated and minimized the IB Lagrangian by injecting noise into the layers of the network. The idea of injecting noise into the stochastic hidden layer of the network with a regularizer was also proposed by Chen et al. [11] and Im et al. [39]. Achille et al. proposed injecting noise by a generalization of the dropout layer. Each representation T from X is achieved by element-wise multiplication of random samples ϵ and deterministic mapping $F : \mathcal{X} \to \mathcal{T}$. Random samples ϵ are drawn from a noise distribution $p_{\alpha(X)}(\epsilon)$ with unit mean and variance that depends on input X, therefore

$$T = \epsilon \odot F(X) \tag{3.17}$$

where $\odot$ denotes element-wise multiplication and $\epsilon \sim p_{\alpha(X)}(\epsilon)$. According to the authors the Bernoulli distribution and the log-normal distribution $\log \mathcal{N}(0, \alpha^2(X))$, can be considered for $p_{\alpha(X)}(\epsilon)$. By fixing the noise distribution and considering a prior distribution for the second term of the approximation of the IB Lagrangian, one can minimize the following objective function:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t \sim p_{T|X_i}(t|x_i)}[-\log p_{Y_i|T}(y_i|t)] + \widehat{\beta} \mathrm{KL}(p_{T|X_i}(t|x_i) \| p_T(t)) \tag{3.18}$$

where the first term denotes cross-entropy loss.

In addition to this, Achille et al. in [1] discussed a new approach to the IB problem in which the information in the weights is taken into consideration. They introduced an IB Lagrangian between the weights of a network and the training data, while traditional works had introduced an IB Lagrangian between the activations and test datum. Therefore they proposed the IB optimization formulation

$$\mathcal{L}(q_{W|\mathcal{D}}(w|\mathcal{D})) = H_{p,q}(Y|X, W) + \beta I(W; \mathcal{D}) \tag{3.19}$$

where the first term is cross-entropy loss that the network is trained to minimize and is defined as

$$H_{p,q}(Y|X, W) = \mathbb{E}_{\mathcal{D}} \mathbb{E}_{w \sim q_{W|\mathcal{D}}(w|\mathcal{D})} \sum_{i=1}^{N} -\log q_{Y|X,W}(y_i|x_i, w) \tag{3.20}$$

Here p shows the real data distributions and q the approximate distributions achieved during training. The second term denotes information that weights carry about the dataset $\mathcal{D}$, for computing this term, Achille et al. used an upper bound

$$\begin{aligned} I(W; \mathcal{D}) &= \mathbb{E}_{\mathcal{D}} \mathrm{KL}(q_{W|\mathcal{D}}(w|\mathcal{D}) \| q_W(w)) \\ &\leq \mathbb{E}_{\mathcal{D}} \mathrm{KL}(q_{W|\mathcal{D}}(w|\mathcal{D}) \| q_W(w)) + \mathrm{KL}(q_W(w) \| p_W(w)) \\ &= \mathbb{E}_{\mathcal{D}} \mathrm{KL}(q_{W|\mathcal{D}}(w|\mathcal{D}) \| p_W(w)) \end{aligned} \tag{3.21}$$

where $p_W(w)$ is any fixed distribution of the weights. The optimization formulation (3.19) is a function of weights rather than T, and the authors verified the performance of this approach by experiment [1].

Goldfeld et al. [23] investigated the approximation of the regularization part in the IB method by injecting noise to intermediate layers. In [23], estimation of $I(X, T_l)$ is investigated where T_l is the output of l^{th} hidden layer. In this study mapping from X to T is considered as a stochastic mapping by injecting noise to intermediate layers (but not to the final output $T_L = f_L(T_{L-1})$):

$$T_l = f_l(T_{l-1}) + Z_l \tag{3.22}$$

where $f_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ and $Z_l \sim \mathcal{N}(0, \beta^2 I_{d_l})$. In [23], an estimator for $I(X, T_l)$ is designed based on differential entropy estimation and Goldfeld et al. explained each differential entropy is estimated and computed by two steps: one can approximate each true entropy by the differential entropy of a known Gaussian mixture, but since there is no closed form expression for Gaussian mixture, in the second step, a Monte Carlo Integration (MCI) method is developed to numerically evaluate it.

Sampling procedure and estimator contains two cases: unconditional entropy, and conditional entropy. In the first case, $T_l = S_l + Z_l$ where S_l and Z_l are independent, so $p_{T_l} = p_{S_l} * \varphi_\beta$. For estimating $h(p_{T_l})$ each $x \in \mathcal{X}$ is fed to the neural network and outputs in the $(l-1)$-th layer are collected, then function f_l is applied on these collected output to achieve $S_l = \{s_{l,i}\}_{i\in[n]}$. Given this, $h(p_{T_l})$ can be estimated via $h(\widehat{p}_{S_l} * \varphi_\beta)$ which is differential entropy of a Gaussian mixture with centers $\{S_{l,i}\}_{i\in[n]}$ [23], and $\widehat{p}_{S_l}$ is denoted as empirical probability mass function. In conditional entropy case, $h(p_{T_l|X=x})$ is estimated where $p_{T_l|x} = p_{S_l|x} * \varphi_\beta$. In this case $p_{S_l|X=x}$ can be achieved by giving input object x to the neural network n_x times and collecting T_{l-1} outputs, so $S_l^{(x)} = \{s_{l,i}^{(x)}\}_{i\in[n_x]}$. Therefore, each $h(p_{T_l|X=x})$ is estimated by $h(\widehat{p}_{S_l^{(x)}} * \varphi_\beta)$.

Considering these approximations, mutual information estimator estimates $I(X; T_l)$ by

$$\widehat{I}_{\text{SP}} = h(\widehat{p}_{S_l} * \varphi_\beta) - \frac{1}{n}\sum_{x\in\mathcal{X}} h(\widehat{p}_{S_l^{(x)}} * \varphi_\beta) \tag{3.23}$$

They also proposed a bound for the estimation risk of $\widehat{I}_{\text{SP}}$, as follows:

Theorem 3.3.1. *[23] Fix $l \in [L-1]$ and assume $\|f_l\|_\infty \le 1$. For $\widehat{I}_{SP}$ from equation (3.23)*

with $n = n_x$, for all $x \in \mathcal{X}$, we have

$$\sup_{p_X} \mathbb{E}|I(X, T_l) - \widehat{I}_{SP}| \leq \frac{8c^{d_l} + d_l \log(1 + \frac{1}{\sigma^2})}{4\sqrt{n}} \tag{3.24}$$

where c is a numerical constant and d is dimension.

Amjad et al. in [5] investigated the problem of IB with respect to deterministic deep neural networks. The first problem that they addressed is IB in practical cases is either infinity or piecewise constant function of the set of parameters. In Theorem 3.3.2, they proved if X and T both have continuous distribution, $I(X, T)$ is infinity.

Theorem 3.3.2. *[5] Let $X = L_0$ be an N-dimensional RV, the distribution of which has an absolutely continuous component with a probability density function f_X that is continuous on a compact set $\mathcal{X}$ in $\mathbb{R}^N$. Suppose that the activation function σ is either bi-Lipschitz or continuously differentiable with strictly positive derivative. Then, for every $i = 1, \cdots, m$ and almost every choice of weight matrices $\boldsymbol{W}_0, \cdots, \boldsymbol{W}_i$, we have*

$$I(X; L_{i+1}) = \infty \tag{3.25}$$

T can have continuous distribution if the activation function is selected as sigmoid, or tanh. If one selects activation function as ReLU or step activation function the problem is different, IB is finite but it becomes piecewise constant. The problem of being piecewise constant function is the optimization of it is difficult. In this case gradient is almost zero everywhere, so one has to use the optimization heuristic which are not gradient based. The second problem that they addressed is solving IB automatically doesn't solve classification problem and it is not robust to noise.

They proposed three solutions for these problems: forcing intermediate layer to be discrete; training stochastic deep neural networks; replacing IB functional by a more behaved cost function inspired by IB framework.

For the first solution they proposed to use a decision rule such that a fixed function $\delta : \mathbb{R}^{|\tilde{Y}|} \to \mathcal{Y}$ and $\widehat{Y} = \delta(\tilde{Y})$ when the optimization problem is

$$\min_{\theta_i} I(X, \widehat{Y}) - \beta I(Y, \widehat{Y}) \tag{3.26}$$

In this case compression is done by decision rule δ and the term $I(X, \widehat{Y})$ is useless. For the latent representation T, one has $\delta : \mathbb{R}^{|T|} \to \mathcal{Y}$ and minimizing equation (3.26) for $\widehat{Y} = \delta(T)$ [5].

Another solution is to use stochastic deep neural networks rather than deterministic ones. This can be achieved by injecting noise to intermediate representations. Their final solution is applying quantization over X and T before calculating optimization problem.

$$\min_{\theta_i} \ \overline{I}_C(X,T) - \beta \overline{I}_P(Y,T) \tag{3.27}$$

where

$$\begin{aligned} \overline{I}_C(X,T) &= I(Q_X(X), Q_T(T)) \\ \overline{I}_P(Y,T) &= I(Y, Q'_T(T)) \end{aligned} \tag{3.28}$$

and $Q_X(X)$, $Q_T(T)$, and $Q'_T(T)$ are quantizers that are adopted according to statistics of T and with respect to one another, and they only perform for computing (3.27). The quantization prevents that $\overline{I}_C(X,T)$ becomes infinite. And good choice of $Q'_T(T)$ causes simpler representations.

Another formulation of IB was proposed by Kolchinsky et al. [48]. IB Lagrangian optimization has difficulty in finding mutual information which can involve intractable integrals. Therefore, they proposed an approach which is different from the variational approaches that are suggested in [9, 36, 43, 4, 2, 23]. They proposed a general setting for IB which is called nonlinear IB [48]. In this method, T is considered as a continuous random variable while X and Y can be either continuous or discrete values and with any desired joint distribution. In addition, in this setting both encoding and decoding maps can be nonlinear.

3.4 Mutual Information Neural Estimator

Recently Belghazi [7] presented a new approach, which he calls the Mutual Information Neural Estimator (MINE), for estimating the regularization part in the Information Bottle-

neck method [74] in a continuous setting. This approach is scalable, flexible, and completely trainable via backward propagation.

Suppose that $\mathcal{U}$ and $\mathcal{V}$ are two spaces with a joint distribution $p_{UV}(u,v)$ on $\mathcal{U} \times \mathcal{V}$ defining a pair (U,V) of random variables. Assume we can perform i.i.d. sampling of $p_{UV}(u,v)$ and we wish to estimate the mutual information $I(U;V)$ from the samples. In the framework of MINE, a family Γ of functions is constructed as a neural network, where each $\gamma \in \Gamma$ is a function mapping $\mathcal{U} \times \mathcal{V}$ to the set $\mathbb{R}$ of real numbers. Then due to the dual representation of KL-divergence [17], the mutual information $I(U;V)$ can be estimated as

$$\widehat{I}(U;V) := \max_{\gamma \in \Gamma} \{\mathbb{E}_{(uv)\sim p_{UV}(u,v)} \gamma(u,v) - \log \mathbb{E}_{(u,v)\sim p_U(u)\otimes p_V(v)} \exp(\gamma(u,v))\}. \tag{3.29}$$

The authors showed experimentally the efficiency of this estimator [7].

3.5 Summary

In this chapter, we explained the IB principle which has been applied in different domains such as coding theory and quantization [8, 51, 13], speech, and image recognition [80, 32, 77, 81]. We also surveyed the related research carried out to solve the IB optimization problem.

Chapter 4

The Equivalence Between Information Bottleneck Learning and Information Bottleneck Quantization

4.1 Introduction

In this chapter, we explain the information bottleneck learning problem in Section 4.2. Then in Section 4.3, we explain the information bottleneck quantization problem, and we illustrate the equivalence between these two problems. In order to achieve this objective, we introduce two theorems, 4.3.1 and 4.3.2. Establishing equivalence between information bottleneck learning problem and information bottleneck quantization problem is the key point for reaching our goal in this study. Finally, we summarize this chapter in Section 4.5.

4.2 Information Bottleneck Learning

In the context of classification, we assume that objects and labels are distributed according to an unknown distribution $p_{XY}(x, y)$ on $\mathcal{X} \times \mathcal{Y}$ where we are given a set $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_N, Y_N)\}$ of i.i.d samples from $p_{XY}(x, y)$. The objective of learning in this setting is to find a classifier from $\mathcal{D}$ that classifies X into its label Y.

Finding such a representation of X which only contains all information about X relevant to its class label Y is a representation learning problem. Such a problem can be naturally formulated using the information bottleneck principle [74] and will be referred to as the *Information Bottleneck (IB) learning* problem.

The main goal in IB learning is to learn a representation T of X in some space $\mathcal{T}$ such that the mutual information $I(X;T)$ between X and T is as small as possible, while at the same time ensuring that the mutual information $I(Y;T)$ between T and the class label Y is as large as possible. In other words, this principle tends to squeeze away all information in X that is irrelevant to the classification task while keeping the relevant information intact. Intuitively, minimizing $I(X;T)$ forces the model not to over-fit to the irrelevant features of X, whereas maximizing $I(Y;T)$ extracts all features useful for the classification task. Therefore, the learning process exhibits two phases, first fitting then compressing. These two optimization objectives are in conflict with each other. This dichotomy allows us to formulate the IB learning problem as a constrained optimization problem, in which one objective is the optimization objective and the other is a constraint, subject to the Markov chain $Y - X - T$:

$$\widehat{p}_{T|X}(t|x) = \arg \min_{p_{T|X}(t|x):I(X;T)\leq A} -I(Y;T), \tag{4.1}$$

for a nonnegative value A or equivalently,

$$\widehat{p}_{T|X}(t|x) = \arg \min_{p_{T|X}(t|x):I(Y;T)\geq A'} I(X;T), \tag{4.2}$$

for a nonnegative value A'. The Markov chain assumption ensures that any information in feature T about label Y is obtained from X only. For later use, we denote the minimum mutual information in equation (4.2) as IB learning rate, $R_{\mathrm{IBL}}(A')$, i.e.,

$$R_{\mathrm{IBL}}(A') = \min_{p_{T|X}(t|x):I(Y;T)\geq A'} I(X;T). \tag{4.3}$$

The IB learning problem can then be regarded as, for a given A', finding $p_{T|X}(t|x)$ that achieves $R_{\mathrm{IBL}}(A')$.

We note that solving this IB learning problem, i.e., obtaining the optimal $\widehat{p}_{T|X}(t|x)$ and its corresponding bottleneck representation T, does not automatically solve the classification

problem in its entirety. A classifier still needs to be built that predicts the class label Y based on the representation T of X. Nonetheless later in this thesis, we will show that solving a variational approximation of the IB learning problem may in fact provide a direct solution to the classification problem of interest.

4.3 Information Bottleneck Quantization

In this section, we formulate the *Information Bottleneck (IB) quantization* problem.

Let $(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)$ be drawn i.i.d from $p_{XY}(x, y)$. The sequences $(X_1, X_2, .., X_n)$ and $(Y_1, Y_2, \cdots, Y_n)$ are denoted by X^n and Y^n, respectively.

An $(n, 2^{nR})$ *IB-quantization code* is a pair (f_n, g_n) in which f_n maps each sequence X^n to an integer in $\{1, 2, \cdots, 2^{nR}\}$ and g_n maps an integer in $\{1, 2, \cdots, 2^{nR}\}$ to a sequence $T^n := (T_1, T_2, \cdots, T_n) \in \mathcal{T}^n$. Using the standard nomenclature in quantization, the quantity R is referred to as the *rate* of the code and n as the *length* of the code. Using this code, f_n encodes the sequence X^n as the integer $f_n(X^n)$ and g_n reconstructs X^n as a representation $T^n := g_n(f_n(X^n))$, when T^n is used to predict Y^n as accurately as possible.

In the context of IB quantization, an IB-quantization code (f_n, g_n) denotes a *vector quantizer* when $n > 1$, and a scalar quantizer when $n = 1$. The difference between these two types of quantizers is shown in Figure 4.1. A scalar quantizer is shown on the left side of Figure 4.1 quantizing Gaussian random variables (X_1, X_2) using 2 bits per X-symbol and distortion is measured using the squared error of reconstructing (X_1, X_2). The square is the plane on which (X_1, X_2) lives. An optimal scalar quantizer quantizes each of X_1 and X_2 into two bits. Based on rate-distortion theory this is not optimal and hence a fundamental limitation of scalar quantization. In a vector quantizer on the other hand, (shown on the right side of Figure 4.1), one quantizes n symbols $X_1, X_2, \cdots, X_n$ at once using nR bits [14]. In other words, a vector quantizer partitions the plane into 16 regions and assigns a 4-bit string to each region. The vector quantizer can better exploit the joint distribution of (X_1, X_2) and the freedom in partitioning the plane. Thus it achieves smaller average distortion.

Intuitively, encoding the components of a sequence independently as in scalar quantization should be inferior to the joint encoding of the sequence as in vector quantization since the latter provides a richer family of encoders and decoders, i.e., more freedom in dividing the source domain into reconstruction regions [14, 27, 21].

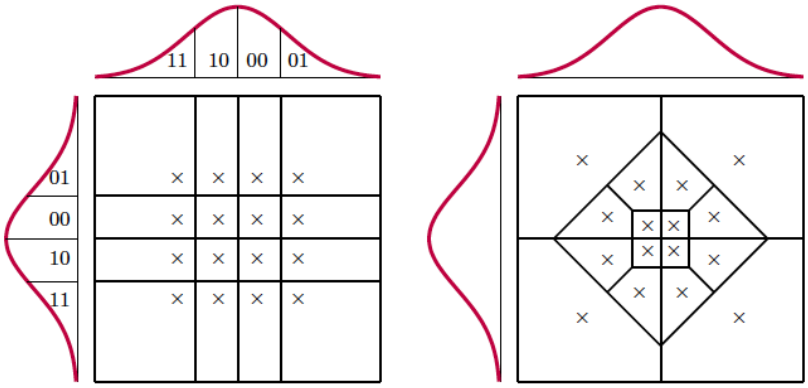


Figure 4.1: Left: scalar quantizer. Right: vector quantizer. $(n = 2)$

Unlike the standard quantization problems, the IB quantization problem uses a distortion measure that may depend on the code. To that end, for any $x \in \mathcal{X}$, $t \in \mathcal{T}$ and for any two conditional distributions $q_{Y|X}(y|x)$ and $q_{Y|T}(y|t)$, we define

$$d_{\mathrm{IB}}(x, t; q_{Y|X}(y|x), q_{Y|T}(y|t)) := \mathrm{KL}(q_{Y|X}(.|x) \| q_{Y|T}(.|t)), \tag{4.4}$$

where $\mathrm{KL}(.\|.)$ denotes the Kullback-Leibler (KL) divergence.

Note that the code (f_n, g_n), together with $p_{XY}(x, y)$, induce a joint distribution over the Markov chain Y^n — X^n — T^n. Under this joint distribution the conditional distributions $p_{Y_i|X_i}(y_i|x_i)$ and $p_{Y_i|T_i}(y_i|t_i)$ are well defined for each $i = 1, 2, ..., n$. Hence, given the code (f_n, g_n) and for any two sequences $x^n \in \mathcal{X}^n$ and $t^n \in \mathcal{T}^n$, their *IB distortion* is defined as:

$$\overline{d}_{\mathrm{IB}}(x^n, t^n) := \frac{1}{n} \sum_{i=1}^{n} d_{\mathrm{IB}}(x_i, t_i; p_{Y_i|X_i}(y_i|x_i), p_{Y_i|T_i}(y_i|t_i)), \tag{4.5}$$

We note that the quantity $\overline{d}_{\mathrm{IB}}(x^n, t^n)$ measures a "loss of information about Y" when the code (f_n, g_n) is used to represent x^n as t^n. Specifically, consider the source coding problem of compressing Y^n based on observing X^n. In other words, if the conditional distribution

$p_{Y_i|X_i}(\cdot|x_i)$ for each i is mistaken as $p_{Y_i|T_i}(\cdot|t_i)$ in the design of the source code, the average additional coding overhead per Y-symbol is precisely $\bar{d}_{\text{IB}}(x^n, t^n)$.

Using this distortion measure, the *IB quantization problem* is to find a code (f_n, g_n) having the smallest rate R subject to the constraint $\mathbb{E}\bar{d}_{\text{IB}}(X^n, T^n) \leq D$, where $\mathbb{E}$ denotes expectation. For given $p_{XY}(x, y)$ and $\mathcal{T}$, a rate distortion pair (R, D) is called *achievable* if $\mathbb{E}\bar{d}_{\text{IB}}(X^n, T^n) \leq D$ for some sequence of (f_n, g_n) codes. As usual, the *rate-distortion* function for the IB quantization problem, which we denote by $R_{\text{IBQ}}(D)$, is defined as the smallest rate R such that (R, D) is achievable. Theorem 4.3.1 characterizes this rate-distortion function.

Theorem 4.3.1. *Given $p_{XY}(x, y)$ and $\mathcal{T}$, the rate-distortion function for the IB quantization problem can be written as*

$$R_{\text{IBQ}}(D) = \min_{p_{T|X}(t|x):\mathbb{E}d_{\text{IB}}(X,T)\leq D} I(X;T) \tag{4.6}$$

where the expectation is over the Markov chain $Y - X - T$ specified by $p_{XY}(x, y)$ and $p_{T|X}(t|x)$ and is defined as

$$\mathbb{E}d_{\text{IB}}(X, T) := \sum_{(x,t)\in\mathcal{X}\times\mathcal{T}} d_{\text{IB}}(x, t; p_{Y|X}(y|x), p_{Y|T}(y|t))p_{XT}(x, t) \tag{4.7}$$

A proof of this theorem is given in Section 4.4. This theorem provides a limit on the achievable rates of the IB quantization problem. We note that a similar result was first shown in [64]. However in [64], the result relies on the assumption that $|\mathcal{T}| \geq |\mathcal{X}| + 2$, whereas in this theorem that condition is removed. Equivalently, this theorem can also be stated in terms of the distortion-rate function in which the smallest distortion is achievable for a given rate R:

$$D(R_{\text{IBQ}}) = \min_{p_{T|X}(t|x):I(X;T)\leq R} \mathbb{E}d_{\text{IB}}(X, T) \tag{4.8}$$

The form of the rate-distortion function R_{IBQ} for the IB quantization problem given in Theorem 4.3.1 resembles greatly the optimal objective R_{IBL} in equation (4.3). In fact, simple manipulation of the constraints of equation (4.3) and equation (4.6) gives rise to the following theorem:

Theorem 4.3.2. $R_{\mathrm{IBL}}(A') = R_{\mathrm{IBQ}}(I(X;Y) - A')$

Proof.

For any distribution $p_{YXT}(y,x,t)$ defining a Markov chain $Y \multimap X \multimap T$, we have

$$\begin{aligned}
\mathbb{E}d_{\mathrm{IB}}(X,T) &= \sum_{(x,t)\in\mathcal{X}\times\mathcal{T}} d_{\mathrm{IB}}(x,t;p_{Y|X}(y|x),p_{Y|T}(y|t))p_{XT}(x,t) \\
&= \sum_{(x,t)\in\mathcal{X}\times\mathcal{T}} \sum_{y\in\mathcal{Y}} p_{Y|X}(y|x)\log\left(\frac{p_{Y|X}(y|x)}{p_{Y|T}(y|t)}\right) p_{XT}(x,t) \\
&= \sum_{(y,x,t)\in\mathcal{Y}\times\mathcal{X}\times\mathcal{T}} p_{YXT}(y,x,t)\log\frac{p_{Y|X}(y|x)}{p_{Y|T}(y|t)} \\
&= \sum_{(y,x,t)\in\mathcal{Y}\times\mathcal{X}\times\mathcal{T}} p_{YXT}(y,x,t)\log\left(\frac{p_{Y|X}(y|x)}{p_Y(y)}\cdot\frac{p_Y(y)}{p_{Y|T}(y|t)}\right) \\
&= \sum_{(y,x,t)\in\mathcal{Y}\times\mathcal{X}\times\mathcal{T}} p_{YXT}(y,x,t)\log\frac{p_{Y|X}(y|x)}{p_Y(y)} \\
&+ \sum_{(y,x,t)\in\mathcal{Y}\times\mathcal{X}\times\mathcal{T}} p_{YXT}(y,x,t)\log\frac{p_Y(y)}{p_{Y|T}(y|t)} \\
&= I(X;Y) - I(T;Y) \qquad (4.9)
\end{aligned}$$

where the first two equalities follow from the definitions of the expectation and d_{IB}. The third equality, follows from the Markov structure of $Y \multimap X \multimap T$. The remaining equalities follow simple manipulations and the definition of mutual information. Thus, we may rewrite equation (4.6) in Theorem 4.3.1 as

$$\begin{aligned}
R_{\mathrm{IBQ}}(D) &= \min_{p_{T|X}:I(X;Y)-I(T;Y)\le D} I(X;T) \\
&= \min_{p_{T|X}:I(Y;T)\ge I(X;Y)-D} I(X;T) \\
&= R_{\mathrm{IBL}}(I(X;Y) - D) \qquad (4.10)
\end{aligned}$$

The theorem then follows via the substitution $A' := I(X;Y) - D$. ■

This theorem relates the IB learning problem to the IB quantization problem, where we note that $I(X;Y)$ is a constant that depends only on $p_{XY}(x,y)$. By this theorem, solving the IB learning problem where the information about Y contained in T needs to be no less than A' is equivalent to solving the IB quantization problem so that the distortion is no more than $I(X;Y) - A'$.

In the next chapter, we develop a variational approach to solve the IB Learning problem. Based on this equivalence and using the variational approach, we propose a novel framework for learning neural network classifiers, that we call "Aggregated Learning (AgrLearn)".

4.4 Proof of Theorem 4.3.1

Here we give a brief review of typical sequences [58], which will be useful in proving Theorem 4.3.1. We remark that the notion of typicality here is stronger than the widely used (weak) typicality in [14]. A comprehensive treatment of the subject is given in [18]. In some places, we might use subscripts to explicitly indicate the random variables with respect to which the expectation is performed.

1. Empirical distribution: For a given sequence $x^n \in \mathcal{X}^n$, is defined as
$$\pi(x|x^n) := \frac{1}{n}|\{i : x_i = x\}| \text{ for all } x \in \mathcal{X} \tag{4.11}$$

2. Typical set: For $X \sim p_X(x)$ and $\epsilon \in (0,1)$, the set of ϵ-typical sequences is defined as
$$\mathcal{S}_\epsilon^n(X) :=\{x^n||\pi(x|x^n) - p_X(x)| \leq \epsilon p_X(x) \text{ for all } x \in \mathcal{X}\} \tag{4.12}$$

3. Typical average lemma: For any $x^n \in \mathcal{S}_\epsilon^n(X)$ and any nonnegative function g on $\mathcal{X}$, we have
$$(1-\epsilon)\mathbb{E}[g(X)] \leq \frac{1}{n}\sum_{i=1}^{n} g(x_i) \leq (1+\epsilon)\mathbb{E}[g(X)] \tag{4.13}$$
Note that by choosing g to be the log function, one recovers the notion of typicality in [14]. The typicality here is stronger than the one in [14], however, similar to weak typicality, most i.i.d. sequences are still typical under this definition. Namely, for

any i.i.d sequence X^n of random variables with $X_i \sim p_X(x_i)$, by the Law of Large Numbers (LLN), the empirical distribution $\pi(x|X^n)$ converges (in probability) to $p_X(x)$, for all $x \in \mathcal{X}$, and so such a sequence, with high probability, belongs to the typical set.

4. Joint typicality: Items 1 and 2 extend to a joint source $(X, Y) \sim p_{XY}(x, y)$ in the obvious way, i.e., by treating X and Y as one source (X, Y). Given a sequence $(x^n, y^n) \in \mathcal{X}^n \times \mathcal{Y}^n$, it induces an empirical distribution on $\mathcal{X} \times \mathcal{Y}$ defined as

$$\pi(x, y|x^n, y^n) := \frac{1}{n}|\{i : x_i = x, y_i = y\}| \text{ for all } (x, y) \in \mathcal{X} \times \mathcal{Y} \tag{4.14}$$

For $X \sim p_X(x)$ and $\epsilon \in (0, 1)$, the set of ϵ-typical sequences is defined as

$$\mathcal{S}^n_\epsilon(X, Y) := \{(x^n, y^n) || \pi(x, y|x^n, y^n) - p_{XY}(x, y)| \leq \epsilon p_{XY}(x, y) \text{ for all } (x, y) \in \mathcal{X} \times \mathcal{Y}\} \tag{4.15}$$

5. Joint typicality lemma: Let $(X, Y) \sim p_{XY}(x, y)$ and $p_Y(y)$ be the marginal distribution $\sum_{x \in \mathcal{X}} p_{XY}(x, y)$. Then, for $\epsilon' < \epsilon$, there exists $\delta(\epsilon) \to 0$ as $\epsilon \to 0$ such that

$$p\{(x^n, Y^n) \in \mathcal{S}^n_\epsilon(X, Y)\} \geq 2^{-n(I(X;Y)+\delta(\epsilon))} \tag{4.16}$$

for $x^n \in \mathcal{S}^n_{\epsilon'}$, $Y^n \sim \prod_{i=1}^n p_Y(y_i)$, and sufficiently large n.

We should make a few remarks before presenting a proof for Theorem 4.3.1. The proof follows standard techniques from information theory for proving the results of this nature. It is worth noting that the conventional proof of achievability [14] of the rate-distortion theorem does not directly apply here since the distortion measure d_{IB} depends on the distribution $p_{T|X}(t|x)$. This was addressed in [64] by extending the definition of distortion of jointly typical sequences to multi-distortion of jointly typical sequences [14]. Our approach exploits the notion of typicality which is presented and closely follows the proof of achievability in [18] of the rate-distortion theorem

$$R'(D) := \min_{p_{T|X}(t|x):\mathbb{E}[d(X,T)] \leq D} I(X; T) \tag{4.17}$$

We need to show $R_{\mathrm{IBQ}}(D) = R'(D)$.

Proof of the converse in Theorem 4.3.1:

We first show $R_{\mathrm{IBQ}}(D) \geq R'(D)$ by showing that for any sequence of $(n, 2^{nR})$ codes satisfying $\mathbb{E}\bar{d}_{\mathrm{IB}}(X^n, T^n) \leq D$, it must be the case that $R \geq R'(D)$. We have

$$\begin{aligned}
nR &\overset{\text{(i)}}{\geq} H(f_n(X^n)) \\
&\overset{\text{(ii)}}{\geq} H(f_n(X^n)) - H(f_n(X^n)|X^n) = I(X^n; f_n(X^n)) \\
&\overset{\text{(iii)}}{\geq} I(X^n, T^n) = H(X^n) - H(X^n|T^n) \\
&\overset{\text{(iv)}}{=} \sum_{i=1}^{n} H(X_i) - H(X^n|T^n) \\
&\overset{\text{(v)}}{=} \sum_{i=1}^{n} H(X_i) - \sum_{i=1}^{n} H(X_i|T^n, X_{i-1}, \cdots, X_1) \\
&\overset{\text{(vi)}}{\geq} \sum_{i=1}^{n} H(X_i) - \sum_{i=1}^{n} H(X_i|T_i) \\
&= \sum_{i=1}^{n} I(X_i; T_i) \\
&\overset{\text{(vii)}}{\geq} \sum_{i=1}^{n} R'(\mathbb{E}[d(X_i, T_i)]) \\
&= n(\frac{1}{n}\sum_{i=1}^{n} R'(\mathbb{E}[d(X_i, T_i)])) \\
&\overset{\text{(viii)}}{\geq} nR'(\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[d(X_i, T_i)]) \\
&\overset{\text{(ix)}}{=} nR'(\mathbb{E}[d(X^n, T^n)]) \\
&\overset{\text{(x)}}{\geq} nR'(D)
\end{aligned} \tag{4.18}$$

where (i) follows from the fact that f_n takes its values from $\{1, \ldots, 2^{nR}\}$,

(ii) from the non-negativity of conditional entropy,

(iii) from the data processing inequality since $T^n = g_n(f_n(X^n))$,

(iv) from the fact that X_i are independent,

(v) from the chain rule for entropy,

(vi) from the conditioning reduces entropy,

(vii) from equation (4.17) by noting that $R'(\mathbb{E}[d(X_i, T_i)]) = \min_{p_{T_i|X_i}} I(X_i; T_i)$,

(ix) from equation (4.5),

(x) from $\mathbb{E}\bar{d}_{\mathrm{IB}}(X^n, T^n) \leq D$ since $R'(D)$ is a decreasing function in D.

To prove (viii), it is sufficient to show that R' is a convex function in D, which is shown in the following lemma.

Lemma 1. *[3]. The function $R'(D)$ defined in equation (4.17) is a convex function.*

Proof.

In this lemma, we show $R'(\lambda D_1 + (1-\lambda)D_2) \leq \lambda R'(D_1) + (1-\lambda)R'(D_2)$ for $\forall \lambda \in (0,1)$, where (D_1, R_1) and (D_2, R_2) are two points on $R'(D)$ attained, respectively, by T_1 and T_2 via the minimizers $p_{T_1|X}(t_1|x)$ and $p_{T_2|X}(t_2|x)$ of equation (4.17):

$$P_{T_1|X} := \arg \min_{p_{T_1|X}(t_1|x):\mathbb{E}[d(X,T_1)]\leq D_1} I(X; T_1) \tag{4.19}$$

$$P_{T_2|X} := \arg \min_{p_{T_2|X}(t_2|x):\mathbb{E}[d(X,T_2)]\leq D_2} I(X; T_2). \tag{4.20}$$

We construct a new random variable $U = (T, Z)$ such that drawing Z from a Bernoulli distribution and

$$T = \begin{cases} T_1, & \text{if } Z = 1 \\ T_2, & \text{if } Z = 2 \end{cases} \tag{4.21}$$

where $Z \in \{1, 2\}$ is a random variable independent of (T_1, T_2, X, Y) with $p_Z(1) = \lambda$. Then,

$$p_{XTZ}(x, t, z) = \begin{cases} \lambda p_{XT_1}(x, t_1), & \text{if } Z = 1 \\ (1-\lambda) p_{XT_2}(x, t_2), & \text{if } Z = 2 \end{cases} \tag{4.22}$$

and so

$$
\begin{aligned}
I(X;(T,Z)) &= \sum_{(x,t,z)\in\mathcal{X}\times\mathcal{T}\times\mathcal{Z}} p_{XTZ}(x,t,z)\log\frac{p_{XTZ}(x,t,z)}{p_X(x)p_{TZ}(t,z)} \\
&= \sum_{(x,t_1)\in\mathcal{X}\times\mathcal{T}} \lambda p_{XT_1}(x,t_1)\log\frac{\lambda p_{XT_1}(x,t_1)}{\lambda p_X(x)p_{T_1}(t_1)} \\
&+ \sum_{(x,t_2)\in\mathcal{X}\times\mathcal{T}} (1-\lambda)p_{XT_2}(x,t_2)\log\frac{(1-\lambda)p_{XT_2}(x,t_2)}{(1-\lambda)p_X(x)p_{T_2}(t_2)} \\
&= \lambda\ I(X;T_1) + (1-\lambda)I(X;T_2)
\end{aligned} \tag{4.23}
$$

Moreover, we have

$$
\begin{aligned}
\mathbb{E}[d(X,(T,Z))] &= \sum_{(x,t,z)\in\mathcal{X}\times\mathcal{T}\times\mathcal{Z}} p_{XTZ}(x,t,z)\mathrm{KL}(p_{Y|X}(.|x)\|p_{Y|TZ}(.|t,z)) \\
&= \sum_{(x,t,z)\in\mathcal{X}\times\mathcal{T}\times\mathcal{Z}} p_{XTZ}(x,t,z)\sum_{y\in\mathcal{Y}} p_{Y|X}(y|x)\log\frac{p_{Y|X}(y|x)}{p_{Y|TZ}(y|t,z)} \\
&= H(Y|T,Z) - H(Y|X) \\
&= -H(Y|X) + \lambda H(Y|T_1) + (1-\lambda)H(Y|T_2) \\
&= \lambda H(Y|T_1) + (1-\lambda)H(Y|T_2) - \lambda H(Y|X) - (1-\lambda)H(Y|X) \\
&= \lambda\mathbb{E}[d(X,T_1)] + (1-\lambda)\mathbb{E}[d(X,T_2)]
\end{aligned} \tag{4.24}
$$

Since (T,Z)—X—Y is a Markov chain resulting in cost and constraint which are linear functions of the original costs and constraints, the claim follows from the definition of R' in equation (4.17). ■

Proof of Achievability in Theorem 4.3.1:

We need to show that for $R = R'(D)$ there exists a sequence $(2^{nR}, n)$ of codes satisfying $\mathbb{E}\bar{d}_{\mathrm{IB}}(X^n, T^n) \leq D$.

Random codebook: Let $R = R'(D)$ and fix $p_{T|X}(t|x)$ to be an optimal distribution to the minimization (4.17) at $D/(1+\epsilon)$, i.e., we pick a conditional distribution that

attains $R'(D/(1+\epsilon))$. Let $p_T(t) = \sum_{x\in\mathcal{X}} p_X(x)p_{T|X}(t|x)$. Generate 2^{nR} i.i.d. sequences $t^n(m) \sim \prod_{i=1}^n p_T(t_i)$, $m \in \{1,\dots,2^{nR}\}$. These sequences form the codebook which is revealed to the encoder and decoder.

Encoder: The encoder uses joint typicality encoding. Given a sequence x^n, find an index m such that $(x^n, t^n(m)) \in \mathcal{S}_\epsilon^n(X,T)$ and send m. If there is more than one index then choose m to be the smallest index, and if there is no index then choose $m = 1$. In other words, the encoder sets $f_n(x^n)$ to be the index m, where m is as described above.

Decoder: Upon receiving index m, set $t^n = t^n(m)$. In other words, the decoder sets $g_n(m)$ to be the row of the codebook indexed by m.

Expected distortion: Let $\epsilon' < \epsilon$ and M be the index chosen by the encoder. We first bound the distortion averaged over codebooks. Towards this end, define the event

$$\mathcal{E} := \{(X^n, T^n(m)) \notin \mathcal{S}_\epsilon^n(X,T)\} \tag{4.25}$$

then by the union bound and the choice of the encoder, we have

$$p(\mathcal{E}) \le p(\mathcal{E}_1) + p(\mathcal{E}_2) \tag{4.26}$$

where

$$\begin{aligned}
\mathcal{E}_1 &:= \{X^n \notin \mathcal{S}_{\epsilon'}^n(X)\},\\
\mathcal{E}_2 &:= \{X^n \in \mathcal{S}_{\epsilon'}^n, (X^n, T^n(m)) \notin \mathcal{S}_\epsilon^n(X,T) \ \ \forall m \in \{1,\dots,2^{nR}\}\}
\end{aligned} \tag{4.27}$$

We have $\lim_{n\to\infty} p(\mathcal{E}_1) = 0$ by the LLN and

$$\begin{aligned}
p(\mathcal{E}_2) &= \sum_{x^n\in\mathcal{S}_{\epsilon'}^n} p_{X^n}(x^n) p_{X^nT^n}\big\{(x^n, T^n(m)) \notin \mathcal{S}_\epsilon^n \ \forall m \mid X^n = x^n\big\}\\
&\overset{\text{(i)}}{=} \sum_{x^n\in\mathcal{S}_{\epsilon'}^n} p_{X^n}(x^n) \prod_{m=1}^{2^{nR}} p_{X^nT^n}\big\{(x^n, T^n(m)) \notin \mathcal{S}_\epsilon^n\big\}\\
&\overset{\text{(ii)}}{=} \sum_{x^n\in\mathcal{S}_{\epsilon'}^n} p_{X^n}(x^n)\big(p_{X^nT^n}\big\{(x^n, T^n(1)) \notin \mathcal{S}_\epsilon^n\big\}\big)^{2^{nR}}
\end{aligned}$$

$$\overset{\text{(iii)}}{\leq} \sum_{x^n \in \mathcal{S}^n_{\epsilon'}} p_{X^n}(x^n)\big(1 - 2^{-nI(X;T)+\delta(\epsilon)}\big)^{2^{nR}}$$

$$\leq \big(1 - 2^{-nI(X;T)+\delta(\epsilon)}\big)^{2^{nR}}$$

$$\overset{\text{(iv)}}{\leq} \exp\big(-2^{n(R-I(X;T)-\delta(\epsilon))}\big) \tag{4.28}$$

where (i) and (ii) are by the i.i.d assumption on the codewords, (iii) is by the joint typicality lemma equation (4.16), (iv) is by the fact $(1-\alpha)^k \leq \exp(-k\alpha)$ for $\alpha \in [0,1]$ and $k \geq 0$. Hence, we have $\lim_{n\to\infty} p(\mathcal{E}_2) = 0$ for $R > I(X;T) + \delta(\epsilon)$.

Now, the distortion averaged over X^n and over the random choice of the codebook is given as

$$\begin{aligned}
\mathbb{E}_{X^n,T^n,M}[d(X^n, T^n(M))] &= p(\mathcal{E}) \cdot \mathbb{E}_{X^n,T^n,M}[d(X^n, T^n(M))|\mathcal{E}] \\
&\quad + p(\mathcal{E}^c) \cdot \mathbb{E}_{X^n,T^n,M}[d(X^n, T^n(M))|\mathcal{E}^c] \\
&\leq p(\mathcal{E}) \cdot d_{\max} + p(\mathcal{E}^c) \cdot \mathbb{E}_{X^n,T^n,M}[d(X^n, T^n(M))|\mathcal{E}^c] \\
&= p(\mathcal{E}) \cdot d_{\max} + p(\mathcal{E}^c) \cdot \mathbb{E}_{X^n,T^n}[d(X^n, T^n(1))|\mathcal{E}^c] \\
&\leq p(\mathcal{E}) \cdot d_{\max} + p(\mathcal{E}^c) \cdot (1+\epsilon)\mathbb{E}_{X,T}[d(X,T)]
\end{aligned} \tag{4.29}$$

where $d_{\max} = \max_{(x,t)\in\mathcal{X}\times\mathcal{T}} d(x,t)$. By the choice of $p_{T|X}(t|x)$, we have $\mathbb{E}_{X,T}[d(X,T)] \leq D/(1+\epsilon)$, and so

$$\lim_{n\to\infty} \mathbb{E}_{X^n,T^n,M}[d(X^n, T^n(M))] \leq D \tag{4.30}$$

for $R > I(X,T) + \delta(\epsilon)$, where $\delta(\epsilon) \to 0$ as $n \to \infty$. Since the expected distortion, averaged over codebooks, satisfies the distortion constraint D, there must exist a sequence of codes that satisfies the constraint. This shows the achievability of the rate-distortion pair $(R(D/(1+\epsilon)) + \delta(\epsilon), D)$. By the continuity of $R(D)$ in D the achievable rate $R(D/(1+\epsilon)) + \delta(\epsilon)$ converges to $R(D)$ as $\epsilon \to 0$.

4.5 Summary

In this chapter, we explained the IB learning problem and the IB quantization problem. We proved that the objective of IB quantization, namely, designing quantizers that achieve the rate-distortion limit, is equivalent to the objective of IB learning. This result establishes an equivalence between the two problems.

Chapter 5

The Aggregated Learning Framework

5.1 Introduction

Having established the equivalence between Information Bottleneck (IB) learning and Information Bottleneck (IB) quantization in the previous chapter, in this chapter we propose a novel way of solving the IB learning problem. In Section 5.2 we demonstrate a variational approach to the IB learning problem. In Section 5.3, based on the equivalence between IB learning and IB quantization and using the variational approach, we propose a novel framework for learning neural networks, that we call *Aggregated Learning (AgrLearn)*. We formulate two classes of AgrLearn, *deterministic AgrLearn (dAgrLearn)* and *probabilistic AgrLearn (pAgrLearn)*. We explain the training and testing of AgrLearn framework. Finally, we summarize this chapter in Section 5.4.

5.2 Variational Approach to Information Bottleneck Learning

Having established the equivalence between IB learning and IB quantization, we now solve the IB learning problem. The objective of this section is to develop a variational approach to this problem which not only provides a bottleneck representation T for X but also gives

rise to a classifier needed for the classification problem. We note that the result presented in this section also underlies the variational IB approach of [4]. We first establish the following result.

Theorem 5.2.1. *Under any distribution $p_{XYT}(x, y, t)$ that satisfies the Markov chain Y — X — T, we have*

$$I(Y;T) \geq \mathbb{E}_{\substack{(x,y)\sim p_{XY}(x,y),\\ t\sim p_{T|X}(\cdot|x)}} \log q_{Y|T}(y|t) + H(Y) \tag{5.1}$$

for any conditional distribution $q_{Y|T}(y|t)$ of a random variable on $\mathcal{Y}$ conditioned on T. In addition, the above inequality holds with equality if and only if $q_{Y|T}(y|t)$ is equal to $p_{Y|T}(y|t)$.

Proof.

$$\begin{aligned}
I(Y;T) &= \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log \frac{p_{Y|T}(y|t)}{p_Y(y)} \\
&= \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log \frac{p_{Y|T}(y|t)}{q_{Y|T}(y|t)} \cdot \frac{q_{Y|T}(y|t)}{p_Y(y)} \\
&= \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log q_{Y|T}(y|t) + \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log \frac{p_{Y|T}(y|t)}{q_{Y|T}(y|t)} \\
&\quad - \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log p_Y(y) \\
&= \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log q_{Y|T}(y|t) + \mathbb{E}_{t\sim p_T(t)} \mathrm{KL}\left(p_{Y|T}(\cdot|t) \| q_{Y|T}(\cdot|t)\right) + H(Y) \\
&\geq \sum_{(t,y)\in\mathcal{T}\times\mathcal{Y}} p_{YT}(y,t) \log q_{Y|T}(y|t) + H(Y) = \mathbb{E}_{\substack{(x,y)\sim p_{XY}(x,y),\\ t\sim p_{T|X}(\cdot|x)}} \log q_{Y|T}(y|t) + H(Y)
\end{aligned} \tag{5.2}$$

Note that the inequality above is due to the non-negativity of KL-divergence, in which equality is achieved precisely when $q_{Y|T}(y|t)$ is identically equal to $p_{Y|T}(y|t)$. ■

As a consequence of this theorem the mutual information $I(Y;T)$ can be written as

$$I(Y;T) = \max_{q_{Y|T}(y|t)} \mathbb{E}_{\substack{(x,y)\sim p_{XY}(x,y),\\ t\sim p_{T|X}(\cdot|x)}} \log q_{Y|T}(y|t) + H(Y). \tag{5.3}$$

Substituting this in the IB learning problem as formulated in equation (4.1), and removing $H(Y)$ since it is a constant, we have

$$\begin{aligned}\widehat{p}_{T|X}(t|x) &= \arg \min_{p_{T|X}(t|x):I(X;T)\leq A} -I(Y;T) \\ &= \arg \min_{p_{T|X}(t|x):I(X;T)\leq A} \left\{ - \max_{q_{Y|T}(y|t)} \mathbb{E}_{\substack{(x,y)\sim p_{XY}(x,y),\\ t\sim p_{T|X}(\cdot|x)}} \log q_{Y|T}(y|t) \right\} \\ &= \arg \min_{p_{T|X}(t|x):I(X;T)\leq A} \min_{q_{Y|T}(y|t)} \left\{ -\mathbb{E}_{\substack{(x,y)\sim p_{XY}(x,y),\\ t\sim p_{T|X}(\cdot|x)}} \log q_{Y|T}(y|t) \right\} \end{aligned} \tag{5.4}$$

Now suppose that we have a neural network representing the mapping $p_{T|X}(t|x)$ and we represent $q_{Y|T}(y|t)$ using another network. Then we may construct an overall network by concatenating the two networks. Specifically, each object X will be first passed to the network $p_{T|X}(t|x)$, the output T of which is passed to the network $q_{Y|T}(y|t)$. If the true class label Y is modeled as being generated from this concatenated network, it is easy to see that the cross-entropy loss ℓ_{CE} of the network is the expectation above, i.e.,

$$\ell_{\text{CE}} = -\mathbb{E}_{(x,y)\sim p_{XY}(x,y), t\sim p_{T|X}(\cdot|x)} \log q_{Y|T}(y|t). \tag{5.5}$$

In other words, the IB learning problem can be formulated as solving the following optimization problem:

$$\min_{p_{T|X}(t|x), q_{Y|T}(y|t)} \ell_{\text{CE}} \left(p_{T|X}(t|x), q_{Y|T}(y|t) \right) \text{ subject to } I(X;T) \leq A \tag{5.6}$$

Hence, introducing a Lagrange multiplier, will allow us to focus on the following unconstrained problem

$$\min_{p_{T|X}(t|x), q_{Y|T}(y|t)} \ell_{\text{CE}} \left(p_{T|X}(t|x), q_{Y|T}(y|t) \right) + \alpha I(X;T) \tag{5.7}$$

for nonnegative α.

An apparent advantage of this approach to IB learning is that, when the optimization problem (5.7) is solved, not only is the bottleneck representation T found, but also the entire classification network is obtained.

It is remarkable that the variational formulation (5.7) of IB learning can be viewed as a generalization of learning with standard neural networks under cross-entropy loss.

Specifically, learning with standard neural networks is a reduction of equation (5.7) in which the standard neural network contains no term $\alpha I(X;T)$, or equivalently has $\alpha = 0$.

The generalization of learning with standard neural networks to the formulation of IB learning in equation (5.7) is arguably beneficial in two respects.

1. The $\alpha I(X;T)$ regularization term in equation (5.7) serves to control the model complexity so as to reduce the generalization gap.

2. Generalizing the deterministic map from X to T in standard neural networks to a stochastic one in equation (5.7) minimizes the cross-entropy loss ℓ_{CE} over a larger space; this potentially allows further decrease of ℓ_{CE}, thereby achieving better classification accuracy.

In the remainder of this chapter, we present a new strategy, termed "Aggregated Learning", to implement the IB learning formulation (5.7).

5.3 Aggregated Learning (AgrLearn)

We now introduce the Aggregated Learning (AgrLearn) framework for learning with neural networks. We will stay with the IB learning formulation of equation (5.7) while keeping in mind that it results from a variational approximation of the formulation in equation (4.1).

Recall from Theorem 4.3.2 that the IB learning problem is equivalent to the IB quantization problem. In classical rate-distortion theory [67], it is well known that in order to achieve the rate-distortion limit of quantization, in general, one must consider the use of *vector quantizers*.

From rate-distortion theory, better quantizers result from using quantization codes with larger length n. In particular, in order to achieve the rate-distortion function, it is in general required that the length n of the rate-distortion code be made asymptotically large.

Note that a scalar IB-quantization code (f_1, g_1) maps X to T by

$$T = g_1(f_1(X)) := (g_1 \circ f_1)(X). \tag{5.8}$$

Under the equivalence between IB quantization and IB learning, the mapping $g_1 \circ f_1$ induced by the scalar quantizer (f_1, g_1) essentially defines a conditional distribution $p_{T|X}(t|x)$ in IB learning, which simply reduces to the deterministic function $g_1 \circ f_1$. On the other hand, in learning with a standard neural network, the deterministic mapping, say h, from the input space $\mathcal{X}$ to the bottleneck space $\mathcal{T}$ (which could refer to the space of feature representation at any intermediate layer of the network), can be regarded as implementing a scalar IB-quantization code (f_1, g_1) with

$$g_1 \circ f_1 = h. \tag{5.9}$$

The superiority of vector quantizers over scalar quantizers which we explained in Section 4.3 motivates us to develop a vector-quantization approach to IB learning, which we call Aggregated Learning or AgrLearn in short. Like a vector quantizer, which quantizes n signals simultaneously, AgrLearn classifies n input objects jointly at the same time. In other words, this aggregation effectively uses a stochastic mapping that maps each X to T, implicitly making optimization over a larger space than using a non-aggregated deterministic IB learning network.

The AgrLearn framework takes as its input the concatenation of n objects $(X_1, X_2, \ldots, X_n) := X^n$. Such a concatenated input will be referred to as an "n-fold aggregated input". The structure of this framework consists of two parts as seen in Figure 5.1. AgrLearn framework for fold-1 (when there is no aggregation among input objects) and fold-2 (when there is aggregation among two input objects) are depicted in Figures 5.2 and 5.3 respectively. The first part, or the "pre-bottleneck" module maps an aggregated input X^n to an "aggregated bottleneck" T^n.

Based on the type of mapping, two classes of AgrLearn, *deterministic AgrLearn (dAgrLearn)* and *probabilistic AgrLearn (pAgrLearn)* are formulated. The deterministic mapping leads to dAgrLearn, and probabilistic mapping leads to pAgrLearn. The details of dAgrLearn and pAgrLearn are given in the next section.

The second part, or the "post-bottleneck" module, implements a stochastic mapping

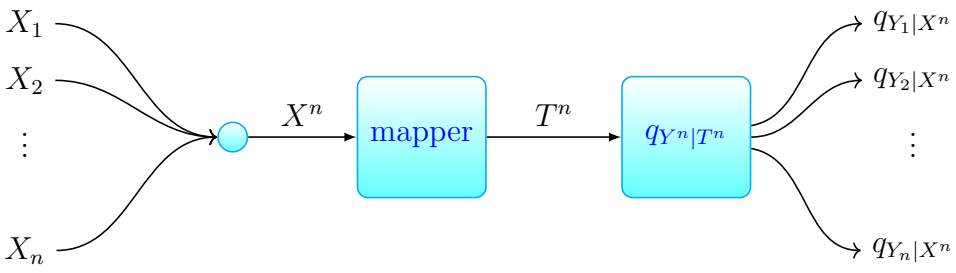


Figure 5.1: The structure of AgrLearn framework. The small circle denotes concatenation.

$q_{Y^n|T^n}(y^n|t^n)$ from $\mathcal{T}^n$ to $\mathcal{Y}^n$ that factorizes according to

$$q_{Y^n|T^n}(y^n|t^n) := \prod_{i=1}^{n} q_{Y_i|T^n}(y_i|t^n) \tag{5.10}$$

Overall the AgrLearn framework expresses a stochastic mapping from $\mathcal{X}^n$ to $\mathcal{Y}^n$, which can be expressed as

$$q_{Y^n|X^n}(y^n|x^n) := \prod_{i=1}^{n} q_{Y_i|T^n}(y_i|t^n). \tag{5.11}$$

On the AgrLearn framework as specified by equation (5.11), define

$$\ell_{\mathrm{CE}}^{(n)} := -\mathbb{E}_{x^n y^n \sim p_{XY}^{\otimes n}} \log q_{Y^n|X^n}(y^n|x^n) \tag{5.12}$$

where $p_{XY}^{\otimes n}$ is the distribution on $(\mathcal{X} \times \mathcal{Y})^n$ induced by drawing n samples i.i.d. from $p_{XY}(x, y)$. Clearly $\ell_{\mathrm{CE}}^{(n)}$ is nothing more than the cross-entropy loss of the network's predictive distribution $q_{Y^n|X^n}(y^n|x^n)$ for the aggregated input X^n with respect to their labels Y^n. As we will be minimizing this cross-entropy loss function, we next discuss its properties.

Following Theorem 5.2.1,

$$\ell_{\mathrm{CE}}^{(n)} \geq nH(Y) - I(Y^n; T^n) \tag{5.13}$$

and if the post-bottleneck network component $q_{Y^n|T^n}(y^n|t^n)$ has sufficient capacity, then

$$\min_{q_{Y^n|T^n}(y^n|t^n)} \ell_{\mathrm{CE}}^{(n)} = nH(Y) - I(Y^n; T^n) \tag{5.14}$$

That is if the post-bottleneck component has sufficient capacity, then minimizing $\ell_{\mathrm{CE}}^{(n)}$ over the entire AgrLearn network also maximizes $I(Y^n; T^n)$.

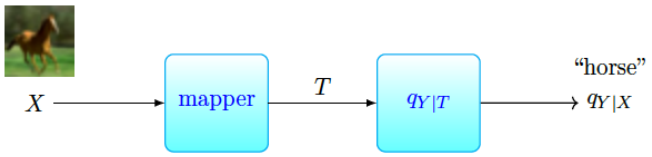


Figure 5.2: AgrLearn framework when there is no aggregation among input objects (fold-1).

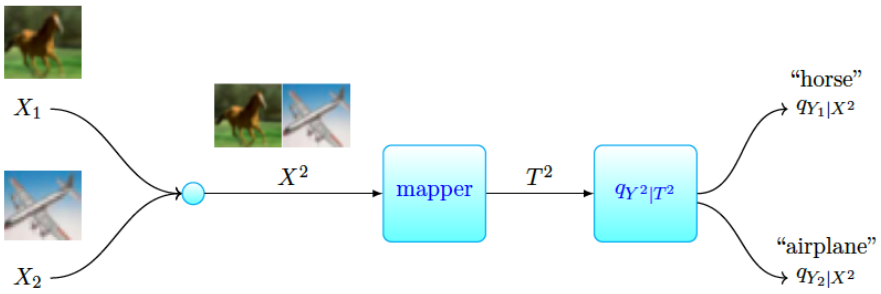


Figure 5.3: AgrLearn framework when there is aggregation among two input objects (fold-2).

5.3.1 Deterministic AgrLearn (dAgrLearn)

Here we explain one of the AgrLearn classes which we call deterministic AgrLearn (dAgrLearn). In dAgrLearn the concatenation of n objects X^n is considered as input. This n-fold aggregated input is mapped to an aggregated bottleneck T^n by a deterministic mapping $h : \mathcal{X}^n \to \mathcal{T}^n$ such that:

$$T^n := h(X^n). \tag{5.15}$$

Then post-bottleneck module applies a stochastic mapping from $\mathcal{T}^n$ to $\mathcal{Y}^n$ such that:

$$q_{Y^n|T^n}(y^n|t^n) := \prod_{i=1}^{n} q_{Y_i|T^n}(y_i|t^n). \tag{5.16}$$

Therefore, cross-entropy loss is equal to

$$\ell_{\mathrm{CE}}^{(n)} := -\mathbb{E}_{x^n y^n \sim p_{XY}^{\otimes n}} \log q_{Y^n|X^n}(y^n|x^n) \tag{5.17}$$

which is defined in equation (5.12). This gives the first term of equation (5.7).

For estimating the second term of equation (5.7), regularization term $I(X;T)$, we add a regularizing network, which is essentially a Mutual Information Neural Estimator (MINE) network [7].

In this approach, the MINE network serves to estimate $I(X;T)$ and penalizes it during the training of the dAgrLearn.

As explained in Chapter 3, mutual information $I(U;V)$ can be estimated as

$$\widehat{I}(U;V) := \max_{\gamma\in\Gamma} \left\{ \mathbb{E}_{(u,v)\sim p_{UV}(u,v)} \gamma(u,v) - \log \mathbb{E}_{(u,v)\sim p_U(u)\otimes p_V(v)} \exp\left(\gamma(u,v)\right) \right\} \tag{5.18}$$

by the dual representation of KL-divergence [17]. We will denote the term that gets maximized in equation (5.18) by $J(U,V;\gamma)$, as shown below,

$$J(U,V;\gamma) := \mathbb{E}_{(u,v)\sim p_{UV}(u,v)} \gamma(u,v) - \log \mathbb{E}_{(u,v)\sim p_U(u)\otimes p_V(v)} \exp\left(\gamma(u,v)\right) \tag{5.19}$$

and re-express $\widehat{I}(U;V)$ as

$$\widehat{I}(U;V) = \max_{\gamma\in\Gamma} J(U,V;\gamma) \tag{5.20}$$

As usual, practical computation of $J(U,V;\gamma)$ exploits Monte-Carlo approximation based on samples drawn from $p_{UV}(u,v)$. A natural way to apply MINE to the estimation of $I(X;T)$ in dAgrLearn is taking $\mathcal{U} := \mathcal{X}^n$, $\mathcal{V} := \mathcal{T}^n$, $U = X^n$, $V = T^n$. This allows us to estimate $I(X^n;T^n)$ by

$$\widehat{I}(X^n;T^n) = \max_{\gamma\in\Gamma} J(X^n,T^n;\gamma) \tag{5.21}$$

where T^n is computed by the pre-bottleneck component of the dAgrLearn with X^n as its input.

5.3.1.a Deterministic AgrLearn Training

We may define an overall objective function $\Omega(h, q_{Y^n|T^n}, \gamma)$ as

$$\Omega(h, q_{Y^n|T^n}, \gamma) := \ell_{\mathrm{CE}}^{(n)} + \alpha J(X^n,T^n;\gamma) \tag{5.22}$$

where we note that the term $\alpha J(X^n,T^n;\gamma)$ also depends on h implicitly. The above development then suggests that solving the IB learning problem in the form of equation (5.7) can be approximated by solving the following min-max problem:

$$\min_{h, q_{Y^n|T^n}} \max_{\gamma} \Omega(h, q_{Y^n|T^n}, \gamma) \tag{5.23}$$

In the training of dAgrLearn, mini-batched SGD can be used to solve the above min-max problem. The training algorithm is given in Algorithm 5.1.

Algorithm 5.1: Training in n-fold dAgrLearn

Initialize h, $q_{Y^n|T^n}$, and γ;

while *not stop training* **do**

Draw $m \times n$ examples to form a batch of m, n-fold aggregated examples $\left\{x^n_{(1)}, x^n_{(2)}, \ldots, x^n_{(m)}\right\}$;

for $z = 1$ to Z **do**

for $i = 1$ to m **do**

$t^n_{(i)} := h(x^n_{(i)})$

Select a random permutation τ on $\{1, 2, \ldots, m\}$;

Forward compute $J := \frac{1}{m}\sum_{i=1}^{m} \gamma(x^n_{(i)}, t^n_{(i)}) - \log \frac{1}{m}\sum_{i=1}^{m} \exp\left(\gamma(x^n_{(i)}, t^n_{(\tau(i))})\right)$;

$\gamma \leftarrow \gamma + \lambda_{\text{in}} \frac{\partial J}{\partial \gamma}$;

Select a random permutation τ on $\{1, 2, \ldots, m\}$;

Forward compute $J := \frac{1}{m}\sum_{i=1}^{m} \gamma(x^n_{(i)}, t^n_{(i)}) - \log \frac{1}{m}\sum_{i=1}^{m} \exp\left(\gamma(x^n_{(i)}, t^n_{(\tau(i))})\right)$;

Forward compute $\ell := \frac{1}{m}\sum_{i=1}^{m} \log q_{Y^n|T^n}(y^n_{(i)}|t^n_{(i)})$;

Compute $\Omega := \ell + \alpha \cdot J$;

$h \leftarrow h - \lambda_{\text{out}} \frac{\partial \Omega}{\partial h}$;

$q_{Y^n|T^n} \leftarrow q_{Y^n|T^n} - \lambda_{\text{out}} \frac{\partial \Omega}{\partial q_{Y^n|T^n}}$;

5.3.1.b Deterministic AgrLearn Prediction

For the prediction phase of dAgrLearn, we can use three protocols: Replicated Classification, Contextual Classification, and Batched Classification.

In the *Replicated Classification* protocol each object X is replicated n times and concatenated to form the input. The average of n predictive distributions generated by the model is taken as the label predictive distribution for X.

In *Contextual Classification*, for each object X, $n-1$ random examples are drawn from the training set and concatenated with X to form the input; the predictive distribution for X generated by the model is then retrieved. This process is repeated k times, and the average of the k predictive distribution is taken as the label predictive distribution for X.

In *Batched Classification*, the set of all objects in the test dataset is classified jointly by drawing k random batches of n objects from the test dataset. The objects in the i^{th} batch B_i are concatenated to form the input and passed to the model. The final label predictive distribution for each object X in the test dataset is taken as the average of the predictive distributions of X output by the model for all batches B_i's containing X. We compare these three protocols later in this chapter.

5.3.2 Probabilistic AgrLearn (pAgrLearn)

The computational complexity of the MINE network is quite high, particularly for aggregated inputs. Therefore, in this section we present an alternative AgrLearn scheme, aimed at reducing complexity without sacrificing performance. We call this class of AgrLearn, probabilistic AgrLearn (pAgrLearn).

To begin with we derive a variational upper bound of $I(X^n;T^n)$.

Lemma 2. *For any distribution r_{T^n} on $\mathcal{T}^n$ and distribution $p_{X^nT^n}$ on $\mathcal{X}^n \times \mathcal{T}^n$ for which $p_{T^n|X^n}$ does not correspond to a deterministic mapping,*

$$I(X^n;T^n) \leq \mathbb{E}_{x^n\sim p_{X^n}} \mathrm{KL}(p_{T^n|X^n}(\cdot|x^n)||r_{T^n}) \tag{5.24}$$

Proof. First note

$$\begin{aligned}
I(X^n;T^n) &:= \mathrm{KL}(p_{X^nT^n}(x^n,t^n)\|p_{X^n}(x^n)p_{T^n}(t^n)) \\
&= \mathbb{E}_{(x^n,t^n)\sim p_{X^nT^n}} \log \frac{p_{X^nT^n}(x^n,t^n)}{p_{X^n}(x^n)p_{T^n}(t^n)} \\
&= \mathbb{E}_{(x^n,t^n)\sim p_{X^nT^n}} \log \frac{p_{T^n|X^n}(t^n|x^n)p_{X^n}(x^n)}{p_{X^n}(x^n)p_{T^n}(t^n)} \\
&= \mathbb{E}_{(x^n,t^n)\sim p_{X^nT^n}} \log \frac{p_{T^n|X^n}(t^n|x^n)}{p_{T^n}(t^n)} \\
&= \mathbb{E}_{(x^n,t^n)\sim p_{X^nT^n}} \log p_{T^n|X^n}(t^n|x^n) - \mathbb{E}_{t^n\sim p_{T^n}} \log p_{T^n}(t^n)
\end{aligned} \tag{5.25}$$

Since computing the marginal distribution of T^n, namely $p_{T^n}(t^n)$, might be difficult, we use a variational approximation of this distribution which is denoted by $r_{T^n}(t^n)$ which is a fixed K-dimensional spherical Gaussian. Since KL-divergence is greater than 0, we have:

$$\begin{aligned}
&\mathrm{KL}(p_{T^n}(t^n)\|r_{T^n}(t^n)) \geq 0 \\
&\mathbb{E}_{t^n\sim p_{T^n}} \log \frac{p_{T^n}(t^n)}{r_{T^n}(t^n)} \geq 0 \\
&\mathbb{E}_{t^n\sim p_{T^n}} \log p_{T^n}(t^n) \geq \mathbb{E}_{t^n\sim p_{T^n}} \log r_{T^n}(t^n)
\end{aligned}$$

replacing this bound in (5.25), gives rise to:

$$\begin{aligned}
I(X^n;T^n) &\leq \mathbb{E}_{(x^n,t^n)\sim p_{X^nT^n}} \log \frac{p_{T^n|X^n}(t^n|x^n)}{r_{T^n}(t^n)} \\
&= \mathbb{E}_{\substack{(x^n,y^n)\sim p_{X^nY^n}, \\ t^n\sim p_{T^n|X^n}(\cdot|x^n)}} \log \frac{p_{T^n|X^n}(t^n|x^n)}{r_{T^n}(t^n)}
\end{aligned} \tag{5.26}$$

■

This lemma allows us to replace $I(X^n;T^n)$ in (5.7) with its upper bound in (5.24). However, in order for the bound to be valid, it is required that the mapping from X^n to T^n not be a deterministic one. For this reason, we modify the pre-bottleneck module to be a stochastic mapping following some conditional distribution $p_{T^n|X^n}$. In particular, we take $p_{T^n|X^n}(\cdot|x^n)$ to be a Gaussian distribution with mean vector $\mu(x^n)$ and standard deviation $\sigma(x^n)$. Specifically, if the dimension of T^n is K, both μ and σ are implemented as networks with K outputs. To ensure that $\sigma(x^n)$ has positive values, the output of σ is passed through a soft-plus function. The soft-plus transformation

$$\mathrm{Softplus}(x) = \frac{1}{\beta}\log(1+e^{\beta x}) \tag{5.27}$$

constrains the output to be positive. β by default is considered as 1.

Despite the stochastic nature of $p_{T^n|X^n}$, such a scheme can be implemented using the Reparameterization Trick [46], in which low-variance back-propagation is elegantly supported. Hence, the output of the pre-bottleneck module is:

$$T^n = \mu(X^n) + \epsilon.\sigma(X^n) \tag{5.28}$$

Algorithm 5.2: Training in n-fold pAgrLearn

Initialize $p_{T^n|X^n}, q_{Y^n|T^n}$;

while *not stop training* **do**

Draw $m \times n$ examples to form a batch of m n-fold aggregated examples $\left\{x^n_{(1)}, x^n_{(2)}, \ldots, x^n_{(m)}\right\}$;

Forward compute $I := \frac{1}{m}\sum_{i=1}^{m} \mathrm{KL}(\mathcal{N}(t^n_{(i)}|\mu(x^n_{(i)}), \sigma(x^n_{(i)}))\|\mathcal{N}(t^n_{(i)}|0, I))$;

$t^n_{(i)} = \mu(x^n_{(i)}) + \epsilon.\sigma(x^n_{(i)})$;

Forward compute $\ell := \frac{1}{m}\sum_{i=1}^{m} \log q_{Y^n|T^n}(y^n_{(i)}|t^n_{(i)})$;

Compute $\Psi := \ell + \alpha \cdot I$;

$p_{T^n|X^n} \leftarrow p_{T^n|X^n} - \lambda_{\text{out}} \frac{\partial \Psi}{\partial p_{T^n|X^n}}$;

$q_{Y^n|T^n} \leftarrow q_{Y^n|T^n} - \lambda_{\text{out}} \frac{\partial \Psi}{\partial q_{Y^n|T^n}}$;

where ϵ is a Gaussian random variable. This completely defines the pre-bottleneck module. The post-bottleneck module takes a structure identical to that in dAgrLearn.

In Chapter 3 we introduced the VIB framework proposed by Alemi et al. [4]. We should point out that if we put $n = 1$ in pAgrLearn, it gives rise to the VIB framework. We also experimentally establish the difference between these two later in this chapter.

The main difference between the pAgrLearn and the dAgrLearn is that the mapping from $\mathcal{X}^n$ to $\mathcal{T}^n$ in pAgrLearn is no longer deterministic. In addition, the regularization term in dAgrLearn is estimated by the MINE network which is different from the KL-divergence approach used in the pAgrLearn.

5.3.2.a Probabilistic AgrLearn Training

In pAgrLearn we may define an overall objective function $\Psi(p_{T^n|X^n}, q_{Y^n|T^n})$ as

$$\Psi(p_{T^n|X^n}, q_{Y^n|T^n}) := \ell^{(n)}_{\text{CE}} + \alpha I(X^n, T^n) \tag{5.29}$$

Therefore solving the IB learning problem in the form of equation (5.7) can be approximated by solving the following minimization problem:

$$\min_{p_{T^n|X^n}, q_{Y^n|T^n}} \Psi(p_{T^n|X^n}, q_{Y^n|T^n}) \tag{5.30}$$

Mini-batched SGD can be used to solve the minimization problem (5.30) in the training of pAgrLearn. The training algorithm is given in Algorithm 5.2.

5.3.2.b Probabilistic AgrLearn Prediction

In the prediction phase of pAgrLearn, we can use three protocols: Replicated Classification, Contextual Classification, and Batched Classification, the same as dAgrLearn.

We use the *Replicated Classification* protocol in pAgrLearn such that each object X is replicated n times and concatenated as an input. Then the average of n predictive distributions generated by the model is taken as the label predictive distribution for X.

5.4 Summary

In this chapter, based on the equivalence between IB learning and IB quantization and using the variational approach, we introduced a novel framework for learning neural networks, namely the AgrLearn framework. We discussed two different classes of the framework, deterministic AgrLearn and probabilistic AgrLearn including their training and prediction phases.

Chapter 6

Application of Aggregated Learning Framework to Classification Tasks

6.1 Introduction

In this chapter, we explain the application of Aggregated Learning (AgrLearn) framework to classification tasks, image recognition and natural language processing (NLP) tasks, specifically sentiment analysis. In Section 6.2 we evaluate AgrLearn framework with deep network architectures for classification tasks over images. We use standard benchmark datasets for this purpose. In addition, we show the results of applying different regularization approaches on AgrLearn framework. In Section 6.3 we first present a brief introduction to sentiment analysis, then we investigate how AgrLearn framework performs when applied to sentiment analysis. Finally, a summary of this chapter is presented in Section 6.4.

6.2 Image Recognition with Aggregated Learning

In this section, we apply the AgrLearn framework to solve image recognition tasks, in which an image is labeled from a fixed set of labels. For this purpose, we set some hyperparameters for AgrLearn framework without any aggregation among input objects (fold-1) and without

any modifications we use them for AgrLearn framework. We use mini-batched backprop for 400 epochs which was selected after investigating [200, 250, 300, 350, 400, 500] epochs, with exactly the same hyper-parameter settings, without dropout. Here an epoch refers to going over N aggregated training examples. Specifically, weight decay is 10^{-4}, and each mini-batch contains 64 aggregated training examples. The learning rate is set to 0.1 initially and decays by a factor of 10 after 100, 150, and 250 epochs. For selecting these values we followed the policy in [85]. The learning rate for the MINE network is set to 0.001 initially after examining [0.0001, 0.001, 0.01,0.05, 0.1] values and decays by a factor 0.95 for each epoch. Each reported performance value is the median of the performance values obtained in the final 10 epochs by averaging that value over running the same setting with different random seed initialization, 7 times. We should mention that selecting the median of the performance values is a more conservative and reliable estimation.

Experiments are conducted on the Canadian Institute For Advanced Research datasets, namely **CIFAR-10**, **CIFAR-100** and Google Street View House Numbers recognition (SVHN) dataset with three widely used deep network architectures, namely ResNet [30, 31], WideResNet [82] and VGG [71] which were introduced in Chapter 2.

The **CIFAR-10** dataset has 60,000, 32×32 color images in 10 classes with 6000 images per class. The classes are airplane, car, cat, bird, deer, dog, frog, ship, horse, and truck. In this dataset there are 50,000 training images, and 10,000 test images [1]. The **CIFAR-100** dataset is similar to CIFAR-10 but with 100 classes [2]. In this dataset, each image not only belongs to a class which is called a fine label but also to a superclass which is called a coarse label. The contents of this dataset are shown in Table 6.1.

SVHN is the Google Street View House Numbers recognition dataset with 73,257 digits (0-9) for training, 26,032 digits for testing, and 531,131 additional, easier samples, as extra training data [3]. We did not use the additional images. This dataset contains 10 classes. It has one label for each digit, for example, digit 1 has label 1.

We apply the AgrLearn framework to the 18-layer and 34-layer pre-activation and

[1] https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

[2] https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz

[3] http://ufldl.stanford.edu/housenumbers/

Table 6.1: Labels of the CIFAR-100 dataset

Superclass (coarse label)	Class (fine label)
Aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

50-layer pre-activation-bottleneck ResNet (*ResNet-18*, *ResNet-34* and *ResNet-50*) [31] , and the 22-layer WideResNet with widening factor 10 (*WideResNet-22-10*) [82]. The difference between the original ResNet and the pre-activation ResNet is that in pre-activation ResNet the activation function ReLU and Batch Normalization (BN) take place before the weight layers, and this causes improvement in the performance of ResNet. Figure 6.1 depicts the diagram of the pre-activation ResNet.

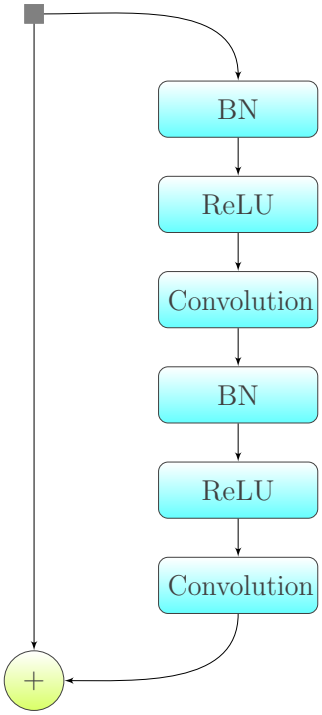


Figure 6.1: The diagram of pre-activation ResNet.

The bottleneck ResNet is introduced to reduce computational complexities since a lot of GPU rams are required for performing 3×3 convolutions in basic ResNet. With this architecture, a 1×1 convolution layer is used to reduce the channels of input before the 3×3 convolution layer and after that, there is a 1×1 convolution layer to return it back to its original shape. For example, if the input has depth 256, the first 1×1 filter reduces the depth to 64 and then the 3×3 convolution layer operates on the vector with lower depth and the final 1×1 layer returns it to the original depth which was 256 [30].

The resulting network structure in AgrLearn differs from the original ResNet, WideResNet, and VGG in its n, parallel, soft-max classifier layers in the post-bottleneck part (as opposed to the single soft-max classifier layer in ResNet, WideResNet, and VGG). The soft-max classifier layer contains a fully connected layer and a soft-max activation function. Moreover, the number of filters in the last layer of the pre-bottleneck module is expanded by the factor n. This is required because the input dimension in AgrLearn framework increases significantly, and the model is required to extract joint features across individual objects in the amalgamated example. The number of parameters in each of these networks

Table 6.2: Number of parameters in per-bottleneck and post-bottleneck networks

Neural Network	**Pre-bottleneck**		**Post-bottleneck**		**Total # of parameters**	
	fold-1	fold-2	fold-1	fold-2	fold-1	fold-2
VGG-16	14,714,688	17,074,496	12,850,698	51,391,498	27,565,386	68,465,994
ResNet-18	10,954,176	20,392,384	12,850,698	51,391,498	23,804,874	71,783,882
ResNet-34	15,747,904	25,186,112	12,850,698	51,391,498	28,598,602	76,577,610
WideResNet-22-10	26,526,208	41,273,088	26,221,450	104,871,690	52,747,658	146,144,778

is shown in Table 6.2.

Note fold number shows the number of aggregated input objects. Therefore, the fold number 1 (fold-1) denotes the standard neural network in case of dAgrLearn or VIB in case of pAgrLearn in which just one object passes to the network, and a fold number greater than 1 denotes the AgrLearn framework either dAgrLearn or pAgrLearn wherein multiple objects are aggregated and passed to the network. The quantity α is the coefficient of the second term in equation (5.7), in which $\alpha = 0$ corresponds to the case when only the cross-entropy loss is considered, and $\alpha > 0$ corresponds to the case when the regularization term is added to the cross-entropy loss. The value of α is heuristically tuned for finding the best performance. In case of dAgrLearn, α is selected from the set $[0.1, 0.3, 0.5, 0.7, 0.9, 1, 4]$, and in case of pAgrLearn $[1, 10^{-2}, 10^{-4}, 10^{-6}]$ are investigated for finding α.

6.2.1 Replicated Classification vs. Contextual Classification

First, we compare the protocols that we introduced for the prediction of dAgrLearn. As we mentioned earlier, in the prediction phase of dAgrLearn, using the Contextual Classification protocol, we concatenate each object X with $n-1$ random examples that are drawn from the training set to form the input. The predictive distribution for X generated by the model is then retrieved. Repeating this process k times and taking the average of the k predictive distribution gives the label predictive distribution for X. We compare the performance of this protocol with the Replicated Classification protocol in which we replicate each object X, n times, and concatenate them to form the input. Then we take the average over n predictive distributions generated by the model as the label predictive distribution for X.

Table 6.3: Test error rates (%) of ResNet-18 and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100 for Contextual Classification protocol and Replicated Classification protocol

Dataset	**Contextual Classification**		**Replicated Classification**	
	fold-1	fold-2	fold-1	fold-2
CIFAR-10	5.08	4.90	5.08	4.89
CIFAR-100	23.70	23.11	23.70	23.03

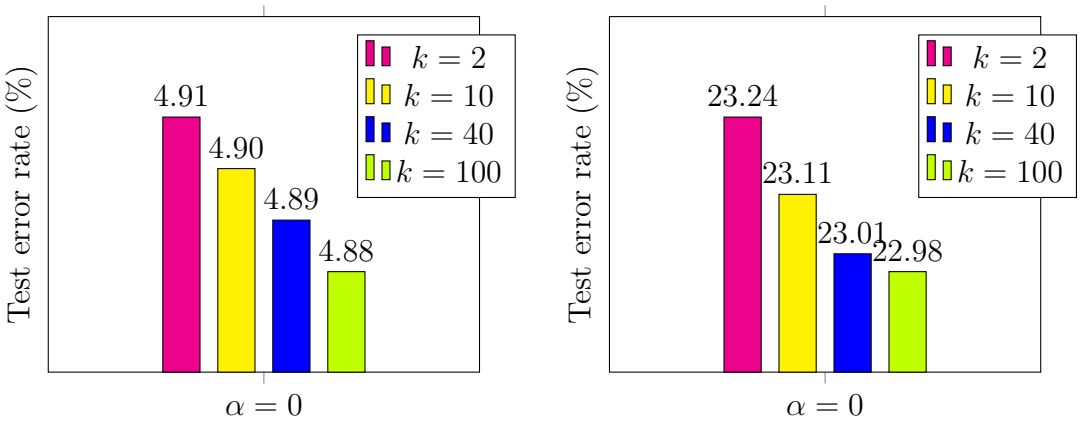
In the implementations for these comparisons, we do not consider any regularization term for the dAgrLearn, which means α in equation (5.7) is equal to 0.

Table 6.3 depicts test error rates for ResNet-18 using the Contextual Classification protocol and Replicated Classification protocol. The relative error reductions achieved for fold-2 using the Contextual Classification protocol with $k = 10$ are 3.54%, and 2.49% on CIFAR-10, and CIFAR-100, respectively. These reductions with respect to the Replicated Classification protocol are 3.74%, and 2.83% on CIFAR-10, and CIFAR-100, respectively. This illustrates that the performance of the Replicated classification protocols is better than the performance of the Contextual classification protocol in the case of $k = 10$.

We also compared the test error rates (%) of ResNet-18 using the Contextual Classification protocol for CIFAR-10 and CIFAR-100 for the different number of k in Figure 6.2. For example, the relative error reductions achieved for $k = 100$ in comparison with $k = 2$ are 0.61%, and 1.12% on CIFAR-10, and CIFAR-100, while the relative error reductions achieved for $k = 40$ in comparison with $k = 2$ are 0.41%, and 0.99% on CIFAR-10, and CIFAR-100, respectively. It can be seen that by increasing k, test error rate decreases in both CIFAR-10 and CIFAR-100.

6.2.2 Replicated Classification vs. Batched Classification

Another protocol that we explored for the dAgrLearn is Batched Classification. In this protocol, all objects in the test dataset are classified jointly by drawing k random batches of n objects from the test dataset. The objects in the i^{th} batch B_i are concatenated to



(a) CIFAR-10 (b) CIFAR-100

Figure 6.2: Test error rates (%) of ResNet-18 for fold-2 with Contextual Classification for different number of k.

form the input and passed to the model. The final label predictive distribution for each object X in the test dataset can be taken by averaging the predictive distributions of X output by the model for all batches B_i's containing X or by finding the maximum predictive distributions of X output by the model for all batches B_i's containing X. The average of the predictive distributions of X output is one approach in this protocol which we call *Batched-average*. Considering the maximum predictive distributions of X output by the model for all batches B_i's containing X is another approach which we call *Batched-max*.

Table 6.4 illustrates test error rates for ResNet-18 by using the Batched-average Classification protocol and Replicated Classification protocol. The achieved test error rates with respect to Batched-average Classification protocol with $k = 2$ for fold-2 are 5.03%, and 23.02% on CIFAR-10, and CIFAR-100, respectively. Test error rate reductions with respect to Batched-average Classification protocol ($k = 2$) in fold-2 for CIFAR-10 and CIFAR-100 are 0.98%, and 2.87% respectively, and these reductions with respect to Replicated Classification protocol are 3.74%, and 2.83%, respectively. We can see in some cases Replicated classification protocol is better than Batched classification protocol and in some cases, the Batched classification protocol is better than Replicated classification protocol. One cannot conclude either one is better.

Test error rates for ResNet-18 by using the Batched-max Classification protocol and

Table 6.4: Test error rates (%) of ResNet-18 and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100 for Batched-average Classification protocol and Replicated Classification protocol

Dataset	**Batched-average Classification**		**Replicated Classification**	
	fold-1	fold-2	fold-1	fold-2
CIFAR-10	5.08	5.03	5.08	4.89
CIFAR-100	23.70	23.02	23.70	23.03

Table 6.5: Test error rates (%) of ResNet-18 and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100 Batched-max Classification protocol and Replicated Classification protocol

Dataset	**Batched-max Classification**		**Replicated Classification**	
	fold-1	fold-2	fold-1	fold-2
CIFAR-10	5.08	5.06	5.08	4.89
CIFAR-100	23.70	22.81	23.70	23.03

Replicated Classification protocol is shown in Table 6.5. The achieved test error rate reductions with respect to Batched-max Classification protocol ($k = 2$) for fold-2 are 0.39%, and 3.76% on CIFAR-10, and CIFAR-100, respectively.

As a result of lower computational complexity of the Replicated Classification protocol, we select the Replicated Classification protocol for the prediction of the dAgrLearn and pAgrLearn. Therefore the results reported in the rest of this thesis have been obtained by using the Replicated Classification protocol unless otherwise specified.

6.2.3 Predictive Performance

In this section, we investigate the performance of AgrLearn framework for benchmark datasets on widely used deep learning architectures.

6.2.3.a Deterministic AgrLearn

The rates of prediction errors of dAgrLearn for different number of folds are shown in Tables 6.6 - 6.10.

It is seen from Table 6.6 that, in the case of ResNet-18, the relative error reductions (RER) achieved for fold-2 in comparison with fold-1, when α is equal to 0 are 3.74%, and 2.83% on CIFAR-10, and CIFAR-100 respectively, and the reductions are 3.86%, and 3.21% respectively when $\alpha > 0$.

Similarly, the relative error reductions for fold-2 for ResNet-34 (Table 6.7), ResNet-50 (Table 6.8), WideResNet-22-10 (Table 6.9), and VGG-16 (Table 6.10) are observed. The relative error reductions in the case of ResNet-34 for fold-2, when α is equal to 0 are 5.26%, and 5.16% on CIFAR-10, and CIFAR-100, and when $\alpha > 0$ the reductions are 5.30%, and 6.59% respectively. These reductions in the case of VGG-16 for fold-2, when α is equal to 0 are 4.98%, and 5.25% on CIFAR-10, and CIFAR-100, and when $\alpha > 0$, the reductions are 2.56%, and 3.94% respectively.

Table 6.6: Test error rates (%) of ResNet-18, its dAgrLearn counterparts and relative error reductions (RER) for fold-2 on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	5.08	4.92	4.89	4.73	3.74	3.86
CIFAR-100	23.70	23.70	23.03	22.94	2.83	3.21

Table 6.7: Test error rates (%) of ResNet-34, its dAgrLearn counterparts and relative error reductions (RER) for fold-2 on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.94	4.91	4.68	4.65	5.26	5.30
CIFAR-100	23.86	23.82	22.63	22.25	5.16	6.59

Table 6.8: Test error rates (%) of ResNet-50, its dAgrLearn counterparts and relative error reductions (RER) for fold-2 on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.80	4.76	4.60	4.60	4.17	3.36
CIFAR-100	23.60	22.56	21.39	21.14	9.36	6.29

Table 6.9: Test error rates (%) of WideResNet-22-10, its dAgrLearn counterparts and relative error reductions (RER) for fold-2 on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.30	4.23	4.19	4.18	2.56	1.18
CIFAR-100	21.13	21.10	20.30	20.28	3.93	3.89

In some cases, we observe that the differences in test error rates between the tests when $\alpha = 0$ and $\alpha > 0$ are very small and in some cases the differences are zero. The reason can be that SGD training under cross-entropy loss has an inherent compression effect [69], this makes the regularization term $I(X;T)$ less effective. Therefore in some cases keeping $\alpha = 0$ can be sufficient.

We have also presented results that show that deeper and wider residual networks, such as ResNet-50 and WideResNet-22-10, performed better in terms of error reductions. We examine the effect of depth and width of the network in Section 6.2.5.

6.2.3.b Probabilistic AgrLearn

The prediction error rates of pAgrLearn are shown in Figures 6.3 - 6.7. We also show the t-test results regarding the reported values in these figures.

As we explained, when fold number is equal to 1, the pAgrLearn reverts to VIB [4], therefore these figures depict the performance of pAgrLearn in comparison with VIB

Table 6.10: Test error rates (%) of VGG-16, its dAgrLearn counterparts and relative error reductions (RER) for fold-2 on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	6.42	6.26	6.10	6.10	4.98	2.56
CIFAR-100	26.84	26.42	25.43	25.38	5.25	3.94

framework. In other words, fold-1 illustrates VIB framework and fold-2 illustrates pAgrLearn which is a class of the AgrLearn framework.

The relative error reductions achieved with respect to ResNet-18 for fold-2, when α is equal to 0 are 6.37%, and 3.81% on CIFAR-10, and CIFAR-100, and when $\alpha > 0$ the reductions are 4.55%, and 3.69% on CIFAR-10, and CIFAR-100, respectively.

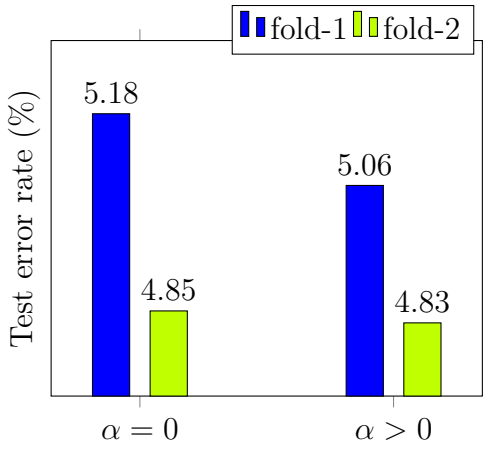
Similarly significant improvement upon ResNet-34, ResNet-50, WideResNet-22-10 and VGG-16 is observed. For example, the relative error reductions in the case of ResNet-34, for fold-2, when α is equal to 0 are 2.20%, and 5.84% on CIFAR-10, and CIFAR-100, and when $\alpha > 0$, the reductions are 3.82%, and 6.80% respectively. As we can see pAgrLearn has significant improvement on performance in comparison with VIB.

In Figures 6.3 - 6.7 we also report the t-value and the p-value. The t-value is defined as a measure of difference in the set of results:

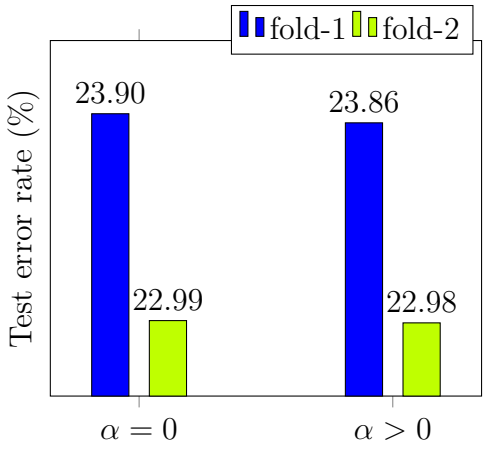
$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1}{m_1} + \frac{\sigma_2}{m_2}}} \tag{6.1}$$

where μ_i, σ_i and m_i denote the mean, variance and number of samples when $i = \{1, 2\}$. The larger the t-value, the greater difference is between two sets of results. The p-value denotes the statistical significance of the results. The significant level is defined for the p-value. In this experiment this value is 0.05. Therefore, the results have significant difference when $p < 0.05$.

In Figure 6.3 the reported p-value with respect to ResNet-18, for fold-1 and fold-2 when $\alpha = 0$ is equal to 2.51×10^{-5} , and when $\alpha > 0$ is equal to 2.71×10^{-4} for CIFAR-100,



(a) CIFAR-10



(b) CIFAR-100

(c) (t, p)-values for CIFAR-10

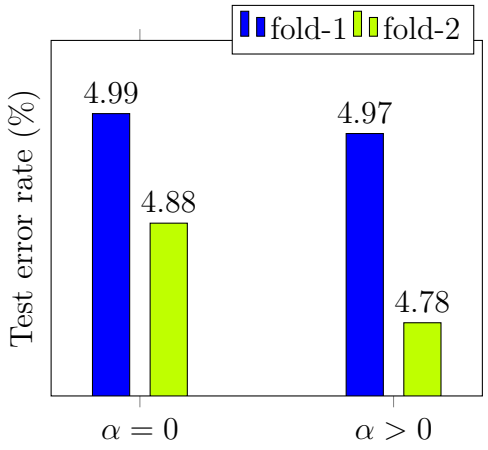
fold-2, $\alpha = 0$	$(4.96, 4.39 \times 10^{-4})$	$(3.56, 2.57 \times 10^{-3})$
fold-2, $\alpha > 0$	$(5.97, 2.78 \times 10^{-4})$	$(6.81, 3.89 \times 10^{-6})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(d) (t, p)-values for CIFAR-100

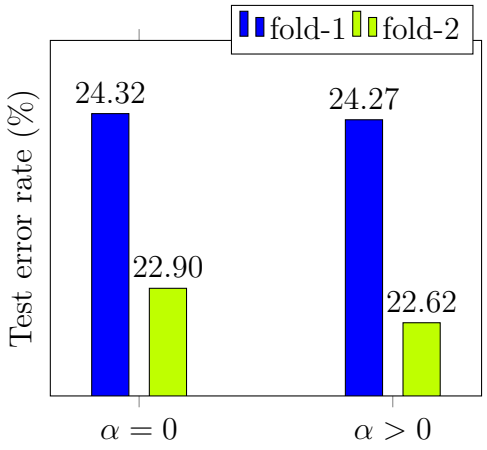
fold-2, $\alpha = 0$	$(6.75, 2.51 \times 10^{-5})$	$(5.58, 1.71 \times 10^{-4})$
fold-2, $\alpha > 0$	$(7.51, 6.79 \times 10^{-5})$	$(6.00, 2.71 \times 10^{-4})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

Figure 6.3: Test error rates (%) of VIB (fold-1) and pAgrLearn (fold-2) for ResNet-18 on (a) CIFAR-10 and (b) CIFAR-100 and their corresponding (c) (t, p)-values on CIFAR-10 and (d) (t, p)-values on CIFAR-100.

which is much smaller than 0.05. The same behavior is observed with respect to other networks and datasets. Overall, the (t, p)-values in these figures give us a high degree of confidence that the results of fold-2 and fold-1 are significantly different.



(a) CIFAR-10



(b) CIFAR-100

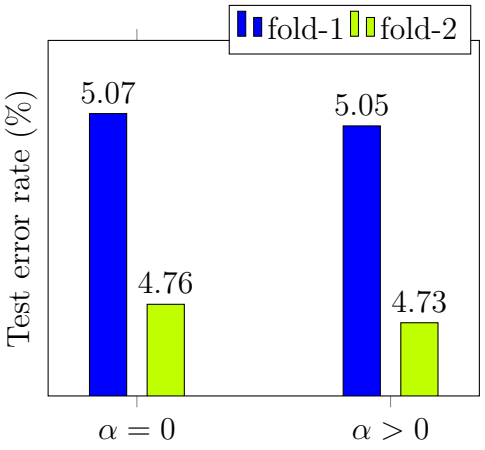
(c) (t, p)-values for CIFAR-10

fold-2, $\alpha = 0$	$(1.89, 4.42 \times 10^{-2})$	$(1.77, 4.27 \times 10^{-2})$
fold-2, $\alpha > 0$	$(3.32, 3.87 \times 10^{-3})$	$(2.70, 1.21 \times 10^{-2})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

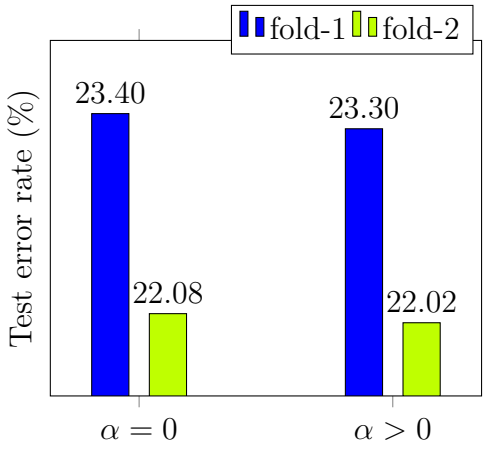
(d) (t, p)-values for CIFAR-100

fold-2, $\alpha = 0$	$(9.87, 4.69 \times 10^{-6})$	$(7.19, 5.52 \times 10^{-6})$
fold-2, $\alpha > 0$	$(21.52, 2.96 \times 10^{-11})$	$(11.17, 1.85 \times 10^{-6})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

Figure 6.4: Test error rates (%) of VIB (fold-1) and pAgrLearn (fold-2) for ResNet-34 on (a) CIFAR-10 and (b) CIFAR-100 and their corresponding (c) (t, p)-values on CIFAR-10 and (d) (t, p)-values on CIFAR-100.



(a) CIFAR-10



(b) CIFAR-100

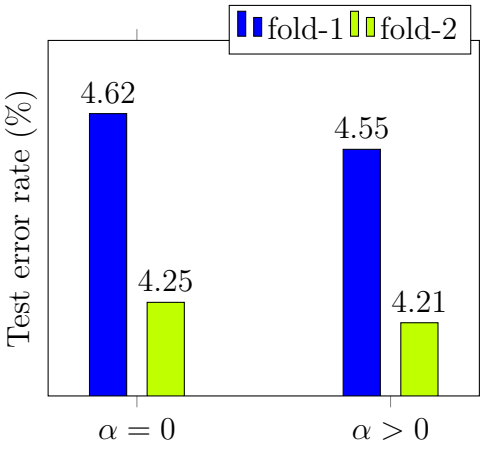
(c) (t, p)-values for CIFAR-10

fold-2, $\alpha = 0$	$(5.17, 1.16 \times 10^{-4})$	$(4.66, 2.76 \times 10^{-4})$
fold-2, $\alpha > 0$	$(4.23, 1.11 \times 10^{-3})$	$(3.94, 1.38 \times 10^{-3})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

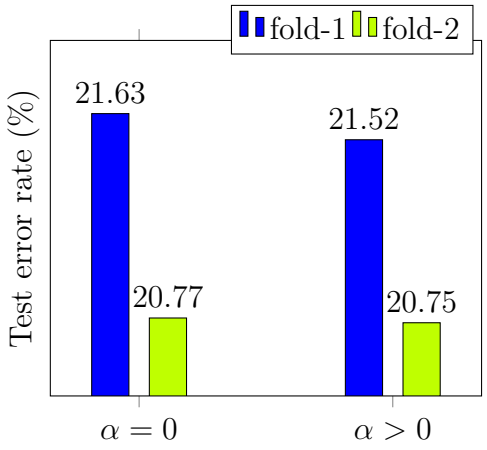
(d) (t, p)-values for CIFAR-100

fold-2, $\alpha = 0$	$(8.76, 5.31 \times 10^{-6})$	$(5.68, 3.76 \times 10^{-4})$
fold-2, $\alpha > 0$	$(7.19, 5.50 \times 10^{-6})$	$(5.24, 1.39 \times 10^{-4})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

Figure 6.5: Test error rates (%) of VIB (fold-1) and pAgrLearn (fold-2) for ResNet-50 on (a) CIFAR-10 and (b) CIFAR-100 and their corresponding (c) (t, p)-values on CIFAR-10 and (d) (t, p)-values on CIFAR-100.



(a) CIFAR-10



(b) CIFAR-100

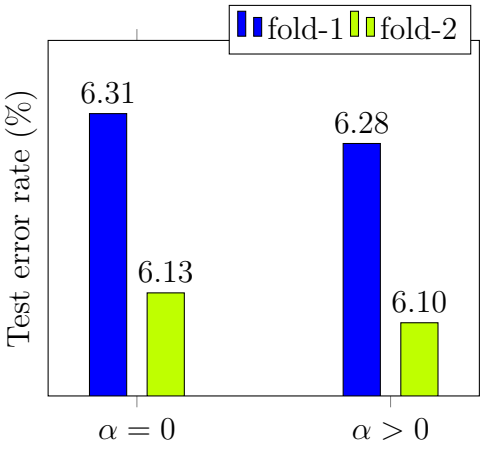
(c) (t, p)-values for CIFAR-10

fold-2, $\alpha = 0$	$(7.90, 2.38 \times 10^{-5})$	$(9.56, 5.79 \times 10^{-7})$
fold-2, $\alpha > 0$	$(8.22, 1.79 \times 10^{-5})$	$(10.07, 7.49 \times 10^{-7})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

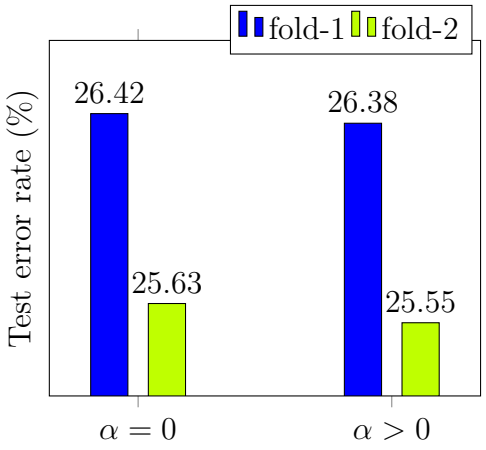
(d) (t, p)-values for CIFAR-100

fold-2, $\alpha = 0$	$(10.13, 7.06 \times 10^{-7})$	$(6.77, 7.13 \times 10^{-5})$
fold-2, $\alpha > 0$	$(10.02, 3.64 \times 10^{-7})$	$(6.81, 3.91 \times 10^{-5})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

Figure 6.6: Test error rates (%) of VIB (fold-1) and pAgrLearn (fold-2) for WideResNet-22-10 on (a) CIFAR-10 and (b) CIFAR-100 and their corresponding (c) (t, p)-values on CIFAR-10 and (d) (t, p)-values on CIFAR-100.



(a) CIFAR-10



(b) CIFAR-100

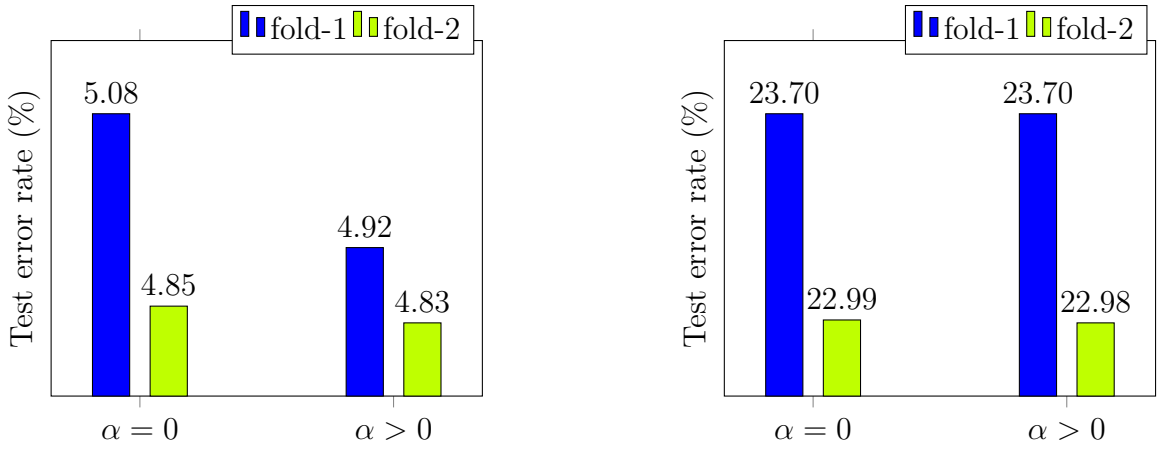
(c) (t, p)-values for CIFAR-10

fold-2, $\alpha = 0$	$(2.93, 6.89 \times 10^{-3})$	$(1.95, 3.77 \times 10^{-2})$
fold-2, $\alpha > 0$	$(3.79, 1.28 \times 10^{-3})$	$(2.33, 2.11 \times 10^{-2})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(d) (t, p)-values for CIFAR-100

fold-2, $\alpha = 0$	$(5.83, 4.06 \times 10^{-5})$	$(5.32, 1.23 \times 10^{-4})$
fold-2, $\alpha > 0$	$(5.54, 6.38 \times 10^{-5})$	$(5.13, 1.24 \times 10^{-4})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

Figure 6.7: Test error rates (%) of VIB (fold-1) and pAgrLearn (fold-2) for VGG-16 on (a) CIFAR-10 and (b) CIFAR-100 and their corresponding (c)(t, p)-values on CIFAR-10 and (d) (t, p)-values on CIFAR-100.



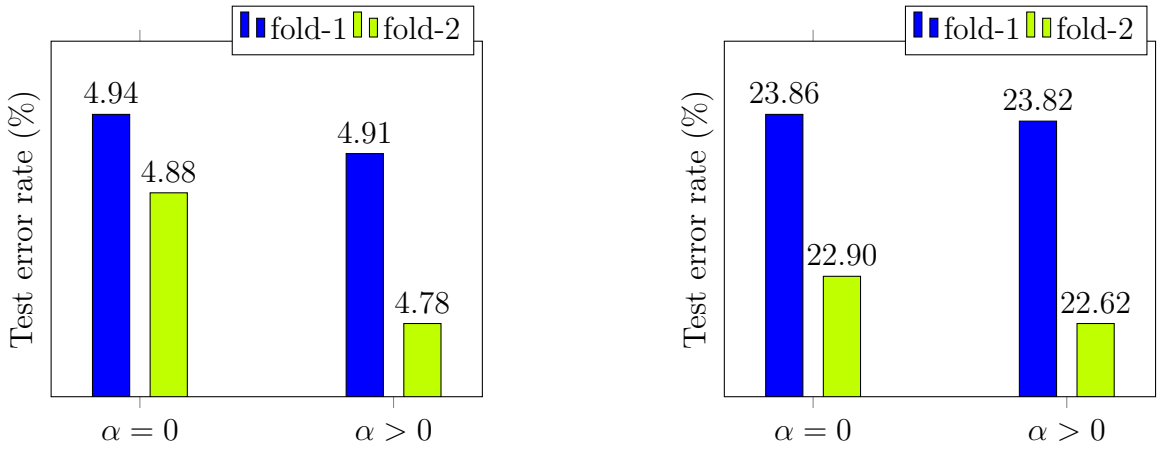
(a) CIFAR-10

(b) CIFAR-100

Figure 6.8: Test error rates (%) of Standard Neural Network (fold-1), and probabilistic AgrLearn (fold-2) for ResNet-18 on (a) CIFAR-10 and (b) CIFAR-100.

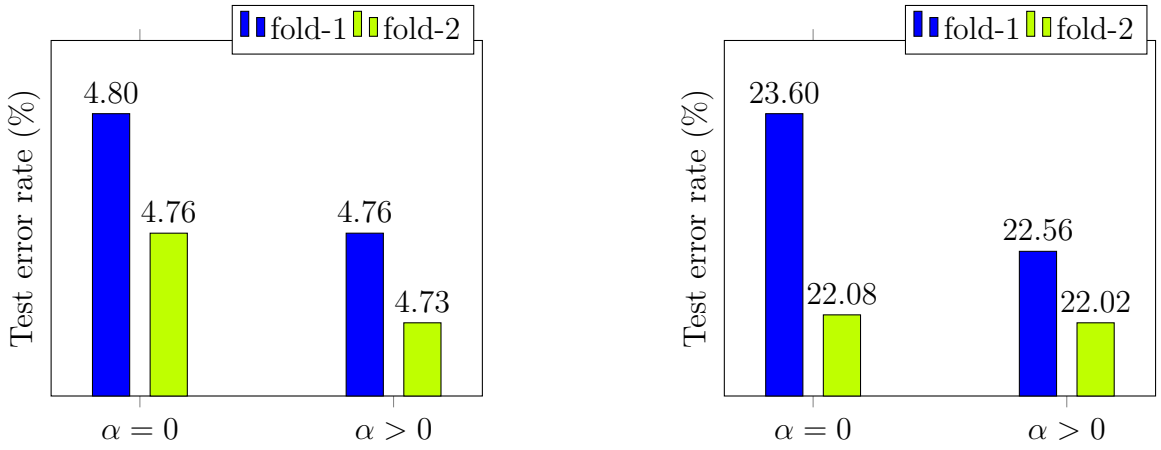
6.2.3.c Probabilistic AgrLearn vs Standard Neural Network

We compare probabilistic AgrLearn with the standard neural network in Figures 6.8 - 6.12. The standard neural network is fold-1 in the dAgrLearn where there is no aggregation among input objects, which means just one object input passes to the network. In Figure 6.8 the relative error reductions with respect to ResNet-18 for fold-2, when $\alpha = 0$ are 4.53%, and 3.00% on CIFAR-10, and CIFAR-100 respectively. Similarly, the relative error reductions for ResNet-34 (Figure 6.9), ResNet-50 (Figure 6.10), WideResNet-22-10 (Figure 6.11), and VGG-16 (Figure 6.12) are observed. The results show that the performance of pAgrLearn is superior to the performance of the standard neural networks.



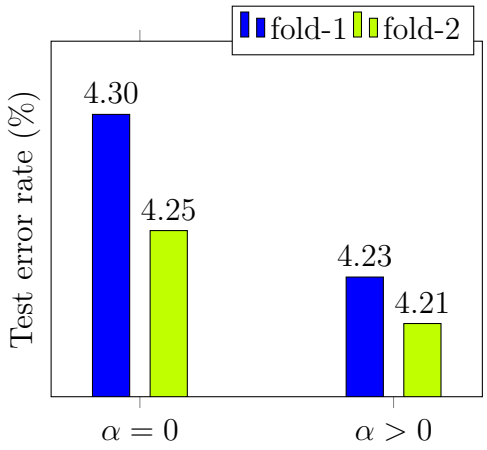
(a) CIFAR-10 (b) CIFAR-100

Figure 6.9: Test error rates (%) of Standard Neural Network (fold-1), and probabilistic AgrLearn (fold-2) for ResNet-34 on (a) CIFAR-10 and (b) CIFAR-100.

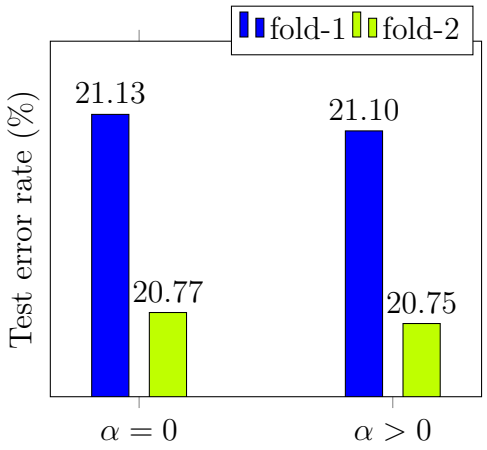


(a) CIFAR-10 (b) CIFAR-100

Figure 6.10: Test error rates (%) of Standard Neural Network (fold-1), and probabilistic AgrLearn (fold-2) for ResNet-50 on (a) CIFAR-10 and (b) CIFAR-100.

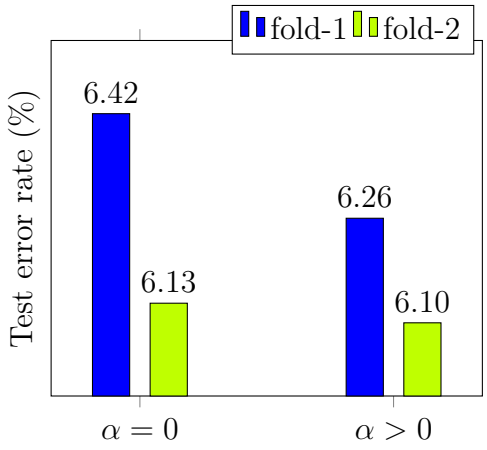


(a) CIFAR-10

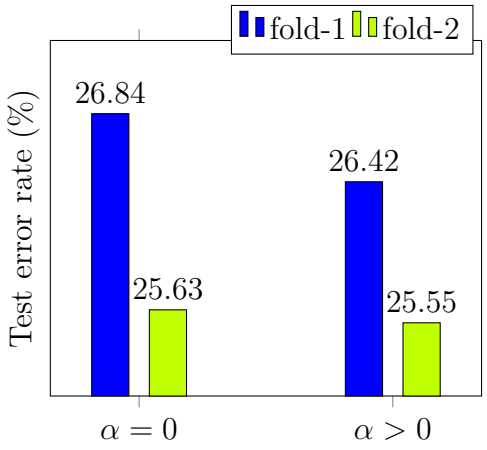


(b) CIFAR-100

Figure 6.11: Test error rates (%) of Standard Neural Network (fold-1), and probabilistic AgrLearn (fold-2) for WideResNet-22-10 on (a) CIFAR-10 and (b) CIFAR-100.



(a) CIFAR-10



(b) CIFAR-100

Figure 6.12: Test error rates (%) of Standard Neural Network (fold-1), and probabilistic AgrLearn (fold-2) for VGG-16 on (a) CIFAR-10 and (b) CIFAR-100.

6.2.3.d Probabilistic AgrLearn vs Deterministic AgrLearn

We also compare test error rates in pAgrLearn with dAgrLearn in Tables 6.11 - 6.15. The test error rates in pAgrLearn with respect to ResNet-18 when $\alpha = 0$ are 4.85% and 22.99% for CIFAR-10, and CIFAR-100 respectively. The test error rates in the dAgrLearn with respect to ResNet-18 when $\alpha = 0$ are 4.89% and 23.03% for CIFAR-10 and CIFAR-100. These results show the test error rates in some cases decrease more in pAgrLearn.

In Table 6.16, we also compare time complexity between pAgrLearn and dAgrLearn on CIFAR-10. Time duration for running 1 epoch with respect to ResNet-18 on CIFAR-10 is 148 seconds for dAgrLearn and 68 seconds for pAgrLearn. The relative percentage with respect to ResNet-18 is 45.95% on CIFAR-10. In VGG-16 the relative percentage is 21.79% on CIFAR-10. The same behavior is observed with respect to ResNet-34, ResNet-50. This shows probabilistic AgrLearn has less time complexity than deterministic AgrLearn.

Table 6.11: Test error rates (%) of deterministic AgrLearn and probabilistic AgrLearn for ResNet-18 on CIFAR-10, and CIFAR-100

Dataset	**deterministic AgrLearn**		**probabilistic AgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.89	4.73	4.85	4.83
CIFAR-100	23.03	22.94	22.99	22.98

Table 6.12: Test error rates (%) of deterministic AgrLearn and probabilistic AgrLearn for ResNet-34 on CIFAR-10, and CIFAR-100

Dataset	**deterministic AgrLearn**		**probabilistic AgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.68	4.65	4.88	4.78
CIFAR-100	22.63	22.25	22.90	22.62

Table 6.13: Test error rates (%) of deterministic AgrLearn and probabilistic AgrLearn for ResNet-50 on CIFAR-10, and CIFAR-100

Dataset	**deterministic AgrLearn**		**probabilistic AgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.60	4.60	4.76	4.73
CIFAR-100	21.39	21.14	22.08	22.02

Table 6.14: Test error rates (%) of deterministic AgrLearn and probabilistic AgrLearn for WideResNet-22-10 on CIFAR-10, and CIFAR-100

Dataset	**deterministic AgrLearn**		**probabilistic AgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	4.19	4.18	4.25	4.21
CIFAR-100	20.30	20.28	20.77	20.75

Table 6.15: Test error rates (%) of deterministic AgrLearn and probabilistic AgrLearn for VGG-16 on CIFAR-10, and CIFAR-100

Dataset	**deterministic AgrLearn**		**probabilistic AgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	6.10	6.10	6.13	6.10
CIFAR-100	25.43	25.38	25.63	25.55

Table 6.16: Time complexity for dAgrLearn and pAgrLearn, fold-2, $\alpha > 0$ on CIFAR-10

Network	**dAgrLearn**	**pAgrLearn**	**Relative percentage (%)**
	1 epoch (second)	1 epoch (second)	
VGG-16	156	**34**	21.79
ResNet-18	148	**68**	45.95
ResNet-34	511	**102**	19.96
ResNet-50	808	**192**	23.76

Table 6.17: Time complexity for dAgrLearn, VGG-16, $\alpha > 0$ for 1 epoch (second)

Dataset	fold-1	fold-2	fold-4	fold-6	fold-8
CIFAR-10	61	79	133	192	248
CIFAR-100	55	79	134	192	248

Table 6.18: Time complexity for pAgrLearn, VGG-16, $\alpha > 0$ for 1 epoch (second)

Dataset	fold-1	fold-2	fold-4	fold-6	fold-8
CIFAR-10	13	16	25	35	44
CIFAR-100	12	17	26	35	44

In Tables 6.17 and 6.18 we also compare time complexity of dAgrLearn and pAgrLearn for different number of folds when only 50% of dataset on VGG-16 is used. Table 6.17 shows in dAgrLearn by increasing the number of folds from 1 to 2 the time complexity increases from 61 to 79 seconds and in the case of fold-8 it increases to 248 seconds for CIFAR-10. In pAgrLearn (Table 6.18) the time complexity from fold-1 to fold-8 increases from 13 to 44 seconds for CIFAR-10. This shows pAgrLearn has less time complexity than dAgrLearn. Moreover, it shows that increasing the number of folds increases the time complexity in the AgrLearn framework which is a limitation of this framework.

6.2.4 Model Behavior During Training

In this section, we track the model behavior during training, and we show how AgrLearn framework behaves in comparison with a standard neural network and the VIB framework.

6.2.4.a Deterministic AgrLearn

Figures 6.13 - 6.15 show training loss (cross-entropy loss, regularization term in dAgrLearn (MINE approach)), training and test error rate, training and test accuracy curves for CIFAR-10 and CIFAR-100 on ResNet-18, ResNet-34, ResNet-50, WideResNet-22-10, and VGG-16, for fold-2 across number of epochs.

It is seen from Figure 6.13 that the cross-entropy loss reduces as we increase the number of epochs. The training and test error rates decrease when the number of epochs is increased (Figure 6.14). The test error rate is higher than the training error rate. The training and test accuracy curves show the differences between training accuracy and test accuracy which indicate that test accuracy is less than training accuracy as expected (Figure 6.15).

We also compare the model behavior during training between two different fold numbers, fold-1 and fold-4. The typical behavior of ResNet-18 for CIFAR-10 and CIFAR-100 in terms of test error rate, across number of epochs, is shown in Figures 6.16 and 6.17, respectively. It is seen that in the "stable phase" of training, the test error for fold-4 (dotted purple curve) continues to decrease whereas the test error for fold-1 (solid purple curve) fails to improve further. This can be explained by the training loss curve of fold-1 (solid blue curve), which drops to zero quickly in this phase and provides no training signal for further tuning of the network parameters. In contrast, the training curve of the fold-4 (dotted blue curve) maintains a relatively high level, allowing the model to keep tuning itself. When the loss is low and one keeps the training, overfitting will occur, but in the case of fold-4 the loss is high and continuing training does not cause overfitting. The relatively higher training loss of fold-4 is due to the much larger space of aggregated examples. Even in the stable phase, one expects that the model is seeing new combinations of images.

6.2.4.b Probabilistic AgrLearn

The behavior with respect to training loss (cross-entropy loss in the case of pAgrLearn in which the regularization term is approximated by KL divergence), training and test error rate, training and test accuracy curves when probabilistic AgrLearn was applied over CIFAR-10 and CIFAR-100 on ResNet-18, ResNet-34, ResNet-50, WideResNet-22-10, and VGG-16, for fold-2 across the number of epochs are shown in Figures 6.18 - 6.20.

In Figure 6.18, it is seen that the cross-entropy loss decreases as the number of epochs increases. Figure 6.19 shows the error rate for train and test reduced by increasing the number of epochs. Therefore, as we see in Figure 6.20, training and test accuracy increases when the number of epochs is increased.

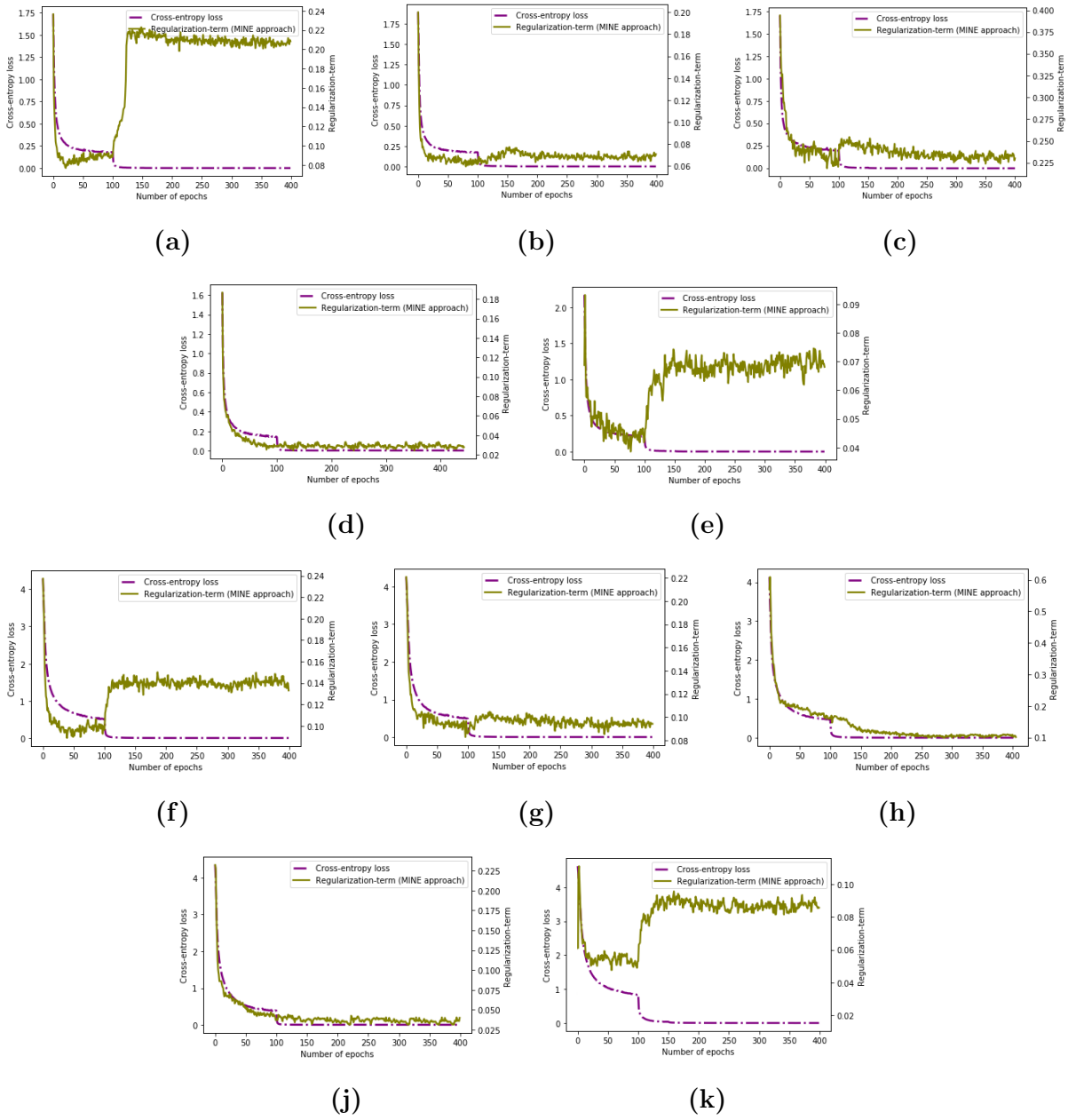


Figure 6.13: Training loss (cross-entropy loss and regularization-term (MINE approach)) on CIFAR-10 for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training loss (cross-entropy loss and regularization-term (MINE approach)) on CIFAR-100 for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

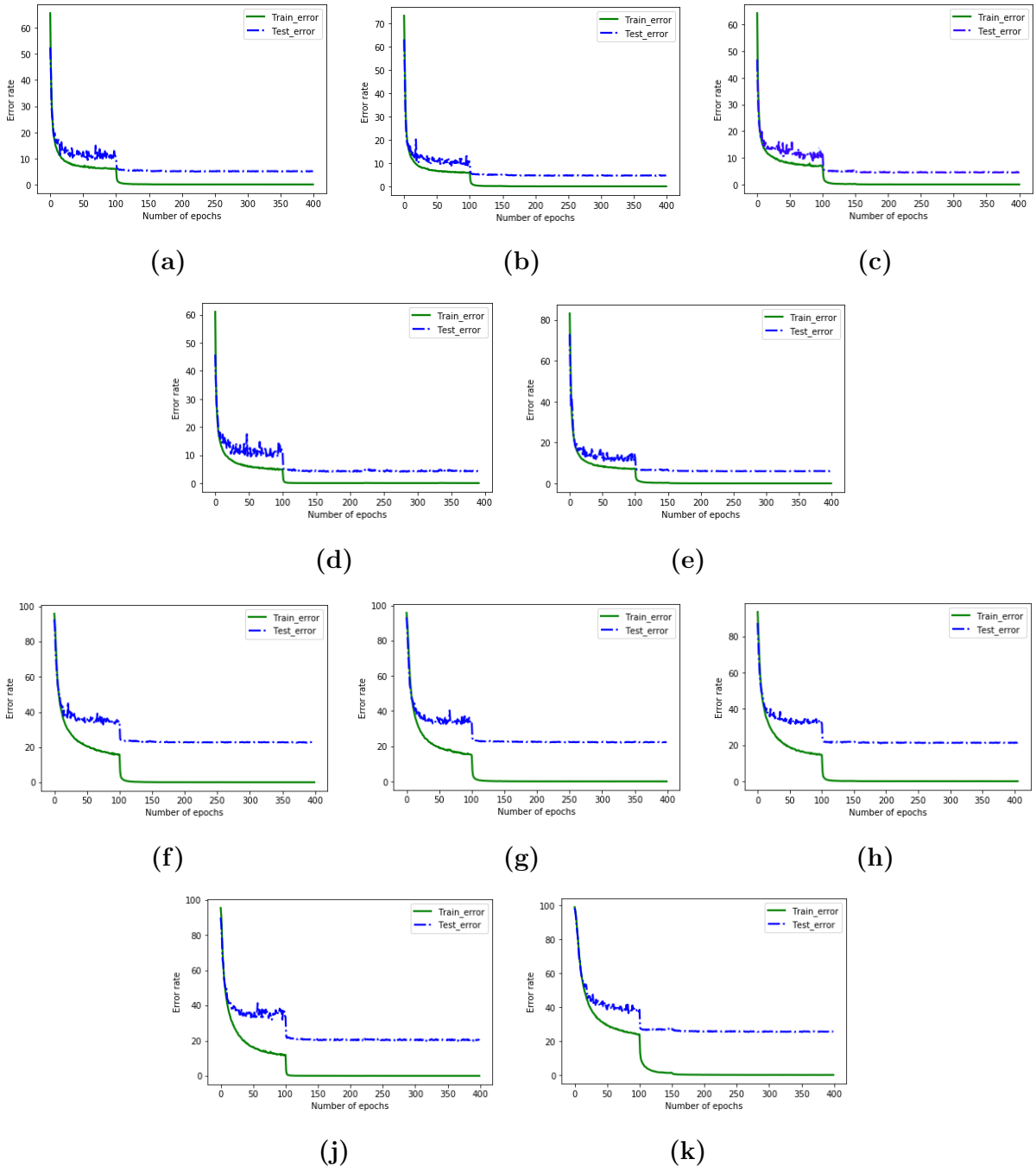


Figure 6.14: Training and test error rate on CIFAR-10 (dAgrLearn) for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training and test error rate on CIFAR-100 (dAgrLearn) for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

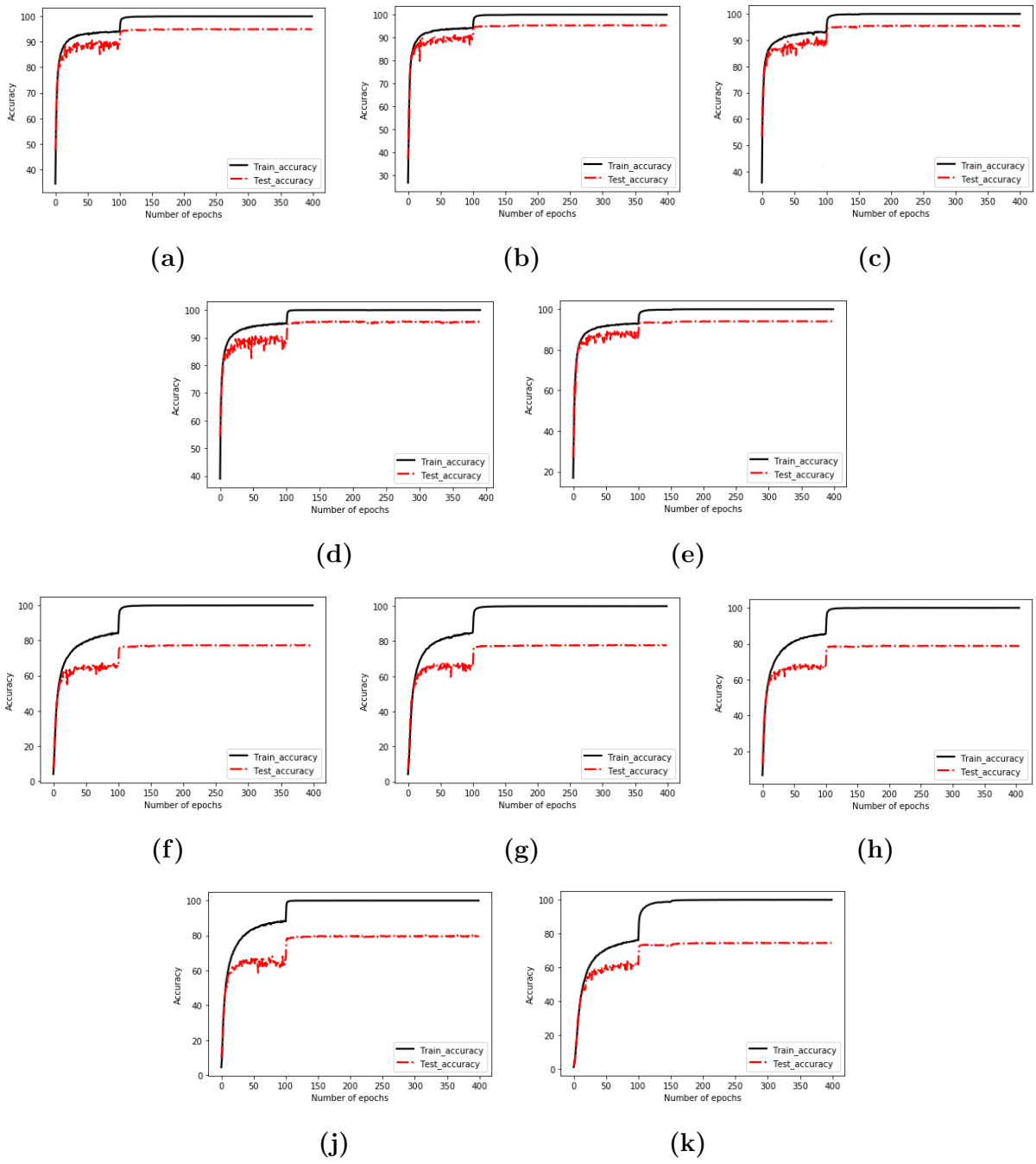


Figure 6.15: Training and test accuracy on CIFAR-10 (dAgrLearn) for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training and test accuracy on CIFAR-100 (dAgrLearn) for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

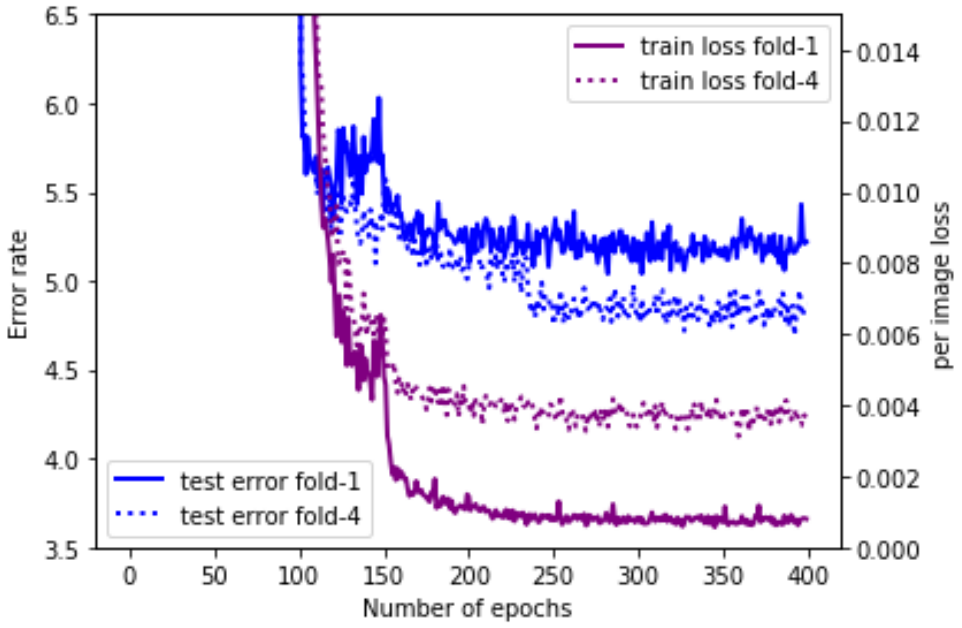


Figure 6.16: Training loss and test error of ResNet-18 on CIFAR-10 (dAgrLearn).

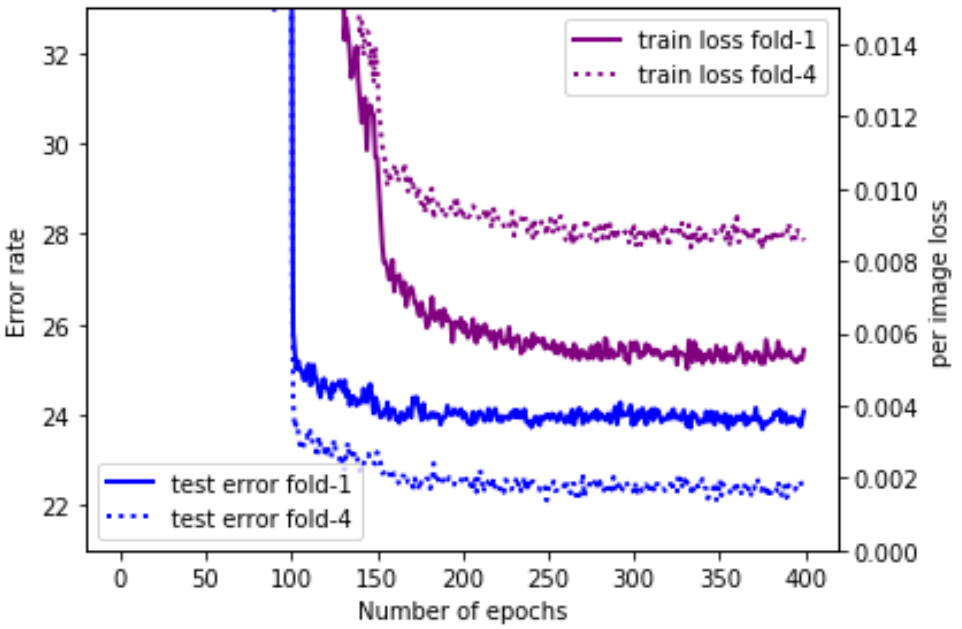


Figure 6.17: Training loss and test error of ResNet-18 on CIFAR-100 (dAgrLearn).

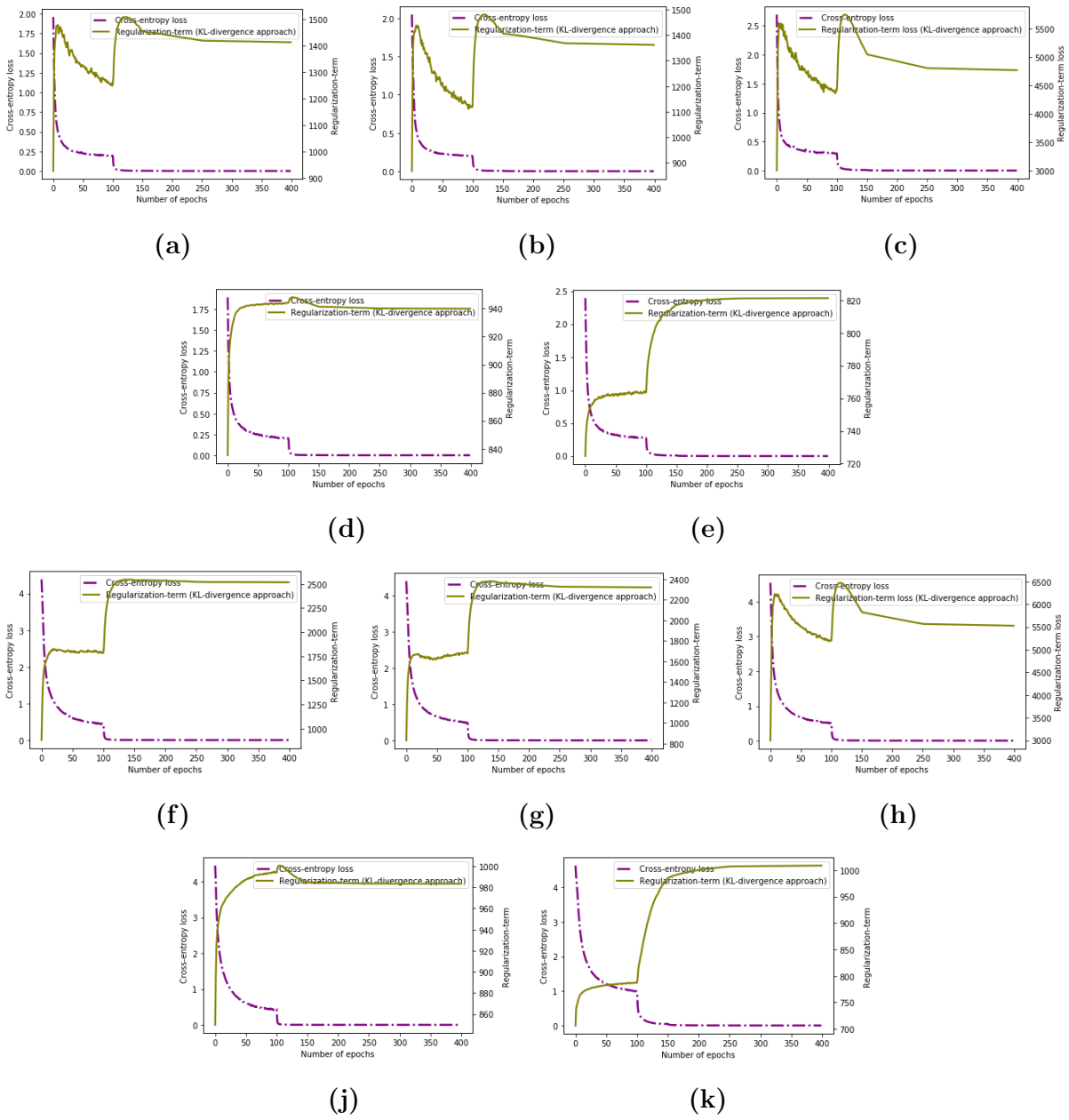


Figure 6.18: Training loss (cross-entropy loss and regularization-term (KL-divergence approach)) on CIFAR-10 for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training loss (cross-entropy loss and regularization-term (KL-divergence approach)) on CIFAR-100 for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

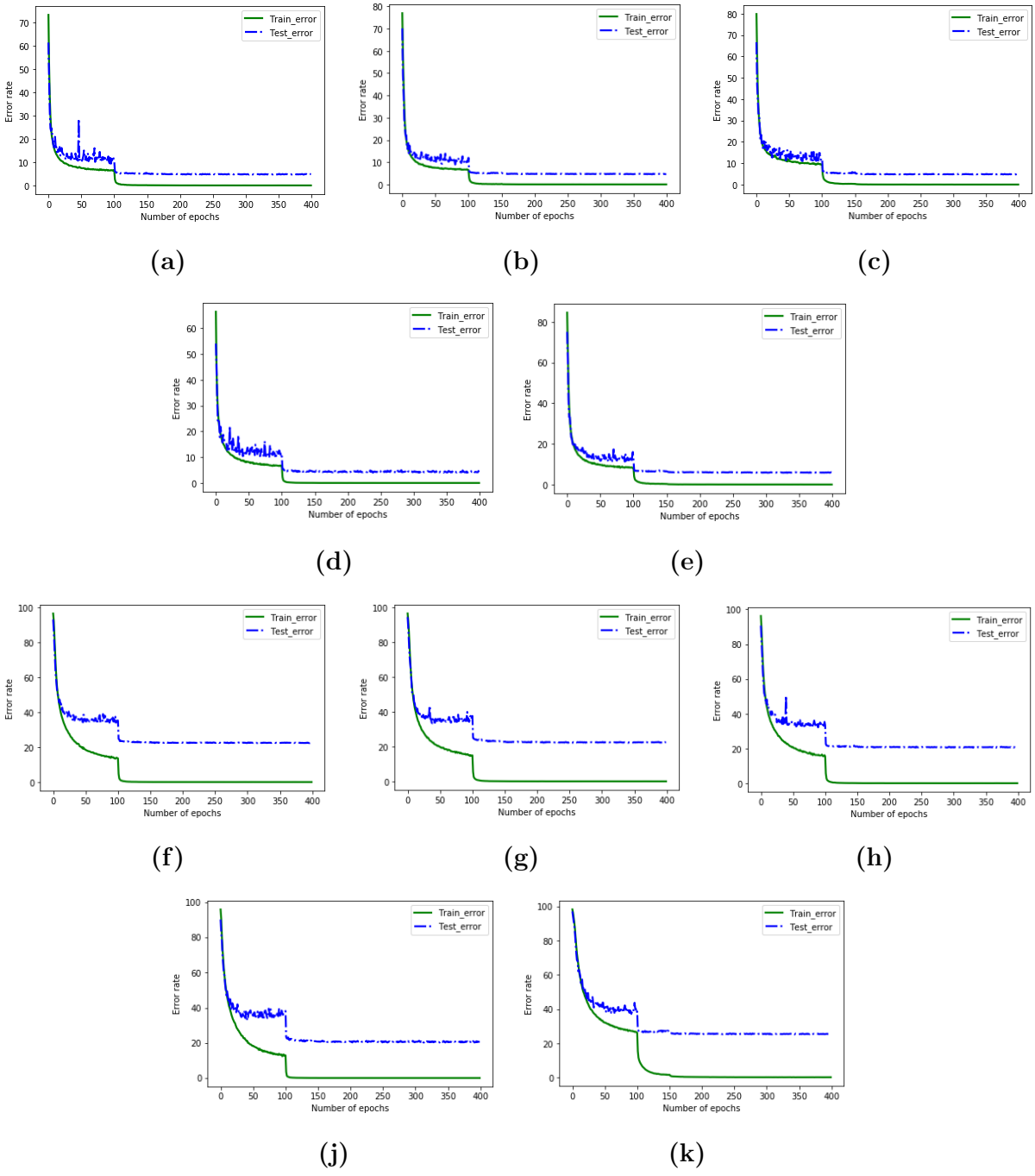


Figure 6.19: Training and test error rate on CIFAR-10 (pAgrLearn) for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training and test error rate on CIFAR-100 (pAgrLearn) for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

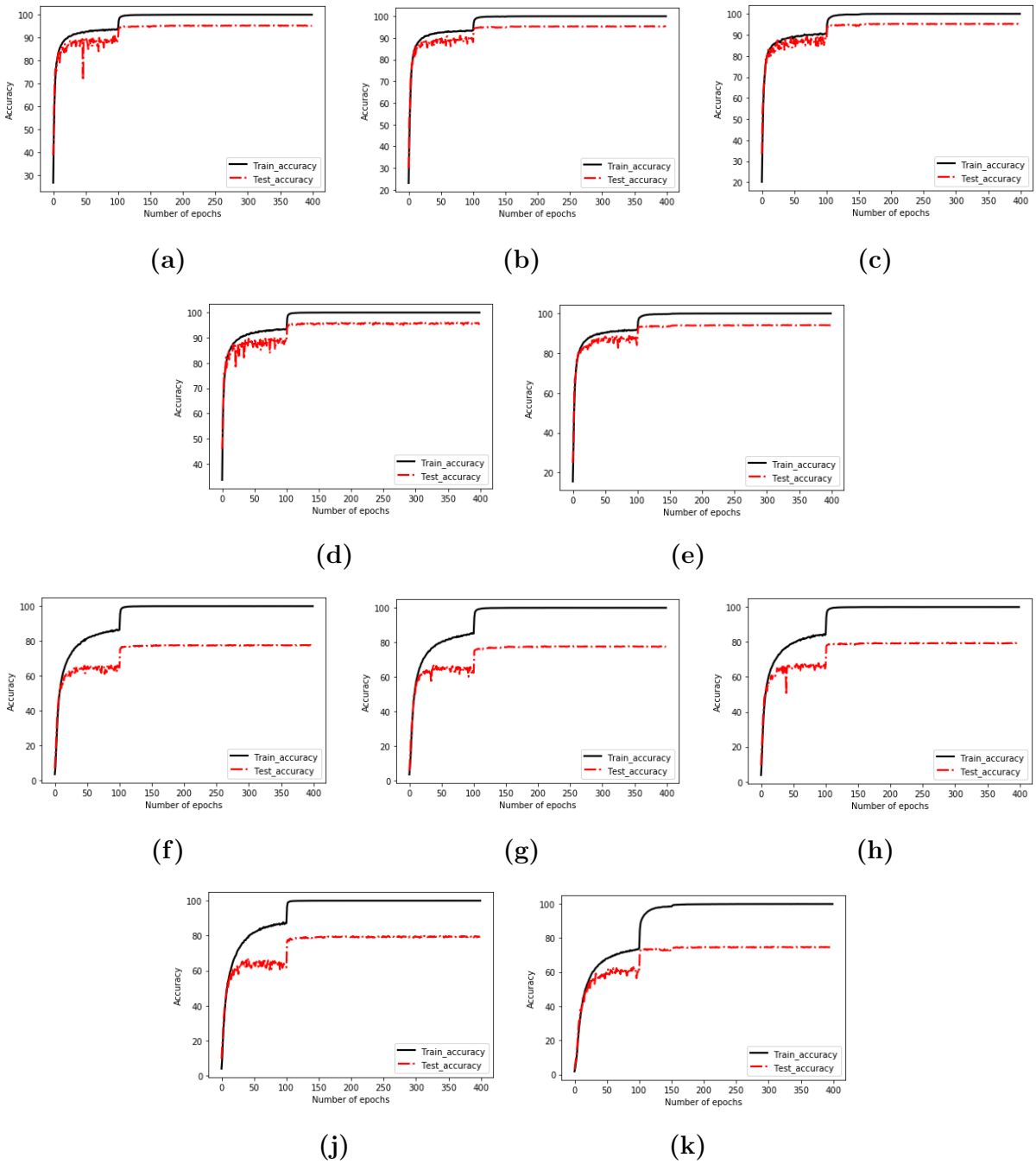


Figure 6.20: Training and test accuracy on CIFAR-10 (pAgrLearn) for **(a)** ResNet-18 **(b)** ResNet-34 **(c)** ResNet-50 **(d)** WideResNet-22-10 **(e)** VGG-16. Training and test accuracy on CIFAR-100 (pAgrLearn) for **(f)** ResNet-18 **(g)** ResNet-34 **(h)** ResNet-50 **(j)** WideResNet-22-10 **(k)** VGG-16.

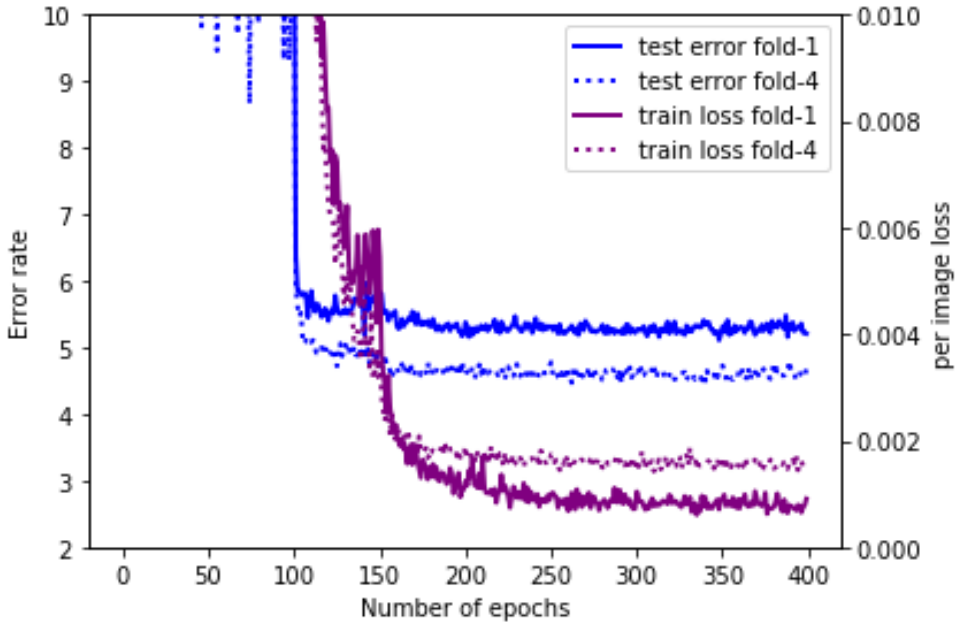


Figure 6.21: Training loss and test error of ResNet-18 on CIFAR-10 (pAgrLearn).

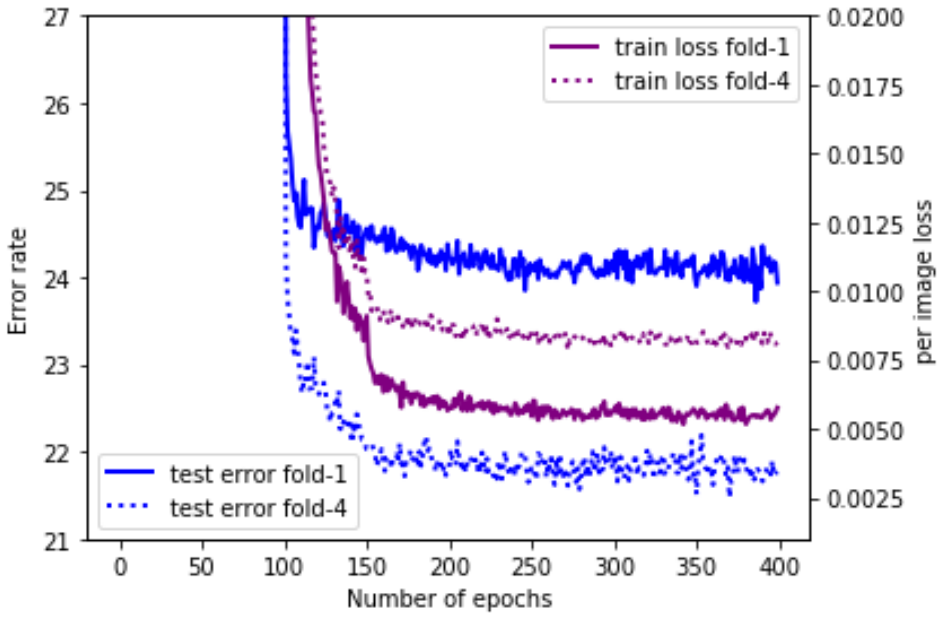


Figure 6.22: Training loss and test error of ResNet-18 on CIFAR-100 (pAgrLearn).

The typical behavior of ResNet-18 for CIFAR-10 and CIFAR-100, in pAgrLearn for fold-1 and fold-4, in terms of test error rate across number of epochs is shown in Figures 6.21 and 6.22, respectively. Comparing these figures with Figures 6.16 and 6.17 shows the same model behavior during training for fold-1 and fold-4 in both probabilistic AgrLearn and the deterministic AgrLearn. In these figures we also observe that the training curve of fold-4 (dotted blue curve) maintains a relatively high level, allowing the model to keep tuning itself.

6.2.5 Sensitivity to Model Complexity

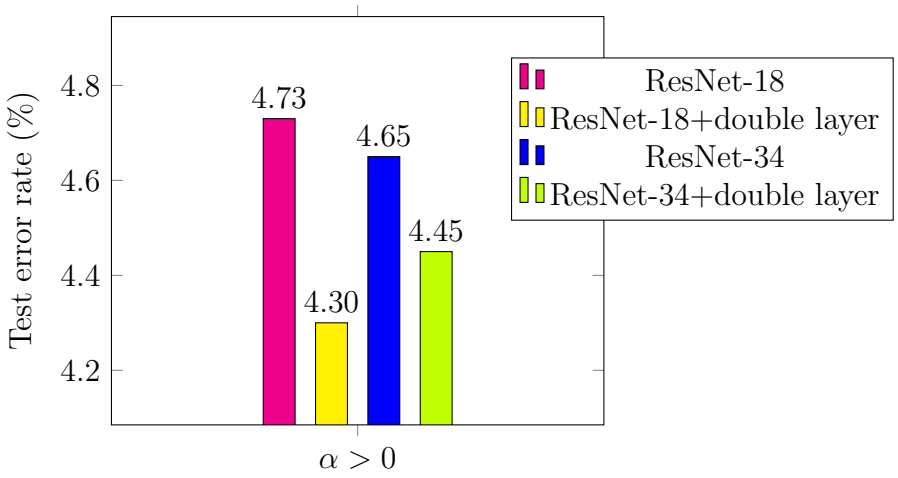
With fold-n AgrLearn, the output label space becomes $\mathcal{Y}^n$. This significantly larger label space seems to suggest that AgrLearn framework favors a more complex model. In this study, we investigate the behavior of the model when it becomes more complex.

6.2.5.a Deterministic AgrLearn

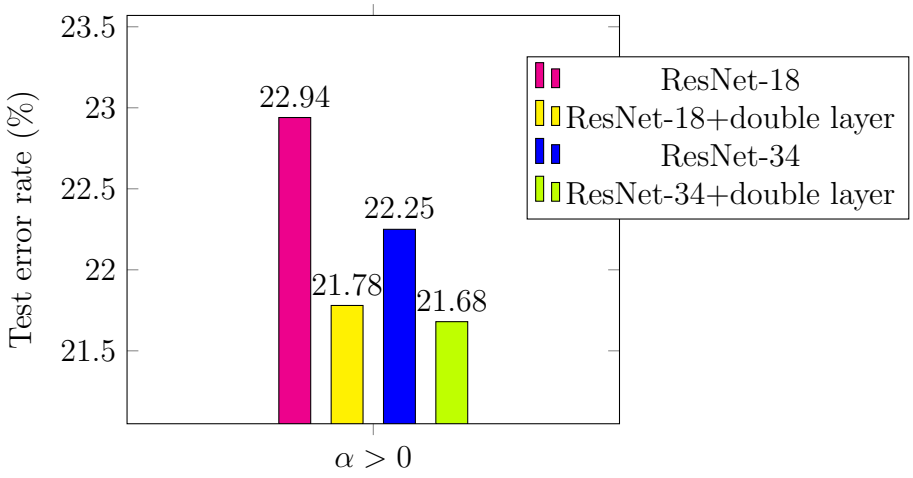
In dAgrLearn, we start with ResNet-18 for fold-2. The options we investigate include increasing the model width (by doubling the number of filters per layer) and increasing the model depth (from 18 layers to 34 layers). The performance of these models for dAgrLearn is given in Figure 6.23.

As shown in Figure 6.23, doubling the number of filters in ResNet-18 reduces the error rate for fold-2 from 4.73% to 4.30% on CIFAR-10, and from 22.94% to 21.78% on CIFAR-100. The same behavior regarding test error rate reductions is observed with respect to ResNet-34. These results show that increasing the model width of ResNet-18, and ResNet-34 improves the performance of AgrLearn on both CIFAR-10 and CIFAR-100.

When the model depth is increased from ResNet-18 to ResNet-34, the relative error reduces from ResNet-18 to ResNet-34 by 1.70%, and 3.00% respectively on CIFAR-10, and CIFAR-100, i.e., increasing the model depth improves performance in case of dAgrLearn.



(a) CIFAR-10

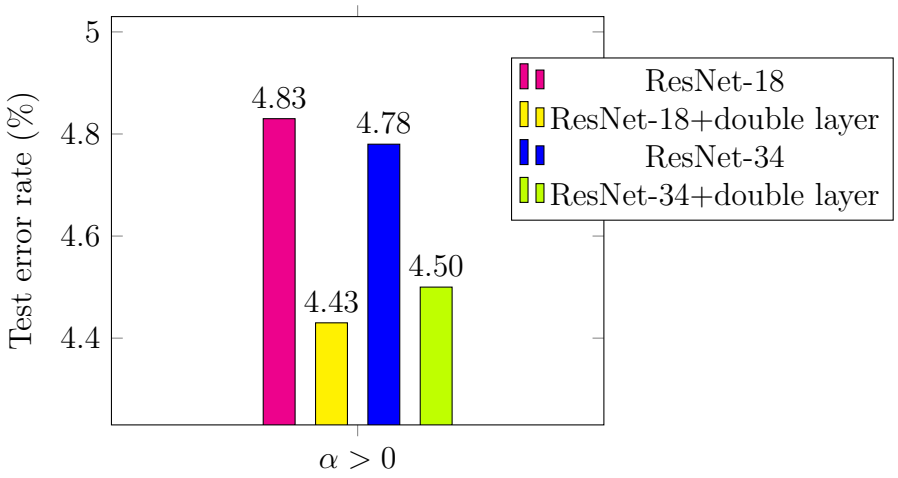


(b) CIFAR-100

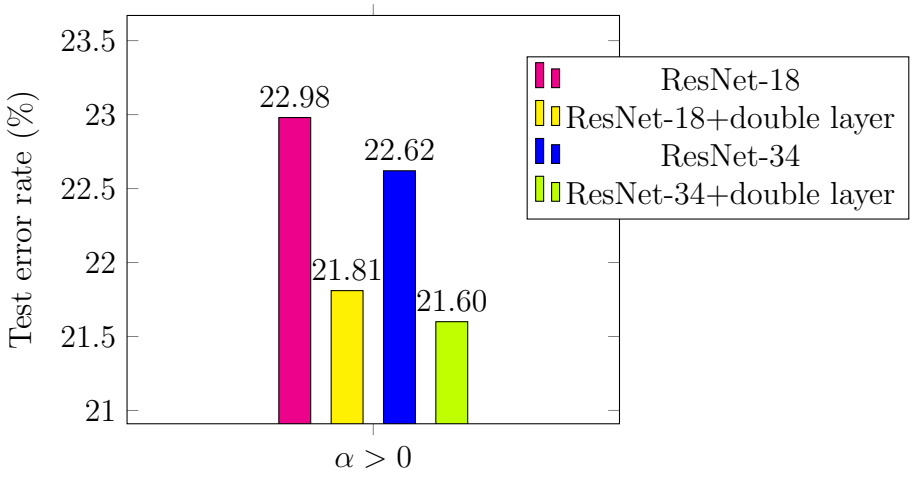
Figure 6.23: Test error rates (%) of ResNet-18 for fold-2, $\alpha > 0$ and its more complex variants (dAgrLearn) on (a) CIFAR-10 and (b) CIFAR-100.

6.2.5.b Probabilistic AgrLearn

Figure 6.24 shows the behavior of the model for probabilistic AgrLearn when the width of the model is increased by doubling the number of filters per layer and the depth of model is increased from 18 layers to 34 layers. This figure shows that, when the model width of ResNet-18 is increased, the error rate reduces from 4.83% to 4.43% for CIFAR-10, and from 22.98% to 21.81% for CIFAR-100. Therefore, we can conclude that the pAgrLearn shows



(a) CIFAR-10



(b) CIFAR-100

Figure 6.24: Test error rates (%) of ResNet-18 for fold-2, $\alpha > 0$ and its more complex variants (pAgrLearn) on (a) CIFAR-10 (b) CIFAR-100.

the same behavior as dAgrLearn with respect to increasing the model width. Regarding the model depth, test error rate decreases by increasing the model depth from ResNet-18 to ResNet-34, or from ResNet-18+double layer to ResNet-34+double layer, except for ResNet-34+double layer on CIFAR-10.

We hypothesize that with AgrLearn framework, the width of a model plays a critical role. This is because the input dimension in AgrLearn framework increases significantly and the model is required to extract joint features across individual objects in the amalgamated

example.

6.2.6 Behavior with Respect to Fold Number

In this section we illustrate the performance of AgrLearn framework when the number of folds is increased, meaning increasing the number of aggregated input objects.

6.2.6.a Deterministic AgrLearn

We also conducted experiments investigating the performance of ResNet-18 and VGG-16 with varying fold numbers. Tables 6.19 and 6.20 show the performance of ResNet-18 and VGG-16 for dAgrLearn. The relative error reductions on ResNet-18 and VGG-16 for fold-2 and fold-4 are shown in Figures 6.25 and 6.26 respectively.

The relative error reductions achieved with ResNet-18 for fold-4 with respect to fold-1, when α is equal to 0 are 4.72%, and 5.11% on CIFAR-10, CIFAR-100. Comparing these error reductions with error reductions for fold-2 with respect to fold-1 ($\alpha = 0$) which were 3.74%, and 2.83% on CIFAR-10, and CIFAR-100 respectively shows increasing the number of folds significantly boosted the performance. Similarly, for VGG-16, the error reductions for fold-4 in comparison with fold-2 is higher.

We also used another dataset SVHN to challenge ResNet-18 and VGG-16 with the dAgrLearn. The results are presented in Tables 6.19 - 6.20. It is seen from the Tables that dAgrLearn improves the performance of both ResNet-18 and VGG-16 networks on the SVHN dataset. Figure 6.25 part (e) shows the relative error reductions on ResNet-18 for fold-4 with respect to fold-1 for SVHN when $\alpha = 0$ and $\alpha > 0$ are 6.29% and 6.05% respectively.

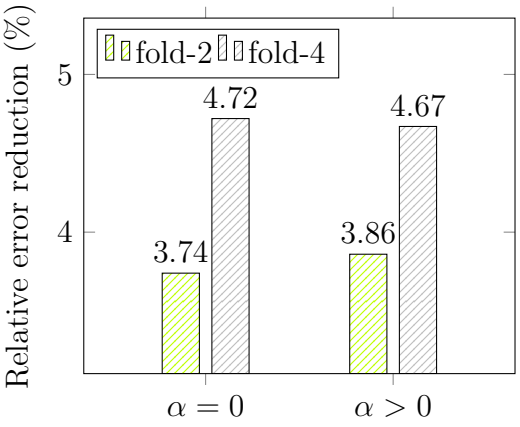
6.2.6.b Probabilistic AgrLearn

The performance of ResNet-18 and VGG-16 for probabilistic AgrLearn are depicted in Tables 6.21 and 6.22. The relative error reduction for fold-4 and fold-2 with respect to

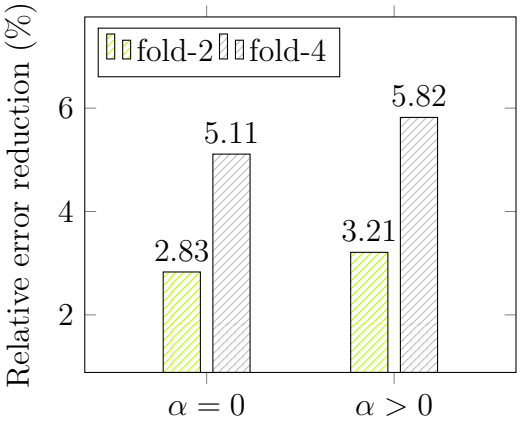
fold-1 is shown in these tables. The relative error reductions achieved on ResNet-18, for fold-4, when α is equal to 0 are 7.53%, and 4.81% on CIFAR-10, and CIFAR-100, and the relative error reductions achieved by fold-2 ($\alpha = 0$), are 6.37%, and 3.81% on CIFAR-10, and CIFAR-100. Similar improvement regarding VGG-16 is observed. The relative error reductions regarding SVHN show significant reductions by increasing the number of folds in the probabilistic AgrLearn. We observe in both dAgrLearn and pAgrLearn, increasing the number of folds improves performance significantly.

Table 6.19: Test error rates (%) of ResNet-18 (dAgrLearn) for varying fold numbers

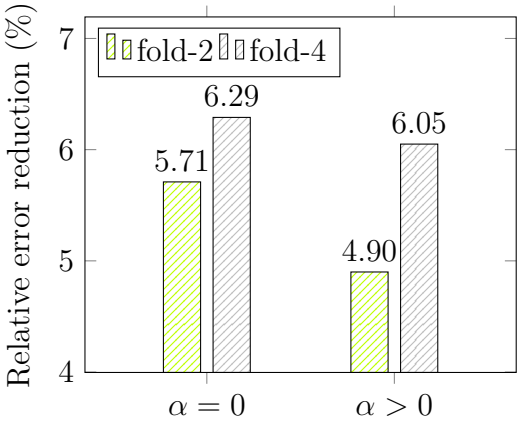
Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	5.08	4.92	4.89	4.73	4.84	4.69
CIFAR-100	23.70	23.70	23.03	22.94	22.49	22.32
SVHN	3.50	3.47	3.30	3.30	3.28	3.26



(c) CIFAR-10



(d) CIFAR-100

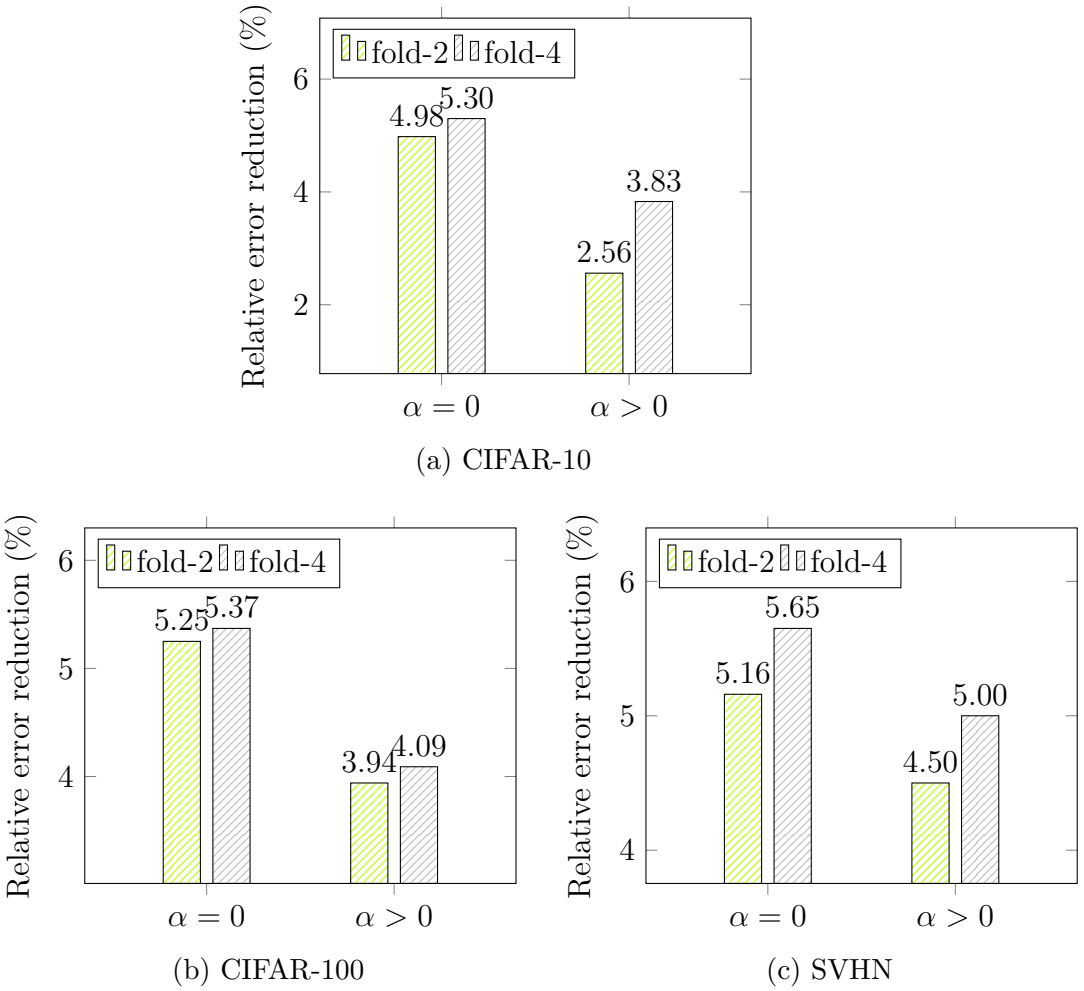


(e) SVHN

Figure 6.25: The relative error reductions achieved for ResNet-18 (dAgrLearn) for fold-2 and fold-4 with respect to fold-1 on (c) CIFAR-10 (d) CIFAR-100 and (e) SVHN.

Table 6.20: Test error rates (%) of VGG-16 (dAgrLearn) for varying fold numbers

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	6.42	6.26	6.10	6.10	6.08	6.02
CIFAR-100	26.84	26.42	25.43	25.38	25.4	25.34
SVHN	4.07	4.00	3.86	3.82	3.84	3.80



(a) CIFAR-10

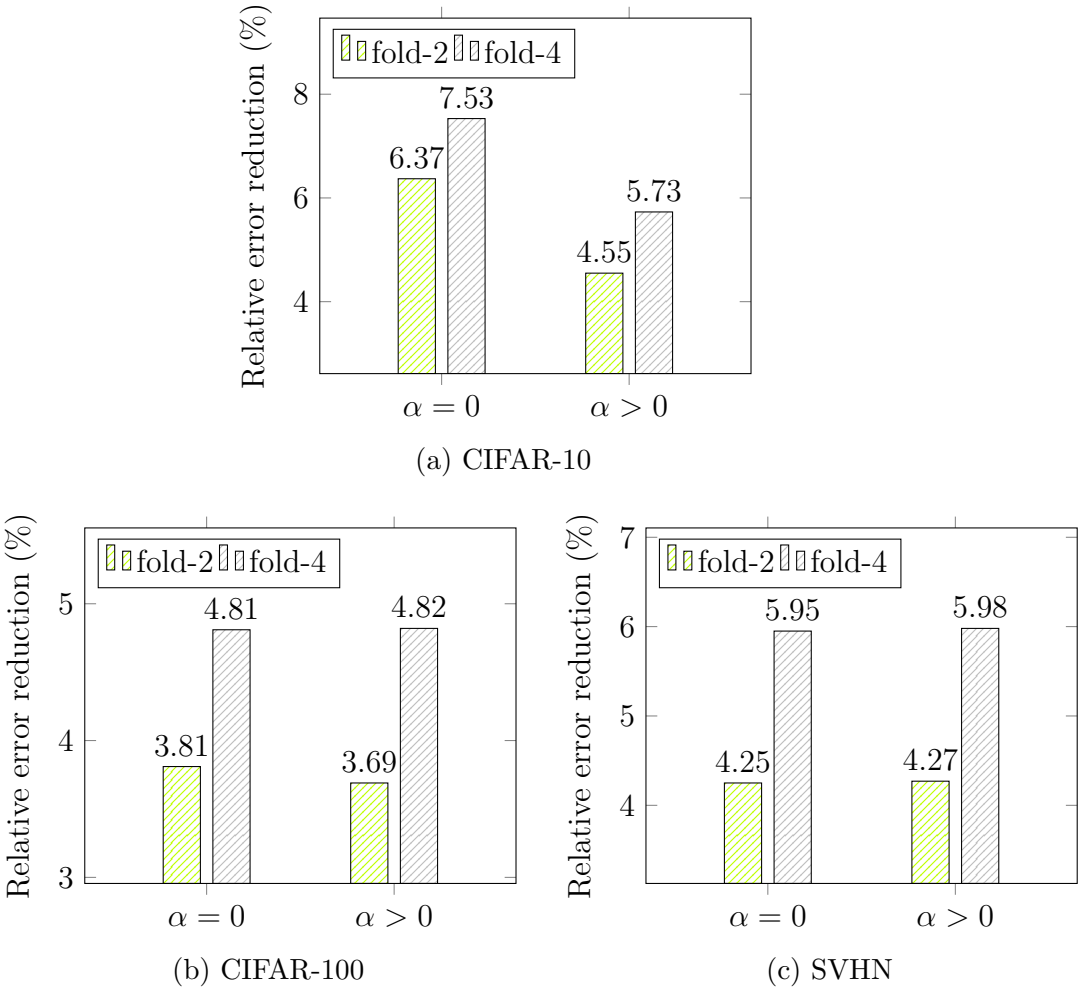
(b) CIFAR-100

(c) SVHN

Figure 6.26: The relative error reductions achieved for VGG-16 (dAgrLearn) for fold-2 and fold-4 with respect to fold-1 on (a) CIFAR-10 (b) CIFAR-100 and (c) SVHN.

Table 6.21: Test error rates (%) of ResNet-18 (pAgrLearn) for varying fold numbers

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$
CIFAR-10	5.18	5.06	4.85	4.83	4.79	4.77
CIFAR-100	23.90	23.86	22.99	22.98	22.75	22.71
SVHN	3.53	3.51	3.38	3.36	3.32	3.30



(a) CIFAR-10

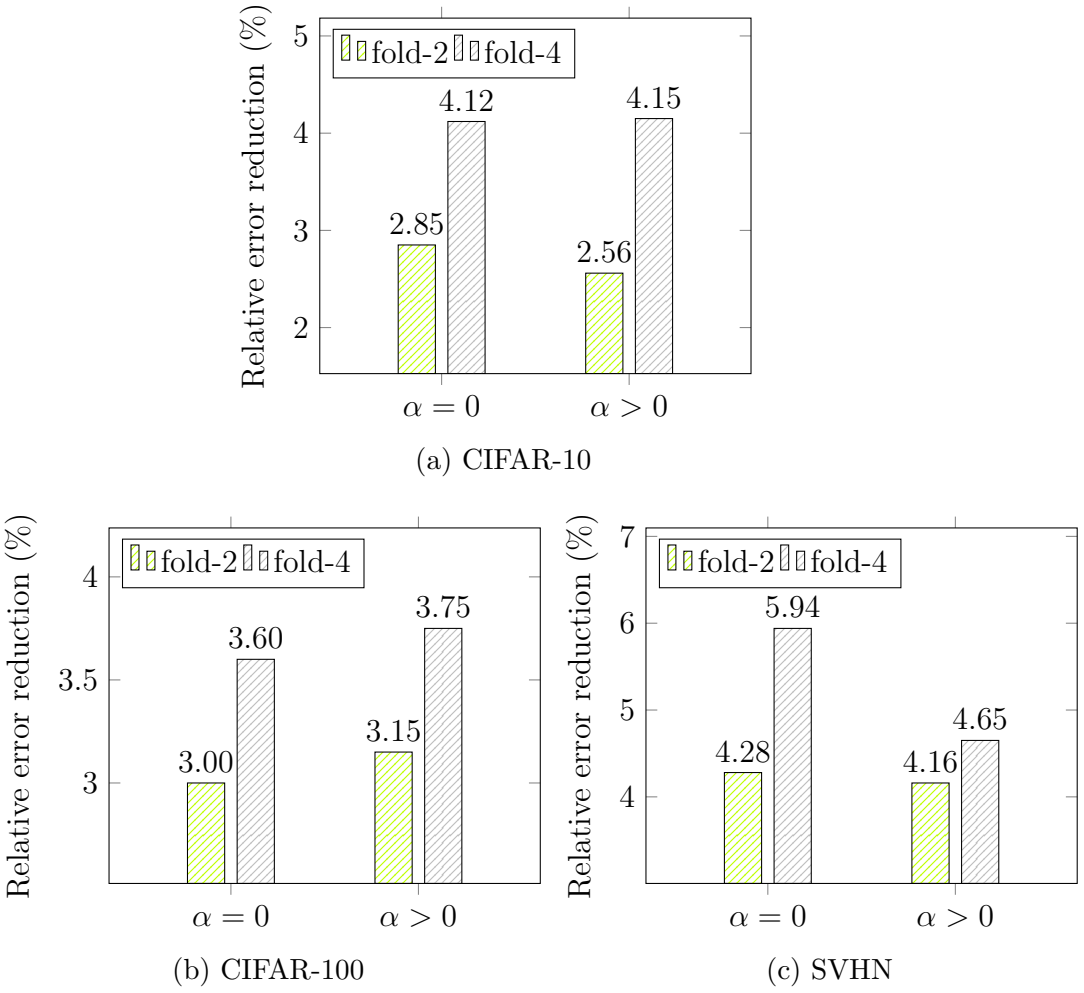
(b) CIFAR-100

(c) SVHN

Figure 6.27: The relative error reductions achieved for ResNet-18 (pAgrLearn) for fold-2 and fold-4 with respect to fold-1 on (a) CIFAR-10 (b) CIFAR-100 and (c) SVHN.

Table 6.22: Test error rates (%) of VGG-16 (pAgrLearn) for varying fold numbers

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$
CIFAR-10	6.31	6.26	6.13	6.10	6.05	6.00
CIFAR-100	26.42	26.38	25.63	25.55	25.47	25.39
SVHN	4.21	4.09	4.03	3.92	3.96	3.90



(a) CIFAR-10

(b) CIFAR-100

(c) SVHN

Figure 6.28: The relative error reductions achieved for VGG-16 (pAgrLearn) for fold-2 and fold-4 with respect to fold-1 on (a) CIFAR-10 (b) CIFAR-100 and (c) SVHN.

Table 6.23: Test error rates (%) of ResNet-18 for varying fold numbers for CIFAR-10 with only 25% dataset

	fold-1		**fold-2**		**fold-4**		**fold-6**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
dAgrLearn	11.74	11.65	11.30	11.29	11.08	11.07	10.94	10.94
pAgrLearn	11.69	11.66	11.35	11.34	11.19	11.13	11.06	11.04

Table 6.24: Test error rates (%) of VGG-16 (pAgrLearn) for varying fold numbers with only 50% dataset

Dataset	**fold-1**		**fold-2**		**fold-4**		**fold-6**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$
CIFAR-100	34.85	34.68	33.03	32.96	32.97	32.86	32.95	32.84

Table 6.25: Test error rates (%) of VGG-16 (pAgrLearn) for varying fold numbers with only 25% dataset

Dataset	**fold-1**		**fold-2**		**fold-4**		**fold-6**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$
CIFAR-10	13.61	13.59	13.10	13.10	12.92	12.89	12.91	12.87

We also investigate the performance of AgrLearn framework when the number of folds is increased to 6. Table 6.23 shows the test error rates for dAgrLearn and pAgrLearn on ResNet-18 for CIFAR-10 when only 25% of dataset is used. In the case of dAgrLearn, test error rate for fold-6 decreases to 10.94% while for fold-2 it decreases to 11.30%.

Moreover, we show the performance of pAgrLearn on VGG-16 for CIFAR-100 when 50% (Table 6.24) and for CIFAR-10 when 25% (Table 6.25) of dataset are used. These tables show that by increasing the number of folds, test error rate reduces more. Therefore we can conclude the performance of AgrLearn improves by increasing the number of folds. In the next section, we investigate the effect of data scarcity on the performance of AgrLearn framework.

6.2.7 Robustness to Data Scarcity

In this section, we investigate the performance of AgrLearn framework when only 50% of a dataset is used.

6.2.7.a Deterministic AgrLearn

In Tables 6.26 - 6.27 using only 50% of a dataset, we investigate the performance of ResNet-18 and VGG-16 for fold-1, fold-2 and fold-4 for dAgrLearn. The relative error reductions that were obtained with ResNet-18 for fold-4 when $\alpha > 0$ are 4.58% and 6.23% for CIFAR-10 and CIFAR-100. The reductions for fold-2 are 3.66% and 5.43%, respectively. Similar behavior is observed regarding VGG-16 with using just 50% of the training dataset.

In Figure 6.29 we compare the performance of dAgrLearn on ResNet-18 when using just 50% of the training dataset with its performance when using the entire training dataset, for CIFAR-10 and CIFAR-100. In Figure 6.29 we can see that when we use only 50% of the training dataset the error is reduced by increasing the number of folds. This is similar to the case when the entire training dataset is used. The performance of the dAgrLearn when used on VGG-16 was also similar, as shown in Figure 6.30. We can conclude that dAgrLearn performs well even when using only half of the training dataset.

6.2.7.b Probabilistic AgrLearn

Tables 6.28 and 6.29 show the performance of ResNet-18 and VGG-16, for pAgrLearn when 50% of training dataset are used. The relative error reductions achieved with ResNet-18 for fold-4 are 3.38%, and 8.17% for CIFAR-10 and CIFAR-100 when using 50% of training dataset. These reductions for fold-2 are 2.73%, and 7.20%, respectively.

In Figures 6.31 and 6.32, the performance of pAgrLearn on ResNet-18 and VGG-16 for fold-1, fold-2 and fold-4 ($\alpha > 0$) when using only 50% of the training dataset is compared with its performance when the whole training dataset is used. In these figures, we can see by increasing the number of folds from 1 to 2 and from 2 to 4 the greater error reductions are achieved.

Table 6.26: Test error rates (%) of ResNet-18, its dAgrLearn counterparts with a fraction of training dataset on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	7.78	7.65	7.39	7.37	7.37	7.30
CIFAR-100	31.54	31.48	29.89	29.77	29.59	29.52

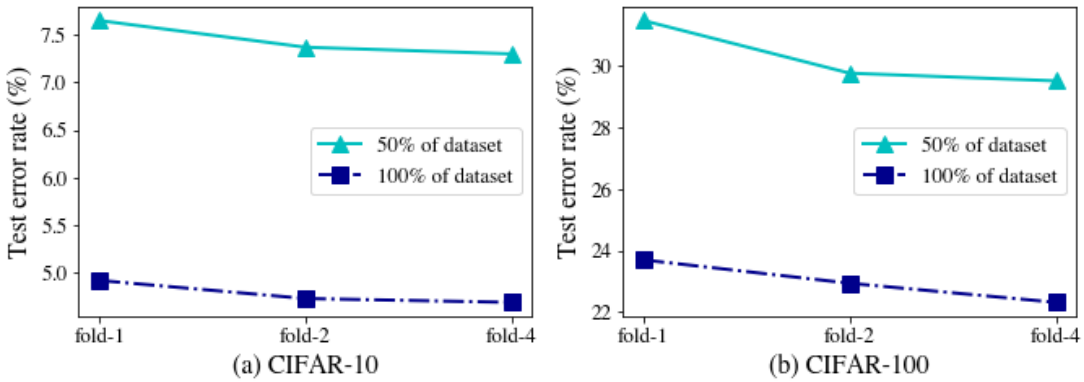


Figure 6.29: Comparison between test error rates (%) of ResNet-18 and its dAgrLearn counterparts ($\alpha > 0$) when 50% and 100% of training dataset is used on (a) CIFAR-10 and (b) CIFAR-100.

The results demonstrate the robustness of pAgrLearn to data scarcity, and this makes dAgrLearn and pAgrLearn appealing solutions to practical learning problems with inadequate labeled data.

Table 6.27: Test error rates (%) of VGG-16, its dAgrLearn counterparts with a fraction of training dataset on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	9.12	9.10	9.05	8.99	8.91	8.89
CIFAR-100	34.63	34.5	33.04	32.98	32.75	32.70

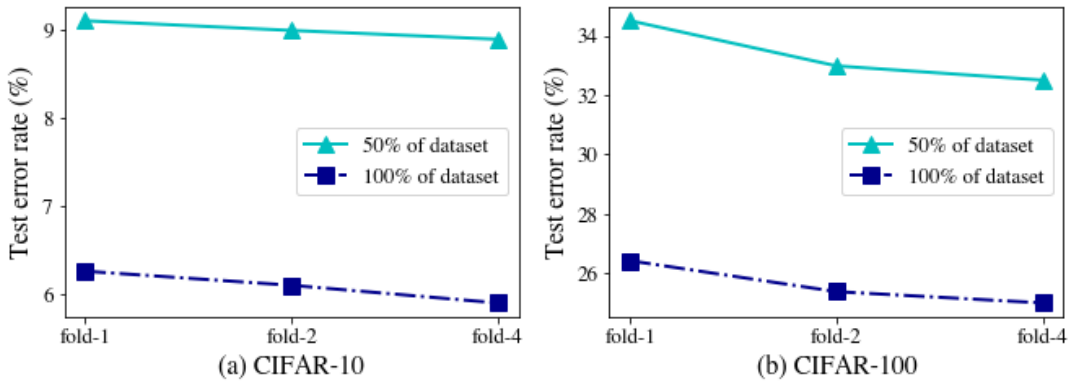


Figure 6.30: Comparison between test error rates (%) of VGG-16 and its dAgrLearn counterparts ($\alpha > 0$) when 50% and 100% of training dataset is used on (a) CIFAR-10 and (b) CIFAR-100.

Table 6.28: Test error rates (%) of ResNet-18, its pAgrLearn counterparts with a fraction of training dataset on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	7.74	7.69	7.52	7.48	7.48	7.43
CIFAR-100	32.47	32.08	29.83	29.77	29.64	29.46

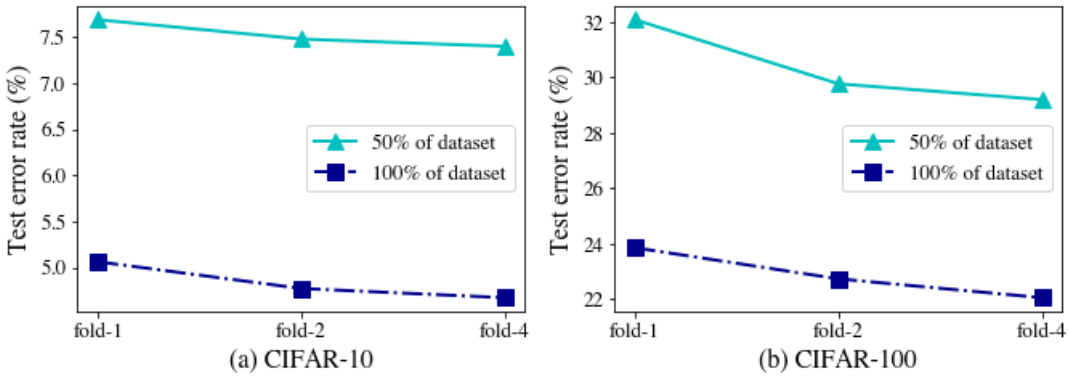


Figure 6.31: Comparison between test error rates (%) of ResNet-18 and its pAgrLearn counterparts ($\alpha > 0$) when 50% and 100% of training dataset is used on (a) CIFAR-10 and (b) CIFAR-100.

Table 6.29: Test error rates (%) of VGG-16, its pAgrLearn counterparts with a fraction of training dataset on CIFAR-10, and CIFAR-100

Dataset	**fold-1**		**fold-2**		**fold-4**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	9.31	9.28	8.95	8.93	8.84	8.80
CIFAR-100	34.85	34.65	33.03	32.96	32.97	32.86

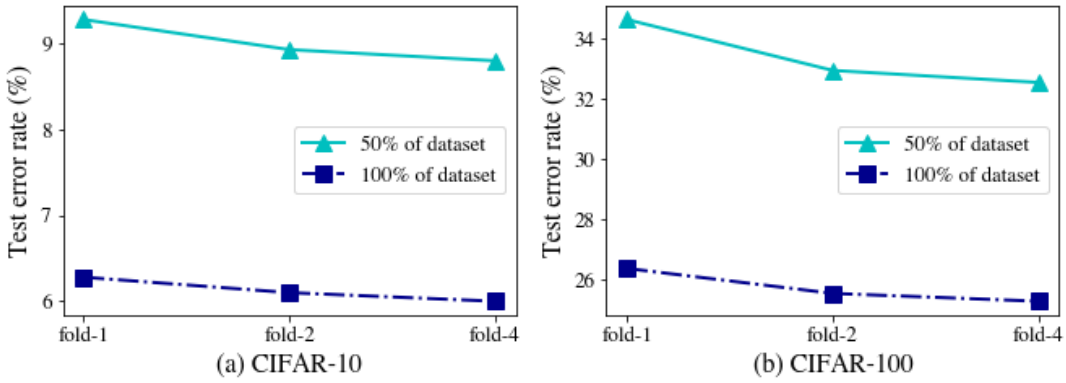


Figure 6.32: Comparison between test error rates (%) of VGG-16 and its pAgrLearn counterparts ($\alpha > 0$) when 50% and 100% of training dataset is used on (a) CIFAR-10 and (b) CIFAR-100.

Table 6.30: Test error rates (%) of ResNet-18+MixUp and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	fold-1		fold-2		RER	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	3.99	3.98	3.95	3.90	1.00	2.01
CIFAR-100	21.51	21.48	20.62	20.58	4.14	4.19

6.2.8 Behavior by Adding Regularization Approaches

In this section we apply recently proposed data augmentation techniques such as MixUp [85] and Between Class Learning (BC Learning) [75] over AgrLearn framework. We use the same ResNet and VGG architecture.

6.2.8.a Deterministic AgrLearn with MixUp

MixUp augments the training data with synthetic training examples, each obtained by interpolating a pair of original examples and their corresponding labels.

Tables 6.30 and 6.31 show the test error rates of ResNet-18 and VGG-16 for dAgrLearn with adding MixUp. In these tables, we show the relative error reduction (RER) for fold-2 with respect to fold-1 for each dataset. As we can see, dAgrLearn is compatible with MixUp, and increasing the number of folds decreases test error rates. In addition, in Table 6.30 it is seen that the test error rates for fold-2 when $\alpha = 0$ for ResNet-18 with MixUp are 3.95%, and 20.62% on CIFAR-10, and CIFAR-100 respectively. And without MixUp, they are 4.89%, and 23.03%. The same behavior regarding the test error rate with respect to VGG-16 is observed in Table 6.31. It is seen that adding MixUp over the dAgrLearn has improved the performance of ResNet-18 and VGG-16.

6.2.8.b Probabilistic AgrLearn with MixUp

Tables 6.32 and 6.33 show the test error rates of ResNet-18 and VGG-16 for the pAgrLearn with adding MixUp. The relative error reduction for fold-2 for each dataset is shown in these

Table 6.31: Test error rates (%) of VGG-16+MixUp and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	5.49	5.47	5.37	5.27	2.19	3.66
CIFAR-100	25.19	24.92	24.28	24.20	3.61	2.89

Table 6.32: Test error rates (%) of ResNet-18+MixUp and its pAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	3.96	3.92	3.90	3.86	1.52	1.53
CIFAR-100	21.71	21.70	20.96	20.91	3.45	3.64

Table 6.33: Test error rates (%) of VGG-16+MixUp and its pAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	**fold-1**		**fold-2**		**RER**	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	5.73	5.69	5.49	5.37	4.19	5.62
CIFAR-100	25.34	25.24	24.25	24.00	4.30	4.91

tables. Similar to dAgrLearn, pAgrLearn is also compatible with MixUp and increasing the number of folds decreases test error rates. Furthermore, the error rates for fold-2 when $\alpha = 0$ for VGG-16 with MixUp are 5.49%, and 24.25% on CIFAR-10, and CIFAR-100 respectively. The error rates for VGG-16 without MixUp for fold-2 when $\alpha = 0$ are 6.13%, and 25.63% on CIFAR-10, and CIFAR-100 respectively. This shows performance is boosted for VGG-16 by using MixUp. Therefore, we can conclude MixUp boosts the performance of pAgrLearn.

Table 6.34: Test error rates (%) of ResNet-18+BC Learning and its dAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	fold-1		fold-2		RER	
	$\alpha = 0$	$\alpha = 0.7$	$\alpha = 0$	$\alpha = 0.3$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	3.89	3.88	3.82	3.80	1.80	2.06
CIFAR-100	21.60	21.32	20.40	20.39	5.56	4.36

6.2.8.c Deterministic AgrLearn with BC Learning

BC Learning technique [75] is another data augmentation method that we applied over the same ResNet-18 architecture for dAgrLearn. The BC Learning technique augments training data with synthetic training examples, each obtained by mixing two images belonging to different classes with a random ratio.

Table 6.34 shows the test error rate of ResNet-18 for the dAgrLearn with BC Learning technique. The test error rates for fold-2 when $\alpha = 0$ in ResNet-18 with BC Learning technique are 3.82%, and 20.40% on CIFAR-10, and CIFAR-100 respectively. These error rates with respect to ResNet-18 without BC Learning technique are 4.89%, and 23.03%. It is seen that the BC Learning technique improves the performance of dAgrLearn significantly.

6.2.8.d Probabilistic AgrLearn with BC Learning

Table 6.35 illustrates the test error rate of ResNet-18 for the pAgrLearn with BC Learning technique. This shows pAgrLearn is compatible with BC Learning technique and increasing the number of folds decreases test error rates. In addition, in this table, we show the relative error reduction for fold-2 for each dataset. The test error rates for fold-2 when $\alpha = 0$ in ResNet-18 with BC Learning technique are 3.89%, and 20.90% on CIFAR-10, and CIFAR-100 respectively. It is seen that the BC Learning technique improves the performance of pAgrLearn significantly.

We can conclude both MixUp and BC Learning techniques boost the performance of dAgrLearn and pAgrLearn, and decrease the test error rate.

Table 6.35: Test error rates (%) of ResNet-18+BC Learning and its pAgrLearn counterparts on CIFAR-10 and CIFAR-100

Dataset	fold-1		fold-2		RER	
	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha = 10^{-6}$	$\alpha = 0$	$\alpha > 0$
CIFAR-10	3.92	3.83	3.89	3.81	0.77	0.52
CIFAR-100	21.81	21.70	20.90	20.79	4.17	4.19

6.3 Text Classification with Aggregated Learning

In this section, we show the application of AgrLearn for text classification, specifically sentiment analysis. Sentiment analysis was introduced as a series of techniques to extract information from documents such as opinion, attitude, or emotion, e.g. positive or negative, bullish or bearish, and anger or happiness [61, 54, 59].

Sentiment analysis classifies a given text in the document into two or three classes, e.g. positive, negative, and neutral [78, 62, 16]. The positive and negative polarity can be shown by mapping to $[-1, 1]$ such that 1 shows positive and -1 shows negative. For example, " `a thoughtful, insistently humanizing film .`" shows a positive review and it can be labeled by 1.

The main application of this technique is in the analysis of the reviews of products or services. This is very helpful to companies in finding out the opinions of their customers, their likes, and dislikes, whether they are satisfied or dissatisfied with a product or a service. They can also use this type of information to try and improve their products.

The importance of this task was a motivation for us to investigate how the AgrLearn framework would perform in this environment. We investigated the performances of both classes of the framework (dAgrLearn and pAgrLearn) when performing sentiment analysis.

In order to apply the AgrLearn framework to sentiment analysis, we set some hyper-parameters. The mini-batch size is taken to be 50, the learning rate is set to 0.001. We use dropout with rate of 0.5.

Experiments were conducted on benchmark sentence-classification datasets, Movie

Table 6.36: Confusion Matrix

		Prediction	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Review [60], Multi-Perspective Question Answering corpus (MPQA) [79], Customer Review [35], Yelp Review [49] and Amazon Review [49].

Movie Review is a dataset with one sentence per review. It has 10,662 sentences with binary labels [4]. The MPQA dataset consists of opinion polarity from news articles and private states such as emotions, beliefs, etc., and has 10,606 sentences [79] [5]. Customer Review contains reviews of different products such as cameras, MP3s, etc. [35]. It has 3,775 sentences [6]. Yelp Review and Amazon Review [49] contain restaurant reviews and amazon.com reviews respectively, with sentences that have a clearly positive or negative connotation, each one has 1,000 sentences [7]. We use 10% of random examples in each dataset for testing and the rest for training.

We apply AgrLearn framework to a widely used NLP deep learning architecture, Convolutional Neural Networks (CNN). We adopt CNN-sentence [44] and implement it exactly as in [45]. The final feature map of CNN is passed to a logistic regression classifier for label prediction. Each sentence enters the model via a learnable, randomly initialized word-embedding dictionary. All sentences are zero-padded to the same length. In the AgrLearn framework for CNN, the aggregation of two sentences in each input simply involves concatenating the two zero-padded sentences.

To quantify the evaluation metrics a confusion matrix can be used as shown in Table 6.36. In the confusion matrix, the actual and predicted labels are shown.

In this matrix, True Positive (TP) is defined when both actual and predicted labels are

[4] https://www.cs.cornell.edu/people/pabo/movie-review-data/

[5] http://www.cs.pitt.edu/mpqa/

[6] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[7] http://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

positive, True Negative (TN) when both actual and predicted labels are negative and False Positive (FP) or False Negative (FN) is defined when the actual label is negative while the predicted label is positive, or when the actual label is positive and the predicted label is negative, respectively. For sentiment analysis task, we use accuracy as an evaluation metric, that is defined as the ratio of correctly predicted samples to total observations:

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}$$

Each reported performance value is obtained by averaging that value after running the programme at the same setting with different random seed initialization, 7 times.

6.3.1 Predictive Performance

In this section we illustrate the performance of the AgrLearn framework for sentiment analysis task, using benchmark sentence-classification datasets on a widely used deep learning architecture.

6.3.1.a Deterministic AgrLearn

We train and test CNN, and its respective dAgrLearn counterparts. We use multiple datasets, and the achieved performances are reported in Table 6.37.

The performance of dAgrLearn on CNN is shown in Table 6.37. The achieved relative improvement rates with respect to CNN for fold-2 when $\alpha = 0$ are 7.27% for Movie Review, 6.99% for Customer Review, and 4.40% for Amazon Review. And when $\alpha > 0$ the relative improvement rates (RIR) for fold-2 are 8.66% for Movie Review, 7.00% for Customer Review, and 4.28% for Amazon Review. Similar improvement is observed in the case of other datasets as well. The results show that the performance of dAgrLearn is superior to the performance of its corresponding CNN counterpart when it comes to sentiment analysis.

Table 6.37: Test Accuracy (%) obtained by CNN, its respective dAgrLearn counterpart and relative improvement rate (RIR) for fold-2 with respect to fold-1

Dataset	fold-1		fold-2		RIR	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
Movie Review	73.84	73.91	79.21	80.31	7.27	8.66
MPQA	83.80	83.90	87.21	87.34	4.07	4.10
Customer Review	79.88	80.00	85.47	85.6	6.99	7.00
Yelp Review	80.07	80.19	81.67	81.83	2.00	2.05
Amazon Review	82.10	82.30	85.71	85.82	4.40	4.28

6.3.1.b Probabilistic AgrLearn

We train and test the CNN and its respective pAgrLearn counterparts. We use multiple datasets, and the achieved performances are reported in Figures 6.33 - 6.34. In these figures, fold-2 denotes pAgrLearn and fold-1 denotes the VIB framework.

The achieved relative improvement rates with respect to CNN for fold-2 when $\alpha > 0$ are 3.82% for Movie Review, 8.63% for Customer Review, and 8.60% for Amazon Review respectively. And when $\alpha = 0$ the relative improvement rates for fold-2 are 4.70% for Movie Review, 7.96% for Customer Review, and 7.71% for Amazon Review. Similar improvement is observed in the case of the other datasets as well. The results show that the performance of pAgrLearn (fold-2) is superior to the performance of VIB framework (fold-1) when it comes to sentiment analysis.

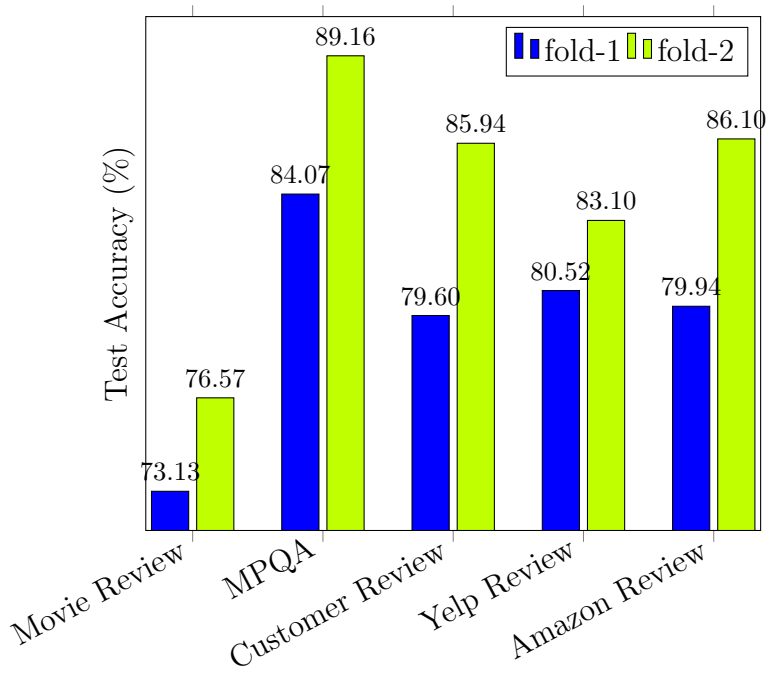


Figure 6.33: Test Accuracy (%) obtained by CNN, with VIB (fold-1) and pAgrLearn (fold-2) when $\alpha = 0$.

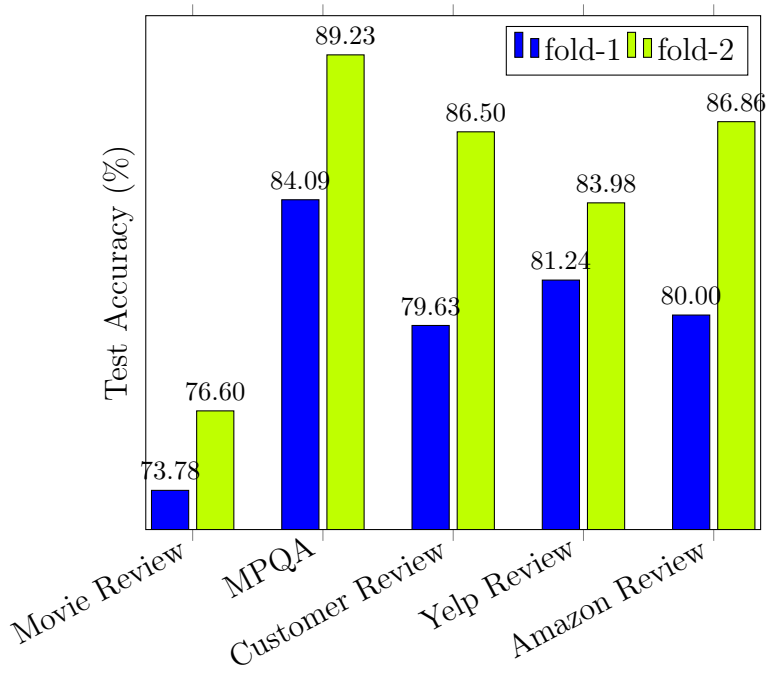


Figure 6.34: Test Accuracy (%) obtained by CNN, with VIB (fold-1) and pAgrLearn (fold-2) when $\alpha > 0$.

Table 6.38: (t, p)-values for pAgrLearn with respect to CNN on (a) Movie Review, (b) MPQA, (c) Customer Review, (d) Yelp Review, (e) Amazon Review.

(a) (t, p)-values for Movie Review

fold-2, $\alpha = 0$	$(4.62, 6.24 \times 10^{-4})$	$(3.49, 2.53 \times 10^{-3})$
fold-2, $\alpha > 0$	$(6.74, 1.59 \times 10^{-5})$	$(4.56, 4.08 \times 10^{-4})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(b) (t, p)-values for MPQA

fold-2, $\alpha = 0$	$(7.69, 8.34 \times 10^{-6})$	$(9.57, 2.86 \times 10^{-7})$
fold-2, $\alpha > 0$	$(8.35, 1.61 \times 10^{-5})$	$(10.89, 3.62 \times 10^{-7})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(c) (t, p)-values for Customer Review

fold-2, $\alpha = 0$	$(7.67, 4.84 \times 10^{-6})$	$(10.83, 3.81 \times 10^{-7})$
fold-2, $\alpha > 0$	$(6.27, 3.06 \times 10^{-5})$	$(7.38, 7.59 \times 10^{-5})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(d) (t, p)-values for Yelp Review

fold-2, $\alpha = 0$	$(1.89, 4.51 \times 10^{-2})$	$(1.97, 3.73 \times 10^{-2})$
fold-2, $\alpha > 0$	$(1.90, 4.07 \times 10^{-2})$	$(1.88, 4.37 \times 10^{-2})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

(e) (t, p)-values for Amazon Review

fold-2, $\alpha = 0$	$(3.00, 6.02 \times 10^{-3})$	$(3.57, 1.92 \times 10^{-3})$
fold-2, $\alpha > 0$	$(3.67, 2.56 \times 10^{-3})$	$(4.54, 4.22 \times 10^{-4})$
Compared with	fold-1, $\alpha = 0$	fold-1, $\alpha > 0$

In Table 6.38 we also report the t-value and the p-value with respect to CNN. The reported p-value, for fold-1 and fold-2 when $\alpha = 0$ is equal to 6.24×10^{-4}, and when $\alpha > 0$ is equal to 4.08×10^{-4} for Movie Review. The same behavior is observed with respect to other datasets. Therefore, we conclude with a high degree of confidence that the results of fold-2 and fold-1 are significantly different.

6.3.1.c Probabilistic AgrLearn vs Standard Neural Network

Figures 6.35 - 6.36 show the comparison between probabilistic AgrLearn and the standard neural network for CNN. The standard neural network is simply fold-1 in the dAgrLearn where there is no aggregation among input objects. The relative improvement rates with respect to probabilistic AgrLearn for CNN when $\alpha = 0$ are 6.40% for MPQA, 7.59% for Customer Review, and 3.78% for Yelp Review respectively. Similar improvement is observed in the case of other datasets as well. The results show that the performance of probabilistic AgrLearn is superior to the performance of the standard neural network with respect to CNN when it comes to sentiment analysis.

6.3.1.d Probabilistic AgrLearn vs Deterministic AgrLearn

We also compare pAgrLearn with dAgrLearn in Table 6.39. The test accuracy in pAgrLearn with respect to CNN when $\alpha = 0$ for Customer Review and Amazon Review are 85.94% and 86.10% respectively. The test accuracy in the dAgrLearn with respect to CNN when $\alpha = 0$ for Customer Review and Amazon Review are 85.47% and 85.71%. These results show the test accuracy in some cases increases more by using pAgrLearn.

6.4 Summary

In this chapter, we investigated the application of AgrLearn framework to image recognition and NLP. We showed the performance of both classes of the AgrLearn framework (dAgrLearn and pAgrLearn). We have demonstrated the difference between both classes of the AgrLearn

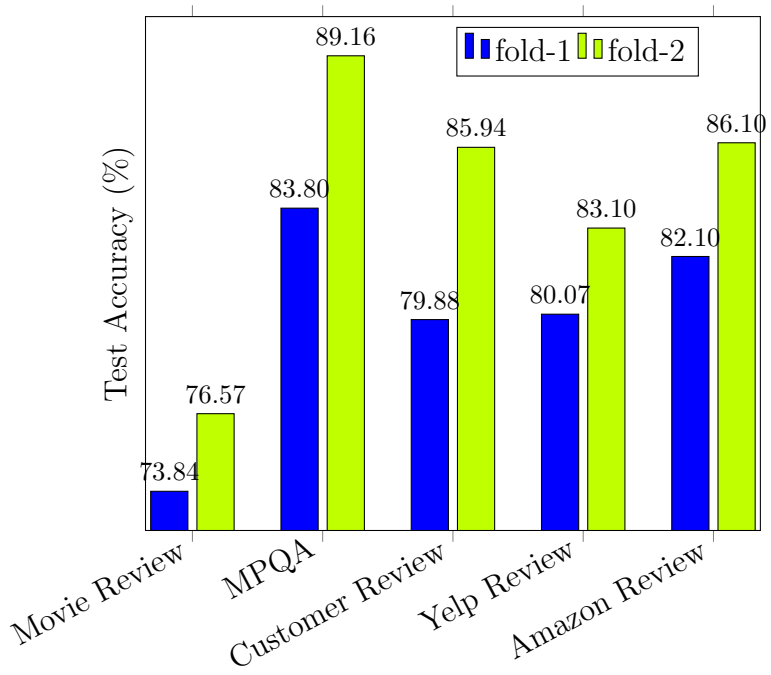


Figure 6.35: Test Accuracy (%) for Standard Neural Network (fold-1) and probabilistic AgrLearn (fold-2) on CNN when $\alpha = 0$.

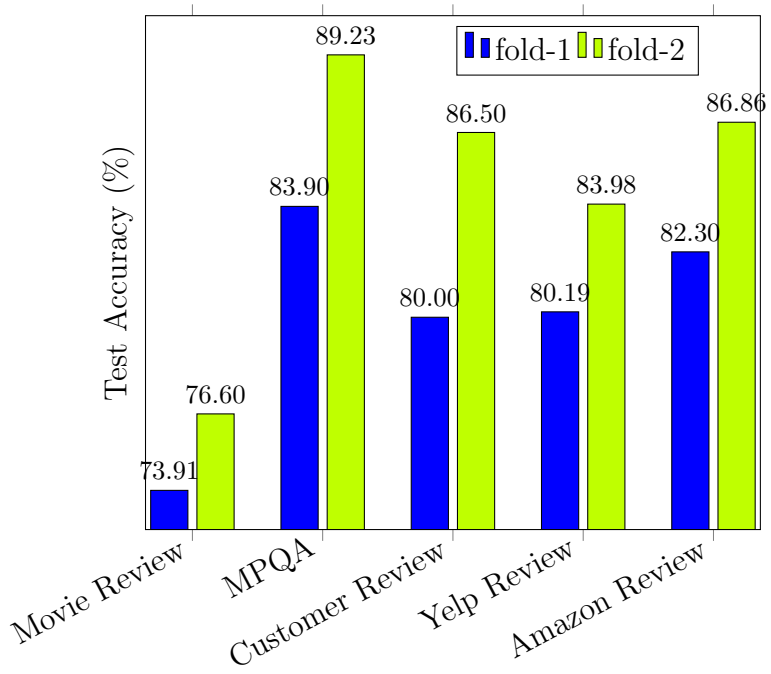


Figure 6.36: Test Accuracy (%) for Standard Neural Network (fold-1) and probabilistic AgrLearn (fold-2) on CNN when $\alpha > 0$.

Table 6.39: Test Accuracy (%) obtained by CNN with respective to pAgrLearn and dAgrLearn model

Dataset	**pAgrLearn**		**dAgrLearn**	
	$\alpha = 0$	$\alpha > 0$	$\alpha = 0$	$\alpha > 0$
Movie Review	76.57	76.60	79.21	80.31
MPQA	89.16	89.23	87.21	87.34
Customer Review	85.94	86.50	85.47	85.60
Yelp Review	83.10	83.98	81.67	81.83
Amazon Review	86.10	86.86	85.71	85.82

framework, deterministic AgrLearn and probabilistic AgrLearn.We have shown that both dAgrLearn and pAgrLearn bring significant performance gain to the standard deep network models in classification tasks.

The results indicate that both dAgrLearn and pAgrLearn show a significant improvement in performance compared to a standard neural network. In addition, the pAgrLearn framework has better performance in comparison with the VIB framework. We also investigated the effect of increasing number of folds, changing the network architectures, using only a portion of the training dataset, and applying different regularization approaches in dAgrLearn and pAgrLearn. The results show:

- The model width improves performance in the case of dAgrLearn and pAgrLearn.
- In both dAgrLearn and pAgrLearn, increasing the number of folds improves performance significantly.
- dAgrLearn and pAgrLearn are robust to data scarcity, therefore they can be appealing solutions to practical learning problems with inadequate labeled data.
- dAgrLearn and pAgrLearn are compatible with regularization approaches such as MixUp. The performance of them is boosted by using MixUp and BC Learning techniques.

We also have demonstrated the performance of dAgrLearn and pAgrLearn by applying them to sentiment analysis tasks. In addition to this, we have shown the superiority of the pAgrLearn over the VIB framework for sentiment analysis tasks. The application of

AgrLearn is possible to be extended for other tasks, and models on NLP. It is interesting to investigate how the AgrLearn framework performs and what additional consideration is needed in that case.

Chapter 7

Conclusion and Future Plans

In this chapter, first, we summarize the major contributions of the thesis, and then we highlight the road-map for future research.

7.1 Conclusion

In this thesis, we have proposed a novel framework which we call Aggregated Learning or AgrLearn to tackle the problem of learning a neural network classifier. It is a simple and effective neural network modeling framework, based on information theory. It builds on an equivalence between IB learning and IB quantization and exploits the power of vector quantization, which is well known in information theory. We have proposed two classes of AgrLearn, deterministic AgrLearn (dAgrLearn) and probabilistic AgrLearn (pAgrLearn).

We have demonstrated the effectiveness of the AgrLearn framework through the significant performance gain it brings to the current state-of-the-art of deep network models. We evaluated its performance by applying it to image recognition and natural language processing (NLP) tasks.

In image recognition task, the results show both dAgrLearn and pAgrLearn perform better than standard neural networks with respect to ResNet-18, ResNet-34, ResNet-50, WideResNet-22-10, and VGG-16 neural networks. The pAgrLearn has less time complexity

in comparison with dAgrLearn. It has better performance than dAgrLearn in some cases such as ResNet-18. In addition, the performance of the pAgrLearn framework is better than the performance of the VIB framework.

We have shown that increasing the number of folds significantly boosts the performance of both dAgrLearn and pAgrLearn. In addition, the width of a network plays a critical role in their performances. The results showed that both dAgrLearn and pAgrLearn demonstrated robustness when challenged with data scarcity; therefore, they can be appealing solutions to practical learning problems with inadequate labeled data. Adding regularization approaches such as MixUp and BC Learning were also investigated and the results showed that these techniques boost the performance of both dAgrLearn and pAgrLearn.

In text classification tasks, specifically sentiment analysis, the results obtained from our simulations show that the performances of both dAgrLearn and pAgrLearn are superior to that of standard neural networks with respect to CNN. In addition, the pAgrLearn performs better than VIB for CNN.

We have established the AgrLearn framework theoretically and we have clearly shown the superiority of this framework over standard neural networks by conducting extensive experiments. We should mention the AgrLearn framework can be combined with most of the other techniques that are used for improving the training of neural networks which are based on single input and the corresponding output.

7.2 Future Research Directions

The investigations and conclusions in this thesis give rise to several ideas that can be considered for future research. One idea for future research is to investigate how we characterize the interaction between model complexity, fold number, and sample size in AgrLearn framework.

We have shown that aggregation of inputs provides additional freedom in the architectural design of networks. This opens up another area of research to investigate how best can such freedom be exploited.

The successful performance of AgrLearn framework on classification problems is also a motivation for further research to investigate how this framework behaves in the case of imbalanced datasets.

References

[1] A. Achille and S. Soatto. Emergence of invariance and disentanglement in deep representations. *Journal of Machine Learning Research (JMLR)*, 19(1):1947–1980, 2018.

[2] A. Achille and S. Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2897–2905, 2018.

[3] R. Ahlswede and J. Korner. Source coding with side information and a converse for degraded broadcast channels. *IEEE Transactions on Information Theory*, 21(6):629–637, 1975.

[4] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations (ICLR)*, 2017.

[5] Rana Ali Amjad and Bernhard Claus Geiger. Learning representations for neural network-based classification using the information bottleneck principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[6] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

[7] M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, A. Courville, and R. D. Hjelm. Mine: mutual information neural estimation. In *International Conference on Machine Learning (ICML)*, 2018.

[8] J. Cardinal. Compression of side information. In *2003 International Conference on Multimedia and Expo. (ICME)*, 2003.

[9] M. Chalk, O. Marre, and G. Tkacik. Relevant sparse codes with variational information bottleneck. In *Advances in Neural Information Processing Systems*, 2016.

[10] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss. Information bottleneck for gaussian variables. *Journal of Machine Learning Research (JMLR)*, 6(Jan):165–188, 2005.

[11] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. Variational lossy autoencoder. In *International Conference on Learning Representations (ICLR)*, 2017.

[12] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381, 1989.

[13] T. A. Courtade and R. D. Wesel. Multiterminal source coding with an entropy-based distortion measure. In *2011 IEEE International Symposium on Information Theory Proceedings*, 2011.

[14] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[15] B. Dai, C. Zhu, B. Guo, and D. P. Wipf. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning, (ICML)*, 2018.

[16] L. Dong, F. Wei, S. Liu, M. Zhou, and K. Xu. A statistical parsing framework for sentiment classification. *Computational Linguistics*, 41(2):293–336, 2015.

[17] M. D. Donsker and S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time, I. *Communications on Pure and Applied Mathematics*, 36(2):183–212, 1983.

[18] A. El Gamal and Y. Kim. *Network information theory*. Cambridge university press, 2011.

[19] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[20] N. Friedman, O. Mosenzon, N. Slonim, and N. Tishby. Multivariate information bottleneck. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 2001.

[21] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Springer Science and Business Media, 2012.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.

[23] Ziv Goldfeld, Ewout van den Berg, Kristjan Greenewald, Igor Melnyk, Nam Nguyen, Brian Kingsbury, and Yury Polyanskiy. Estimating information flow in deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[25] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[26] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[27] R. Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[28] H. Guo, Y. Mao, and R. Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[29] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, 1989.

[30] 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.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.

[32] R. M. Hecht, E. Noor, and N. Tishby. Speaker recognition by gaussian information bottleneck. In *Tenth Annual Conference of the International Speech Communication Association*, 2009.

[33] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[34] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[36] C. Huang and S. Narayanan. Flow of renyi information in deep neural networks. In *2016 IEEE 26th International workshop on Machine Learning for Signal Processing (MLSP)*, 2016.

[37] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, 2016.

[38] P. J. Huber. *Robust statistics*. New York: John Wiley, 1981.

[39] D. J. Im, S. Ahn, R. Memisevic, and Y. Bengio. Denoising criterion for variational auto-encoding framework. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[40] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[41] K. Janocha and W. M. Czarnecki. On loss functions for deep neural networks in classification. In *Theoretical Foundations of Machine Learning (TFML)*, 2017.

[42] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, 2009.

[43] P. Khadivi, R. Tandon, and N. Ramakrishnan. Flow of information in feed-forward denoising neural networks. In *2018 IEEE 17th International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC)*, 2018.

[44] Y. Kim. Convolutional neural networks for sentence classification. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[45] Y. Kim. https://github.com/yoonkim/cnn_sentence, 2014.

[46] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[47] A. Kolchinsky, B. D. Tracey, and S. Van Kuyk. Caveats for information bottleneck in deterministic scenarios. In *International Conference on Learning Representations (ICLR)*, 2019.

[48] A. Kolchinsky, B. D. Tracey, and D. H. Wolpert. Nonlinear information bottleneck. *Entropy*, 21(12):1181, 2019.

[49] D. Kotzias, M. Denil, N. De Freitas, and P. Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[51] S. Lazebnik and M. Raginsky. Supervised learning of quantizer codebooks by information loss minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1294–1309, 2008.

[52] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[54] B. Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[55] O. L. Mangasarian and D. R. Musicant. Robust linear and support vector regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):950–955, 2000.

[56] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[57] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[58] A. Orlitsky and J. R. Roche. Coding for computing. *IEEE Transactions on Information Theory*, 47(3):903–917, 2001.

[59] B. Pang and L. Lee. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 2004.

[60] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on Association for Computational Linguistics*, 2005.

[61] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1–2):1–135, 2008.

[62] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[63] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[64] A. Navot R. Gilad-Bachrach and N. Tishby. An information theoretic tradeoff between complexity and accuracy. In *Conference on Learning Theory (COLT)*, 2003.

[65] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

[66] O. Shamir, S. Sabato, and N. Tishby. Learning and generalization with the information bottleneck. *Theoretical Computer Science*, 411(29-30):2696–2711, 2010.

[67] C. E. Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE National Convention Record*, 4(142-163):1, 1959.

[68] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[69] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[70] P. Y. Simard, Y. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural networks: tricks of the trade*. 1998.

[71] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[72] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

[73] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *International Conference on Machine Learning (ICML)*, 2015.

[74] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of 37th Annual Allerton Conference on Communication, Control and Computing*, 1999.

[75] Y. Tokozume, Y. Ushiku, and T. Harada. Between-class learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[76] Y. Tokozume, Y. Ushiku, and T. Harada. Learning from between-class examples for deep sound recognition. In *International Conference on Learning Representations (ICLR)*, 2018.

[77] S. Van Kuyk, W. B. Kleijn, and R. C. Hendriks. On the information rate of speech communication. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[78] S. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the Association for Computational Linguistics*, 2012.

[79] J. Wiebe, T. Wilson, and C. Cardie. Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation*, 39(2-3):165–210, 2005.

[80] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *Tenth IEEE International Conference on Computer Vision (ICCV)*, 2005.

[81] S. Yaman, J. Pelecanos, and R. Sarikaya. Bottleneck features for speaker recognition. In *Odyssey 2012-The Speaker and Language Recognition Workshop*, 2012.

[82] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

[83] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014.

[84] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

[85] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018.