January 2018

Deep Learning Methods For Visual Object Recognition

Zeyad Hailat
Wayne State University, zmhailat@gmail.com

DEEP LEARNING METHODS FOR VISUAL OBJECT RECOGNITION

by

ZEYAD HAILAT

DISSERTATION

Submitted to the Graduate School,

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2018

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

_____

DEDICATION

To my parents

Majed Hailat & Amal Alshouha.

To my wife

Hebah & to my boys, Karam & Laith.

To my brothers & sisters.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Allah (God) for providing me the ability and persistence to complete this dissertation and to Whom I owe everything good in my life.

Moreover, I thank my advisor Dr. Xuewen Chen for his continuous insightful advice, patient supervision, generous encouragement, and fruitful feedback throughout the journey of my Ph.D. at Wayne State University. This work would not be possible without his guidance, encouragement, and support.

Furthermore, I would like to extend special thanks to all of my dissertation committee members Dr. Jing Hua, Dr. Brahim Medjahed, and Dr. Zichun Zhong for their valuable and constructive feedback.

I am also grateful to my colleagues and friends Dr. Thair Judeh, Dr. Melih Aslan, Dr. Tarik Alafif, Dr. Tin Nguyen, and Artem Komarichev, for always being there whenever needed.

Finally, I would like to dedicate special thanks my lovely family, my loving parents, my sisters, and my brothers for all of the support, encouragement, and patience that they gave me throughout my Ph.D. years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Deep learning methods show great performance in tackling various machine learning problems [2, 4]. Supervised neural networks (e.g., CNNs) achieve state-of-the-art performance on machine learning tasks in the presence of sufficiently large training examples.

In the past, conventional CNNs would learn a fairly small number parameters, which includes a few convolutional layers (depth) and a few number of feature maps (width) due to the need for massive computational power and memory resources. Training a CNN model is also costly and may take several weeks. **Shallow** and **thin** CNN models such as AlexNet [41] are among the state-of-the-art methods in the past few years. Given the increase in computational resources via GPUs and the increase in memory resources, deeper and wider CNN architectures with millions of parameters in their models are now feasible. Deep models with hundreds and thousands of convolutional layers such as VGG [68], Inception [75], and even deeper such as Resnets [27, 28]. Moreover, wider models with hundreds and thousands of feature maps per convolutional layer are now available as well [79].

We introduce the main concepts and nomenclatures that we use throughout this thesis in chapter 2. We introduce a background for both supervised and unsupervised learning methods. Then, we explain residual network basics. After that, we explain various neural network components and functions. Finally, we close the chapter with data preprocessing methods.

In chapter 3, we discuss a new supervised learning that we proposed. We call it Hybrid Residual Networks Method (HyResNet) . HyResNet combines the features of both supervised (CNN) and unsupervised (Conv-AE) residual networks into a single super-

vised learning method.

Convolutional neural networks (CNNs) attain state-of-the-art performance on various classification tasks assuming a sufficiently large number of labeled training examples. Unfortunately, curating a sufficiently large labeled training dataset requires human involvement, which is expensive and time consuming. Semi-supervised methods can alleviate this problem by utilizing a limited number of labeled data in conjunction with sufficiently large unlabeled data to construct a classification model. Self-training techniques are among the earliest semi-supervised methods proposed to enhance learning by utilizing unlabeled data. In chapter 4, we propose a deep semi-supervised learning (DSSL) self-training method that utilizes the strengths of both supervised and unsupervised learning within a single model. We measure the efficacy of the proposed method on benchmark semi-supervised visual object classification tasks.

Deep learning methods are at the front of state-of-the-art leading methods on various supervised, unsupervised, and semi-supervised tasks in a diverse range of domains and applications. Specifically, supervised deep learning class of methods attains topmost performance assuming a sufficiently large number of noise-free labeled training examples. Unfortunately, labeled data is artificially curated and cannot be found noise-free in nature. It requires manual curation, which is expensive, time-consuming, and the labels are subject to noise. Semi-supervised methods can mitigate these obstacles by utilizing the noisy label data to construct a classification model. In chapter 5, we propose a teacher/student deep semi-supervised learning (TS-DSSL) self-training method that exploits the noise tolerance of supervised deep learning methods. We measure the efficiency of TS-DSSL on benchmark semi-supervised visual object classification tasks using benchmark datasets.

TS-DSSL sets a new state-of-the-art record on the aforementioned datasets with various levels of noisy labels. The experiments show that TS-DSSL transcends semi-supervised state-of-the-art methods for most of the aforementioned datasets.

Finally, we conclude in chapter 6.

CHAPTER 2 NOMENCLATURE AND BACKGROUND

This chapter explains the basics and terminology of neural networks from a high-level perspective. More details maybe found in [22, 66, 7]. This chapter explores neural network methods in both supervised and unsupervised fashions including their layers and parameters.

2.1 Convolution Neural Network (CNN)

CNN is a supervised type of feed-forward artificial neural network. The architecture of a CNN model comprises a sequence of convolutional layers (e.g., $Conv(5\times5)$) followed by linear layers, and ends with a supervised loss function. CNNs are at the lead in tackling various machine learning problems in various domains and applications such as visual object recognition.

2.2 Autotoencoder (AE)

Autoencoder is an unsupervised artificial neural network method that aims to learn an efficient encoding [6]. A standard AE architecture includes an encoding component followed by a decoding component. Each encoding / decoding component is a stack of at least one layer. The last component in the decoding component is an unsupervised loss function, such as a mean squared error (MSE.)

Generally speaking, an AE receives an input data (e.g., image) into the encoder. The encoder maps this input to a code. After that, the code is directed to the decoder to map back to its the original form. Finally, an unsupervised loss function compares the original input with the output of the decoder to measure the quality of the encoding component. Then, it back-propagates the error of the loss function to update the parameters od the autoencoder model .

An AE can be constructed from a stack of convolutional and deconvolutional layers, which is called a convolutional autoencoder (Conv-AE). The encoding component of a Conv-AE is a sequence of convolutional layers. Then, it is followed with a sequence of deconvolutional layers that reconstruct or decode the encoded data.

2.3 Neural Network Model Layers and Components

Ideally, the architecture of a neural network model can be constructed from a combination of various layers. For example, a standard convolutional neural network model comprises a sequence of convolutional layers followed by another sequence of fully connected layers and finally a supervised loss function. The architecture may include other in-between layers to prevent overfitting, expedite the training process, and improve the overall performance.

The complexity of a neural network model can differ from one to another by a variety of factors. The *model depth* represents the number of sequential components that constructs the model. It is mainly measured by the number of consecutive convolutional layers, i.e., more convolutional layers equals deeper model. Furthermore, the *model width* is identified by the average number of feature maps per layers, i.e, more feature maps per layer equals a wider model.

From the literature, a*variety of layers and components* can be included in a model to enhance performance, expedite convergence, and to avoid degradation and overfitting. Some examples include the dropout and batch normalization layers. The *order of these layers and components* is very important. For example, two models with the same layers but different orders may have widely different performance characteristics.

Additionally, training a neural network model includes deciding a set of *hyper-parameters*.These

need to be carefully selected given their impact on the overall performance of the model. Some examples include the learning rate, the maximum number of epochs, the kernel size, and the batch size.

We conclude this subsection with a discussion on different components and layers used to improve the performance of CNN models.

Convolution Layer. The convolutional Layer is the primary component in typical neural network models. Selecting the right set of parameters for each convolutional layer in a neural network model is crucial. The convolutional layer parameters such as the kernel size and the number of feature maps (FMs). Throughout this dissertation, we denote a component with n convolutional layers and a kernel size of $(A \times B)$ for each layer, as $[Conv(A \times B)] \times n$

Deconvolutional Layer. The deconvolutional layer reverses the operations of the convolutional layer. The deconvolutional layer is usually employed in the decoding part of the convolutional autoencoder. A component with n deconvolutional layers and a kernel size of $(A \times B)$ for each layer, is denoted as $[Deconv(A \times B)] \times n$

Dropout layer. Hinton et al. [30] introduced *dropout* to the fields of neural networks. The dropout layer integrates randomness in the learning process. Dropout sets a random ratio of the activations in a layer to zeros and keeps the values of the rest.

Batch Normalization. During training, the parameters of each layer in a neural network model changes every iteration, which changes the inputs of the next layer. Unfortunately, this change slows down the training by requiring lower learning rates and careful parameter initialization. Ioffe et al. [36] proposed a Batch Normalization (BN) method. BN performs a normalization for each minibatch in the training dataset. Ioffe et al. [36]

proved that the use of batch normalization after each convolutional layer improved the performance of neural network. Employing BN on each layer input produced the same accuracy with fewer training steps.

Subsampling Layers. It is common in neural network methods to apply a sub-sampling after a few convolutional layers followed by reducing the size of each feature map as well as increasing in the number of features maps. The sub-sampling methods include max pooling, min pooling, and average pooling.

2.4 Residual Networks

A residual network is a convolutional neural network composed of special types of modules called residual blocks [27]. In this section, we explore the various components in residual networks.

2.4.1 Residual Block (RB)

Definition 2.1 (Residual Blocks)**.** *A residual neural network is composed of a sequence of residual blocks [27]. A residual block is constructed from two parallel branches or connections (Fig. 2.1). We denote the top branch as residual branch and the bottom branch as shortcut. In general, we represnt a residual block by:*

$$z_r = \mathcal{F}(\mathrm{W}_{\mathrm{shortcut}}, z_{r-1}) + \mathcal{G}(\mathrm{W}_{\mathrm{resid\_branch}}, z_{r-1}). \tag{2.1}$$

where z_r *is the output of a residual block* r *that is used as an input to the next component,* z_{r-1} *is the input to the residual block* r*,* $\mathcal{F}$ *is the shortcut branch function,* $\mathrm{W}_{\mathrm{shortcut}}$ *are the shortcut branch parameters (weights),* $\mathcal{G}$ *is the residual branch function, and* $\mathrm{W}_{\mathrm{resid\_branch}}$

are the residual branch parameters (weights).

The residual branch in DRB (Fig. 2.1) includes two deconvolutional layers with kernels 4×4 and 3×3, respectively. Each deconvolutional layer is preceded with a batch normalization and a ReLU. The shortcut branch includes a single deconvolutional layer with a 2×2 kernel and a stride of 2.

We differentiate between two types of residual blocks based on the use of convolutional or deconvolutional layers. The **Convolutional Residual Blocks (CRB)** uses a set of convolutional layers. Figure 2.1 shows an example of a CRB. The second type is the **Deconvolutional Residual Blocks (DRB)** that is constructed from a set of deconvolutional layers. Figure 2.2 shows an example of a DRB.

Definition 2.2 (Deonvolutional Residual Block (DRB))**.** *A deconvolutional residual block (DRB) is a residual block that uses a set of deconvolutional layers.*

Definition 2.3 (Convolutional Residual Block (CRB))**.** *A convolutional residual block (CRB) is a residual block that uses a set of convolutional layers.*

2.4.2 Residual Units (RU)

A residual unit is composed of one or more residual blocks. All residual blocks in a RU are of the same type, i.e., either CRB or DRB. Moreover, all residual blocks of one residual unit share the same number of feature maps (FMs). We identify two types of RU. First, the **Convolutional Residual Units (CRU)** comprises one or more convolutional residual blocks. The first block of a CRU usually increases the number of feature maps and reduces the data size. After that, all blocks in one unit maintains the same number of feature maps and the same data size.

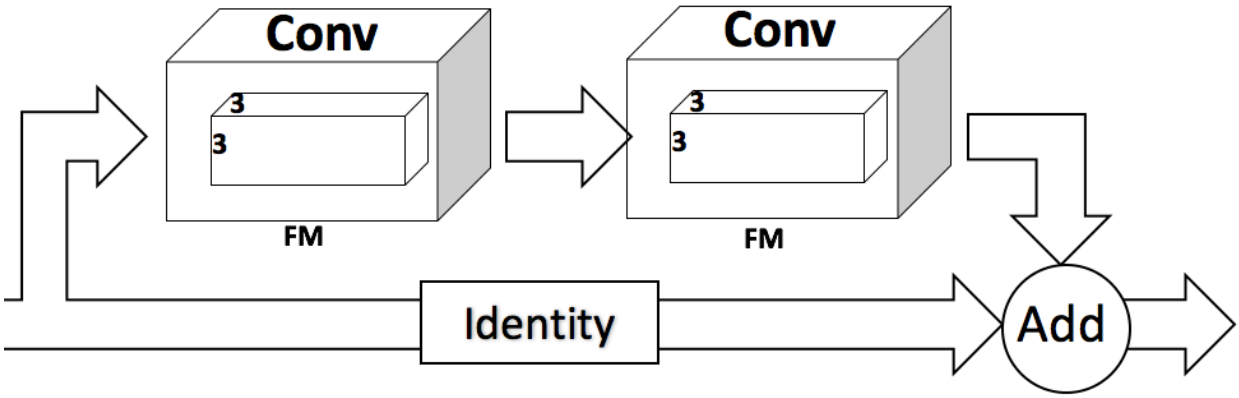


Figure 2.1: A convolutional residual block (CRB) with two branches and a number of feature maps FM. The first branch $[Conv(3 \times 3)] \times 2$ comprises a sequence of two convolutional layers. Each layer uses a kernel of size (3×3). The second branch is an identity shortcut that passes the same input signal. The last component denoted by `Add` is an element-wise addition that sums up the forwarded signals from the two branches then passes the results to the next component.

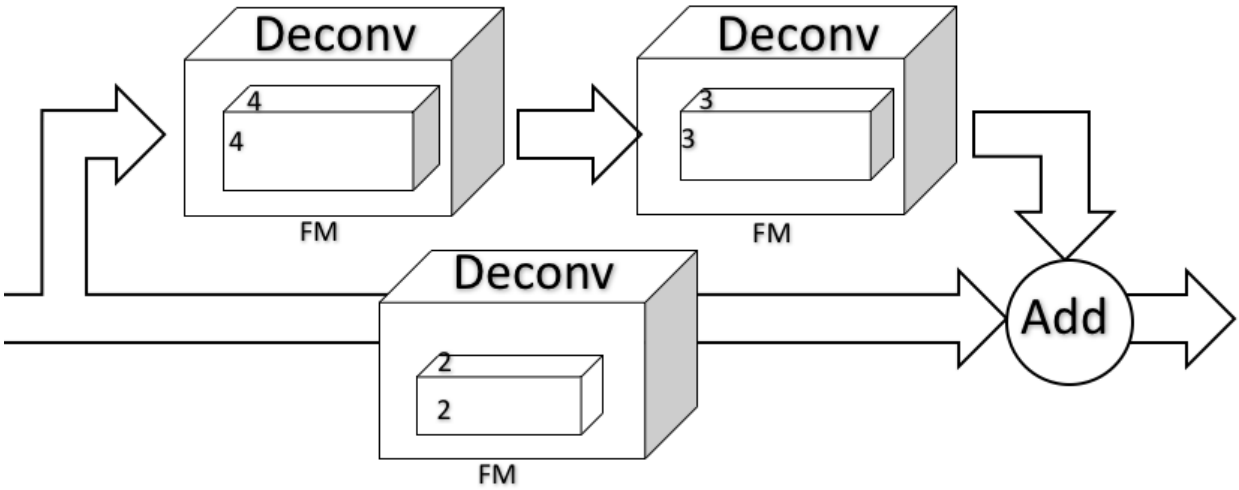


Figure 2.2: (b) A deconvolutional residual block (DRB). The top branch includes a sequence of two deconvolutional layers that are $[Deconv(4 \times 4)]$ and $[Deconv(3 \times 3)]$, respectively. The bottom layer is a deconvolutional layer $[Deconv(2 \times 2)]$. All layers use the same number of feature maps. The last component denoted by *add* is an element-wise addition that sums up the forwarded signals from the two branches then passes the results to the next component.

Definition 2.4 (Residual Unit (RU))**.** *We call a sequence of residual blocks of the same type (e.g., convolutional) and equal number of feature maps (FMs) a residual unit (RU). The RU can be either a convolutional residual unit (CRU) or a deconvolutional residual unit (DRU).*

Second, **Deconvolutional Residual Units (DRU)** includes one or more deconvolutional residual blocks with the same number of feature maps. The first block of the DRU usually reduces the number of feature maps and increases the data size. After that, all blocks in a unit maintain the same number of feature maps and the same data size.

Definition 2.5 (Convolutional Autoencoder (Conv-AE))**.** *A deep learning model that is constructed with CRBs followed by DRBs is called hereafter a convolutional autoencoder (Conv-AE).*

2.4.3 Sampling Component

We refer to the transition between two consecutive residual units as a **transition connection**. The transition connection is a convolutional layer that may change the input size and/or the number of feature maps. If the transition component is a convolutional layer that decreases the size of input (e.g., image size from $32 \Rightarrow 16$) and increases the number of feature maps (e.g., feature maps from $64 \Rightarrow 128$) then we call it *down-sampling convolutional connection*; whereas, we call it *up-sampling deconvolutional connection*. If it is a deconvolutional layer that increases the input size (e.g., image size from $16 \Rightarrow 32$) and decreases the number of feature maps (e.g., feature maps from $256 \Rightarrow 128$,) we call it an up-sampling deconvolutional connection.

2.5 Loss Functions

2.5.1 The Unsupervised Loss Function

On the top of the unsupervised component, we add an unsupervised loss function such as mean square error (MSE). The MSE measures the difference between the original input data (e.g., image) and the network output at the end of the network. The MSE loss function is calculated by the following formula:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(||h(x) - x||^2\right). \tag{2.2}$$

where N is the number of examples in the dataset (or minibatch), x is the input data (e.g., image), and $h(x)$ is the input reconstruction.

2.5.2 The Supervised Loss Function

The supervised components ends with a Cross Entropy (CE) loss function followed by a Softmax layer. The softmax and cross entropy equations are as follows:

$$\mathrm{Softmax}_k = \frac{e^{g_k(x)}}{\sum_{i=1}^{K} e^{(g_i(x))}}. \tag{2.3}$$

where $\mathrm{Softmax}_k$ is the prediction probability of class k and $g(x)$ is the output from the supervised branch.

$$J_{\mathrm{CrossEntropy}} = -\frac{1}{N}\sum_{j=1}^{N}\sum_{k=1}^{K} y_k^j \log(\mathrm{Softmax}_k^j). \tag{2.4}$$

where N is the number of examples in the dataset or the minibatch, K is the total number of classes, y is the actual label, and $\mathrm{Softmax}$ is the predicted label.

2.6 Sparsity

Sparsity sets a few activation values in a layer to zero while retaining the rest. It has been proved that sparsity learns better features. Several sparsity methods have been proposed in the literature. Olshausen et al. [57] introduced sparse coding with (L_1-norm). Ng [55] employed KL-divergence.

Makhzani et al. [49] proposed a *spatial sparsity*. In the feed-forward, the spatial sparsity sets all activation values for each feature map to zeros except the highest K values which are retained. Then, the error is back-propagated through the non-zero units to update the model parameters.

2.7 Data Preprocessing Methods

It is common practice in machine learning to perform several preprocessing normalization steps before training and testing on a dataset. Preprocessing aims to eliminate the variations of between the dataset samples and expedite the model convergence during training. Moreover, various studies such as [14, 4] show that choosing a proper sequence of preprocessing steps play a major role in enhancing the overall model performance.

2.7.1 Mean Normalization

The mean normalization is one of the most direct and simplest normalization methods. It aims to shift all dataset examples to the same mean. It is applied by calculating the mean of the training dataset ($\overline{X}_{Train}$). After that, the mean $\overline{X}_{Train}$ is used to shifting the train and test datasets by subtracting every example from that $\overline{X}_{Train}$ (Eq. 2.6).

$$\overline{X}_{Train} = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} X_{i,j}. \tag{2.5}$$

where: $\overline{X}_{Train}$ is the mean of training dataset; N is number of samples in the training dataset (e.g., the number of images); M is the number of data points in the sample i (e.g., the number of pixels in each image); $X_{i,j}$ is the value in the position j (e.g., pixel in image) within the example i.

This normalization can be employed on both colorful and gray-scale image datasets. For the latter, we calculate the mean for the overall images and then use the mean in normalization. In the former, it is common to calculate and use the mean for each channel separately (Eq. 2.6) instead of the overall mean for all channels. Therefore, we repeatedly use Eq. 2.6 for each channel and replace the $Channel$ with R, G and B, respectively. After that, we use the mean of each channel to normalize all samples in that channel.

$$\overline{X}_{\text{Train}}^{\text{Channel}} = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} X_{i,j}^{\text{Channel}}. \tag{2.6}$$

Then normalized images will be calculated as:

$$\hat{X}_e = [(X_{e,p}^{(R)} - \overline{X}_{Train}^{(R)}); (X_{e,p}^{(G)} - \overline{X}_{Train}^{(G)}); (X_{e,p}^{(B)} - \overline{X}_{Train}^{(B)})]. \tag{2.7}$$

where: $\hat{X}_e$ is the normalized examples from training or testing data and $X_{e,p}^{(R)}$ is a data point (e.g., pixel in images) p in the example e from the red (green or blue) channel.

2.7.2 Mean and Standard Deviation Normalization

In this normalization, we subtract the data points of each example from the mean of the training dataset. After that, we divide the resulted data points by the standard deviation of the training dataset. This is known as the image brightness and contrast normalization .

The mean ($\overline{X}_{Train}$) and standard deviations are calculated for each channel for colorful images. We use the calculated mean from the previous formula (Eq. 2.6.) Then we calculated the standard deviations for each channel of the RGB channels across all training dataset images.

The standard deviation for images is calculated first by finding the variance of each channels var_{Train} for the training data (Eq. 2.8).

$$var_{Train}^{Channel} = \sqrt{\frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} (X_{i,j}^{Channel} - \overline{X}_{Train}^{Channel})^2}. \tag{2.8}$$

After that, we normalize each sample using the channel mean and standard deviation (Eq. 2.9) represented in the formula below:

$$\hat{X}_e^{Channel} = \frac{X_e^{Channel} - \overline{X}_{Train}^{(Channel)}}{\sqrt{var_{Train}^{Channel}}}. \tag{2.9}$$

Finally, the normalized dataset ($X_e^{\hat{RGB}}$) is then restored into three normalized channels per samples (Eq. 2.10).

$$\hat{X}_e^{RGB} = [\hat{X_e^R}; \hat{X_e^G}; \hat{X_e^B}]. \tag{2.10}$$

2.7.3 Global Contrast and ZCA Whitening

Coates et al., [14] tested global contrast followed by ZCA whitening [35] on images datasets . They showed that this normalization leads to better performance than other types of normalizations.

CHAPTER 3 A HYBRID RESIDUAL NETWORK METHOD FOR SUPERVISED DEEP LEARNING (HYRESNET)

3.1 Introduction

The number of convolutional layers (depth) and feature maps (width) in each unit of a convolutional neural network (CNN) are two crucial performance factors. Unfortunately, evaluating a large (deep and wide) CNN model on a large-scale dataset requires a system with large amount of computational power and memory. Additionally, it may fall in various problems such as gradient degradation and overfitting. Consequently, a few years ago, a conventional CNN would be a fairly small model with a small number of parameters with both shallow (few convolutional layers) and thin (few feature maps). **Shallow** and **thin** CNN models, such as AlexNet [41], were among state-of-the-art lead methods for the past few years.

Nowadays, given the increasing availability of computational resources via GPUs, training deep and wide CNN models has become feasible. Deep models with hundreds and thousands of convolutional layers include VGG [68], Inception [75], and residual networks(ResNets) [27, 28]. Wider models with hundreds and thousands of feature maps per convolutional layer are now feasible [79].

Additionally, there has been great progress in investigated the effects of data flow within neural networks leading to methods that alleviate the gradient vanishing and degradation. He et al. [27] used a lightly connected components by passing a shortcut signal with the data from one ResNet to the next. Huang et al., [33] used a densely connected neural network to connect and pass signals across various layers.

Residual network (ResNet) is a convolutional neural network (CNN) that learns in

branches called residual block. A residual block is a simple network that comprises a combination of convolutional layers, activation layers, and batch normalization layers. Every residual block adds a small contribution to the overall network. He et al. [27] introduced a deep residual network, which was then modified [28]. The modified models are scaled to a depth of more than a thousand layers. It used a deeper architecture with various layers, options and combinations. Their model achieved the best performance when it was very deep. Their best performed model comprises 1k convolution layers, in addition to other activation layers.

Zagoruyko et. al [79] modified the model proposed in [28] to reduce its depth and increase the width. Unlike wide models, deep models are hard to parallelize, and consequently, take longer to train. Zagoruyko et. al [79] discussed and tested various wide residual network models.

While supervised learning methods use the class labels to learn and extract class-specific features, unsupervised learning methods learns general set of features across all dataset classes. The literature shows that supervised learning methods are much powerful and attain greater performance than unsupervised methods. This chapter introduces a new supervised method that we call Hybrid Residual Network Method (HyResNet) [25] for deep learning. The proposed method utilizes the power of supervised and unsupervised methods in a supervised fashion by creating high-quality representations (features).

Unlike other hybrid models, HyResNet combines both supervised and unsupervised neural networks in a single model. Other methods such as [10] employed a sequence of two or more separate methods to train and test on a single dataset. The proposed hybrid method starts with a shared set of layers before fork into two branches. One

represents a supervised learning method such as CNN with a supervised loss function while the second branch is an unsupervised learning method such as autoencoder. The unsupervised component tries to reconstruct the input after the set of deconvolutional layers without any considerations to the label. The latter branch uses the mean squared error to measure the difference between the input and the reconstructed data then uses its derivative to back-propagate the gradient. To our knowledge, this methods is the first of its kind.

3.1.1 Problem Formulation

The goal of supervised learning methods is to learn a decision model $\mathcal{M}$ from n available labeled training examples. The training examples are denoted by $\mathcal{T}_n$, where n is the total number of different training examples in $\mathcal{T}$. We denote a training example (e.g., image) by $\mathcal{X}_i \in \mathbb{R}^d$, where $i \in \{1, \ldots, n\}$. Also, we denote the true class label of $\mathcal{X}_i$ by $\mathcal{Y}_i \in \{1, \ldots, \mathcal{C}\}$, where $\mathcal{C}$ is the total number of different classes in $\mathcal{T}_n$. In general, we represent the labeled training dataset by $\mathcal{T}_n = \{(\mathcal{X}_i, \mathcal{Y}_i)\}_{i=1}^{n}$.

3.2 Method

We propose a Hybrid Residual Networks Method (HyResNet). The HyResNet utilizes supervised and unsupervised residual networks. It combines a CNN and a Conv-AE in one model. Algorithm 1 explains the training of a HyResNet. The HyResNet includes three main components as shown in Figure 3.1:

1. Convolutional component (CC): the model starts with a shared common component before branching into two components. The convolutional component is mainly a set

of convolutional residual units. Each unit is a set of CRBs with other components such as a dropout, batch normalization and ReLU.

2. Supervised component (SC): it is a sequence of fully connected (linear) layers and ends with a supervised loss function. The sequence of a convolutional component followed by the supervised component forms a convolutional neural network (CNN) model.

3. Unsupervised component(UC): it compiles a sequence of DRUs. Each DRU is a sequence of DRBs. This component aims to use the CC output and reconstruct (decode) it back to the original input data. Finally, it ends with an unsupervised loss function (e.g., MSE.) that compares the input with the constructed one. A model that is constructed with a sequence of the convolutional component followed by the unsupervised component creates a Conv-AE.

Algorithm 1 The HyResNet Algorithm

Require: D_L = labeled training data
Require: f_c, f_s and f_u = convolutional, supervised and unsupervised components, respectively
Require: $loss_s$ and $loss_u$ = supervised and unsupervised loss functions, respectively
for t in $[1, no\_epochs]$ **do**
  for each minibatch $b \in D_L$ **do**
    $w_b \leftarrow f_c(aug(b))$ ▷ *evaluates the convolutional component on augmented minibatch*
    $z_s \leftarrow f_s(w_b)$ ▷ *evaluates the supervised component on* w_b
    $loss_s \leftarrow -\frac{1}{|b|}\sum_{i \in b} \log z_{s_i}[y_i]$ ▷ *the supervised criterion,* y *is the label*
    $z_u \leftarrow f_u(w_b)$ ▷ *evaluates the unsupervised component on* w_b
    $loss_u \leftarrow \frac{1}{|b|}\sum_{i \in b} ||z_{u_i} - b_i||^2$ ▷ *the unsupervised criterion*
    updates weights using SGD with momentum
  end for
end for

We construct and test the HyResNet model in various settings and configurations. We study and compare HyResNet in the following dimensions:

1. Dropout: we study the effects of using the dropout layer at various components and blocks in HyResNet. Furthermore, we test and compare various dropout ratios.

2. Batch normalization: batch normalization showed great improvement in the neural network supervised learning performance. We study the effect of batch normalization at various components of the models.

3. We construct and test the following unsupervised component scenarios:

   (a) We refer to the deconvolutional layer that decreases the number of feature maps and increases the output size as the *up-sampling layer*. We test the use of zero, one, or two up-sampling layers on HyResNet.

   (b) We move the last convolutional layer from the convolutional unit to the supervised component and the unsupervised components. After that, we add to the unsupervised component either a set of up-sampling layers or residual deconvolutional units.

   (c) We construct the unsupervised component from a deconvolutional residual unit with various number of DRBs.

4. We study the last deconvolutional layer of the HyResNet model with various kernel sizes. We test the kernel sizes of (7×7), (11×11) and (15×15).

5. We test the convolutional layers with various kernel sizes. We evaluate the kernels of sizes of (3×3), (5×5) and (7×7).

6. We study the position of the sparsity layer in the unsupervised component.

7. We evaluate HyResNet on several pre-processing methods. We test the mean normalization, the mean and standard deviation normalization, and the global contrast normalization followed by the ZCA-whitening.

After CC HyResNet branches into UC and SC. During feed-forward, copies of the output of the CC is directed into the UC and SC. During feed-backward, the back-propagated gradients from the SC $grad_{SC}$ are element-wise added to the back-propagated gradients from the UC $grad_{UC}$:

$$grad = grad_{SC} + grad_{UC}. \tag{3.1}$$

Then, the result gradient $grad$ is back-propagated through the CC.

Table 3.1: The architectural details of the HyResNet `Models 1, 2 and 3`. L^* is the number of convolutional residual blocks in `Model 1`. We used $L = 3$ (table 3.3) and $L = 5$ (table 3.4). The last row of the table shows the SC that we used for all three models.

	Unit	**Output, FM**	**Model 1**	**Model 2**	**Model 3**
CC	`Conv1`	(32×32), 64	[Conv(5×5)]$\times 1$	[Conv(5×5)]$\times 1$	[Conv(5×5)]$\times 1$
	`CRU1`	(32×32), 64	[CRB(5×5)]$\times L^*$	[CRB(5×5)]$\times 3$	[CRB(5×5)]$\times 3$
	`CRU2`	(16×16), 128		[CRB(5×5)]$\times 3$	[CRB(5×5)]$\times 3$
	`CRU3`	(8×8), 256			[CRB(5×5)]$\times 3$
SC	[(3×3) MaxPool, stride 2] $\Rightarrow$ dropout $\Rightarrow$ FC-4096 $\Rightarrow$ ReLU $\Rightarrow$ Dropout $\Rightarrow$ FC-4096 $\Rightarrow$ ReLU $\Rightarrow$ FC-10				

We construct and evaluate the HyResNet method in various architectures. In this section, we explain the architectural details of the used methods and report results for each model variations.

3.2.1 Models 1, 2, and 3

This family of models includes the models 1, 2 and 3. The models of this family as described in table 3.1. They use the same SC and different CCs and UCs. All three models starts with a $[Conv(5 \times 5)] \times 1$ layer with 64 feature maps. The supervised component of each model comprises the sequence of $MaxPooling(3 \times 3) \Rightarrow Dropout(0.5) \Rightarrow Linear \Rightarrow ReLU \Rightarrow Dropout(0.5) \Rightarrow Linear \Rightarrow ReLU \Rightarrow Linear$ and finally the supervised loss function.

The convolutional component of `Model 1` is constructed from a single CRU with 64 feature maps. The CRU is tested with three and five CRBs. `Model 2` comes with two CRUs. The CRUs have 64 and 128 feature maps, respectively. Finally, `Model 3` constructed from three CRUs. The CRUs have 64, 128 and 256 feature maps, respectively.

3.2.2 Model 4

We adopt the convention introduced in [79] to design and construct fairly medium sized HyResNet models. We use models that use an overall of 22 convolutional layers and a widening factor of 4 (i.e., $WRN22-4$). We call this model `Model 4`. Table 3.2 shows the layers and architecture of model 4. However, this exact model $WRN22-4$ is not tested in [79]. We decided to construct and use it because it uses a fewer number of layers. This model requires less memory and runs faster with comparable results.

We first evaluate the model `WRN` $22-4$ as a baseline-supervised model (CNN). After that, we modify it to construct our HyResNet model variations by integrating the unsupervised component. Finally, we test the model with various configurations as described later

in this chapter. We test the model with two convolutional layer kernel sizes (3×3) and (5×5), which learn $4.3M$ and $11.85M$ parameters, respectively.

`Model 4` starts with a $Conv(k \times k)$, where k is the kernel dimensions of the convolutional layer. We evaluate $k = 3$ and $k = 5$. After the convolutional layer, the model includes a sequence of three CRUs. Each CRU consists of three CRBs. The CRBs are construed from two consecutive sequences of *Batch Normalization* $\Rightarrow ReLU \Rightarrow Conv(k \times k)$. The supervised component of `Model 4` comprises the sequence of $AveragePooling(8 \times 8) \Rightarrow Linear$ and finally a supervised loss function.

We report the performance of `Model 4` with kernel size (3×3) and various normalization methods on top of the CIFAR-10 dataset. Table 3.12 compares the various data preprocessing methods. Furthermore, the classification accuracy of `Model 4` with the kernel size of $Conv(5 \times 5)$ on mean-normalized CIFAR-10 achieves an accuracy of 94.85%.

3.2.3 Models 1, 2 and 3 *versus* Model 4

In this section, we compare the architectural differences between the two families of models. The first layer of all models is a convolutional layer. It includes 64 feature maps in the `Models 1, 2 and 3` whereas it includes 16 feature maps in `Model 4`.

The output of the first convolutional layer is directed to the first CRB in the `Models 1, 2 and 3`. The residual branch of the first CRB starts with a batch normalization and $ReLU$ layers whereas the first convolutional layer of `Model 4` followed by a batch normalization and $ReLU$ layers after that to the first CRB. Furthermore, the first layer of the first CRB in `Model 4` is a convolutional layer.

Additionally, the skip connection (shortcut) of the first CRB in the `Models 1, 2 and 3`

is an identity layer whereas it is a convolutional layer that increases the number of feature maps from 16 to 64 in `Model 4`.

Table 3.2: The detailed architecture of `Models 4, 5 and 6`. ‡ We test various kernel sizes. The second line in each model unit is the number of feature maps.

	Unit	Output	Model 4	Model 5	Model 6
CC	`Conv1`	32×32	[Conv($3^{\ddagger} \times 3^{\ddagger}$)]$\times 1$ 16	[Conv(3×3)]$\times 1$ 16	[Conv(3×3))]$\times 1$ 16
	`CRU1`	32×32	[CRB ($k \times k$)]$\times 3$ 64	[CRB(3×3)]$\times 6$ 160	[CRB(3×3)]$\times 4$ 160
	`CRU2`	16×16	[CRB ($k \times k$)]$\times 3$ 128	[CRB(3×3)]$\times 6$ 320	[CRB(3×3)]$\times 4$ 320
	`CRU3`	8×8	[CRB ($k \times k$)]$\times 3$ 256	[CRB(3×3)]$\times 6$ 640	[CRB(3×3)]$\times 4$ 640
SC	[AveragePooling(8×8), stride 1] $\Rightarrow$ FC-10				

3.2.4 Models 5 and 6

The `Models 5 & 6` follow the convention introduced in [79]. They are constructed with the large architectures for `Models 5` $WRN40-10$ and `Models 6` $WRN28-10$. Both models use an overall of 40 and 28 convolutional layers, receptively. Furthermore, both of the models use a widening factor of 10. Table 3.2 depicts the detailed architectural design for both models.

3.3 Experiment

We train all models in this chapter from scratch. All of our experiments follow the protocols introduced in [79]. We use a minibatch size of 128 samples. We train each experiments for 200 epochs. The learning rate starts at 0.1. Then it is decreased after the epochs 60,120 and 160 by a factor of 0.2.

We evaluate all of the proposed models on CIFAR-10 [40] dataset. CIFAR-10 consists of $60,000$ RGB images of the size of 32×32 pixels. The dataset includes 10 different

classes with the same number of examples per class. Moreover, it is available in two splits that we adopt hereafter in our experiments. These parts consist of $50,000$ and $10,000$ examples used for training and testing, respectively.

Moreover, we evaluate the best performed HyResNet models on CIFAR-100 [40]. Similar to CIFAR-10, CIFAR-100 compiles $60,000$ images. Each image dimensions are 32×32 pixels. However, CIFAR-100 includes 100 classes compared to the 10 classes found in CIFAR-10.

3.3.1 Model 1 Variations

Table 3.1 explains the architectural details of `Model 1`. We evaluate `Model 1` with three different kernel sizes. The sizes are (3×3), (5×5) and (7×7). Furthermore, we evaluate the model with 3 CRBs and 5 CRBs. The results are reported in the tables 3.3 and 3.4, respectively.

We design models `Model 1-A` and `Model 1-F` as CNN models. Furthermore, we create the HyResNet `Model 1` variations `Model 1-B`, `Model 1-C`, `Model 1-D` and `Model 1-E`. The unsupervised component of the HyResNet `Model 1` variations ends with a sparsity layer followed by a deconvolutional layer.

Each convolutional components of the models `Model 1-A`, `Model 1-B`, and `Model 1-C` is constructed with a single CRU. The CRU is constructed from a sequence of three CRBs (table 3.3). The models `Model 1-A` and `Model 1-B` do not us any dropout. `Model 1-C` employs a dropout with 0.5%.

Models `Model 1-D`, `Model 1-E`, and `Model 1-F` are constructed with a single CRU convolutional components that compiles five CRBs without a dropout. We test them on three

Table 3.3: The performance of `Model 1` with a single CRU in the convolutional components. The CRU compiles three CRBs (L=3). We evaluate this model on a ZCA-whitened version of CIFAR-10. The model `Model 1-A` is a CNN model.

Models	*Accuracy (%)*
`Model 1-A`	91.82
`Model 1-B`	92.25
`Model 1-C`	**92.69**

kernel sizes (3×3), (5×5) and (7×7). Additionally, we evaluate the models with different deconvolutional layer kernel sizes. Table 3.4 shows the results of all aforementioned configurations.

Table 3.4: `Model 1` with one CRU and five CRB (L=5). We applied the experiments on a ZCA-whitened version of CIFAR-10 and without dropout. `Model 1-F` is a CNN model.

Conv. kernel size	`Model 1-D` *[Deconv(11×11)]*	`Model 1-E`	`Model 1-F`
Conv(3×3)	92.25	**92.61** *[Deconv(7×7)]*	
Conv(5×5)	**92.55**		92.11
Conv(7×7)	92.3	**92.67** *[Deconv(15×15)]*	

3.3.2 Model 2 Variations

`Model 2` uses the convolutional component and the supervised components depicted in table 3.1. We modify and evaluate the unsupervised component of `Model 2` in various scenarios. `Model 2` is evaluated on a ZCA-whitened version of CIFAR-10. Table 3.5 shows the performance of various `Model 2` variations. All `Model 2` experiments do not employ any dropout. The unsupervised components that we use with `Model 2` follow below:

In **Model 2-A**, we add a sparsity layer to the unsupervised component. The sparsity layer is added immediately after the fork split that comes at the end of the convolutional component. Then, we use a sequence of a single up-sample deconvolutional layer fol-

lowed by batch normalization, ReLU, $Deconv(11 \times 11)$, batch normalization, and an unsupervised loss function. This model achieves an accuracy of 93.65%.

The unsupervised component of **Model 2-B** is constructed from the sequence of sparsity layer, a single up-sampling deconvolutional layer, ReLU, $Deconv(11 \times 11)$, and an unsupervised loss function. This model achieves an accuracy of 93.99%.

Model 2-C unsupervised component consisting the sequence of sparsity, a single up-sampling deconvolutional layer, ReLU, sparsity , $Deconv(11 \times 11)$, and an unsupervised loss function. The achieved accuracy is 94.04%. Note that, **Model 2-C** is different from **Model 2-B** where we add a second sparsity layer before the ReLU in **Model 2-C**.

Table 3.5: The variations of `Model 2`. All results in this table are for a ZCA-whitened CIFAR-10 and without dropout.

	Model	**Accuracy** (%)
CNN	`Model 2`	93.86
A single up-sampling deconvolutional layer	`Model 2-A`	93.65
	`Model 2-B`	93.99
	`Model 2-C`	**94.04**
	`Model 2-D`	93.78
A single DRB	`Model 2-E`	93.59
	`Model 2-F`	**94.03**
	`Model 2-G`	93.87

In **Model 2-D**, the unsupervised component is constructed from a single up-sampling deconvolutional layer, ReLU, sparsity, $Deconv(11 \times 11)$, and the unsupervised loss function. The accuracy of `Model 2-D` is 93.78%.

The unsupervised component of **Model 2-E** is constructed from sparsity layer, a DRB that includes a batch normalization, ReLU, $Deconv(11 \times 11)$, batch normalization, and an unsupervised loss function. The accuracy is 93.59%

The unsupervised component of **Model 2-F** includes a sparsity layer, a DRB compiles

the sequence of batch normalization, ReLU, $Deconv(11 \times 11)$, and finally an unsupervised loss function. The accuracy for this model is 94.03%.

The last variation is **Model 2-G**. The unsupervised component is constructed from a DRB with batch normalization, ReLU, sparsity, $Deconv(11 \times 11)$, and an unsupervised loss function. The accuracy is 93.87%.

3.3.3 Model 3 Variations

We evaluate various configurations and components of the unsupervised component in `Model 3`. Table 3.6 shows the performance of `Model 3` variations.

Table 3.6: The results of `Model 3` variations evaluated on CIFAR-10.

	Model	Accuracy (%)
CNN, no dropout	`Model 3`	94.97
HyResNet Models	`Model 3-A`	94.27
	`Model 3-B`	**95.03**
	`Model 3-C`	**95.03**
	`Model 3-D`	94.86
	`Model 3-E`	94.92

The convolutional component of **Model 3-A** includes three CRUs. Each CRU includes three CRBs, where the residual branch of each CRB is a sequence of three convolutional layers $Conv(3 \times 3)$, $Conv(5 \times 5)$ and $Conv(7 \times 7)$, respectively. We apply a 50% dropout layer between every two convolutional layers in the CRBs. The unsupervised component comprises two up-sampling layers followed by $Deconv(15 \times 15)$. This model achieves 94.27% of accuracy on ZCA-whitened CIFAR-10.

In **Model 3-B**, the unsupervised component is constructed with two DRBs. Each block includes two deconvolutional layers with kernel sizes (6×6) and (5×5), respectively. There is neither batch normalization nor ReLU after the first DRB. Furthermore, there is

not dropout. We placed the sparsity before the last deconvolutional layer and after the deconvolutional residual blocks. Moreover, the last deconvolutional layer in the unsupervised component has a kernel size of (11×11). We test this model on a mean-normalized CIFAR-10, and the classification accuracy is 95.3%

Model 3-C constructs the unsupervised component from a sequence of two up-sampling layers followed by a sparsity layer and a $Deconv(11 \times 11)$. This model employs a dropout. It achieves a classification accuracy of 95.03% on mean-normalized CIFAR-10.

In **Model 3-D**, the unsupervised component is constructed from a DRU followed by two up-sampling layers, sparsity, and a deconvolutional layer. The DRU consists of three CRBs with 256 feature maps. We do not use dropout in this model. Moreover, all convolutional and deconvolutional layers are preceded with a batch normalization. We evaluate this model on a mean-normalized CIFAR-10. It achieves a classification accuracy of 94.86%

The unsupervised component of **Model 3-E** is constructed with a sequence of two DRUs. Each DRU includes a DRB. We do not employ any batch normalization in the unsupervised component of this model, but we use a dropout. The sparsity in this model is placed before first DRUs. `Model 3-E` shows a classification accuracy of 94.92% on a mean-normalized CIFAR-10.

3.3.4 Model 4 Variations

Figure 3.7 shows an example of the (a) supervised and (b) unsupervised components in `HyResNet Model 4`.

Model 4-A is a wide residual network (WRN 22-4) designed following the convention

Table 3.7: The performance of `Model 4` variations. All of the results in this table are for mean-normalized CIFAR-10.

	Model	**Accuracy** (%)
Model 4-A CNN	No dropout	95.64
	dropout	**95.89**
Deconvolutional Residual Blocks, no dropout	`Model 4-B`	95.38
	`Model 4-C1`	**95.88**
	`Model 4-C2`	95.8
	`Model 4-C3`	95.6
	`Model 4-C4`	95.68
	`Model 4-D`	95.68
	`Model 4-E`	95.47
	`Model 4-F`	95.56
	`Model 4-G`	95.68
Deconvolutional Residual Blocks dropout	`Model 4-H`	**95.69**
	`Model 4-I`	95.63
	`Model 4-J1`	95.33
	`Model 4-J2`	95.75
	`Model 4-K`	95.51
	`Model 4-L1`	95.59
	`Model 4-L2`	95.62
Deconvolutional Up-sampling no dropout	`Model 4-N1`	**95.76**
	`Model 4-N2`	95.65
	`Model 4-N3`	95.42
	`Model 4-N4`	95.42
	`Model 4-O`	95.53

Table 3.8: The accuracy of `Model 4` 22-4 models on a mean normalized CIFAR-10. * We developed and ran these models because they were not provided in the original paper. `Model 4` 22-4 uses $4.2M$ parameters and `Model 4` 22-4 HyResNet uses $5.3M$ parameters.

	`WRN 22-4*`	`WRN 22-4 HyResNet`
No Dropout	95.64	95.88
Dropout	95.89	95.69

proposed in [79]. However, this exact model does not exist in the original work of [79]. We run `Model 4-A` on a mean-normalized CIFAR-10 with $Conv(3 \times 3)$. The model achieves accuracies of 95.89% and 95.64% with and without the use of dropout, respectively.

After that, we adopt the CC and SC components of **Model 4-A** to construct HyResNet with the settings discussed below.

Model 4-B modifies `Model 4-A` by adding the unsupervised component to formu-

late a HyResNet. The added unsupervised component constructs from a $sparsity$ and $Deconv(7 \times 7)$ layers. We evaluate this model on mean-normalized CIFAR-10, and it achieves an accuracy of 95.38%.

We evaluate **Model 4-C** on mean-normalized CIFAR-10. The unsupervised component of **Model 4-C** compiles two DRBs. Each deconvolutional layer of the DRBs is preceded with a batch normalization and a ReLU. We test the model appending different sequences of layers at the end of the UC as illustrated below:

1. `Model 4-C1`: $sparsity$, $Deconv(7 \times 7)$ achieves an accuracy 95.88%
2. `Model 4-C2`: BN, $sparsity$, $Deconv(7 \times 7)$ achieves an accuracy 95.6%
3. `Model 4-C3`: BN, $ReLU$, $sparsity$, $Deconv(7 \times 7)$ achieves an accuracy 95.68%
4. `Model 4-C4`: $ReLU$, $sparsity$, $Deconv(7 \times 7)$ achieves an accuracy 95.8%

Table 3.9: The performance of the `Model 4-C` variations on CIFAR-10. Every column is a model. The layers are top-down (top one is the first layer in the model and so on). * A minibatch of size 64 is used.

	`Model 4-C1`	`Model 4-C2`	`Model 4-C3`	`Model 4-C4`
The Unsupervised Component Layers	`sparsity` `Deconv(7 × 7)`	`BN` `sparsity` `Deconv(7 × 7)`	`BN` `ReLU` `sparsity` `Deconv(7 × 7)`	`ReLU` `sparsity` `Deconv(7 × 7)`
Accuracy (%)	**95.88%** (95.59%*)	95.6%	95.68%	95.8%

Table 3.9 summaries the performance of `Model 4`. Moreover, we experiment and report `Model 4-C1` with a minibatch sizes of 64 and 128 that achieve accuracies of 95.59% and 95.88%, respectively. We also evaluate various kernel sizes of the deconvolutional layers in the DRBs of `Model 4-C1`.

- When we use the shortcut connection as a $Deconv(1 \times 1)$, the accuracy is 95.36%.

- The last deconvolutional layer uses $Deconv(3 \times 3)$. The accuracy is 95.39%.
- The first deconvolutional layer in each DRB is $Deconv(3 \times 3)$. The accuracy is 95.4;
- The shortcut connection is $Deconv(1 \times 1)$ and the first deconvolutional layer in each DRB is $Deconv(3 \times 3)$. The accuracy is 95.63%.
- The shortcut connection is $Deconv(1 \times 1)$. the first deconvolutional layer in each DRB is $Deconv(3 \times 3)$ and the last deconvolutional layer uses $Deconv(3 \times 3)$. The accuracy is 95.39%.

Finally, we evaluate `Model 4-C1` with various kernel sizes of the last deconvolutional layer. Table 3.10 shows the classification accuracy of each kernel size.

The deconvolutional component of **Model 4-D** constructs from two DRUs. Each DRU comprises two DRB, where each deconvolutional layer of the DRBs is preceded with a batch normalization and a ReLU. Then, one of the following sequences is appended to the end of the UC and evaluated on a mean-normalized CIFAR-10.

1. The sequence of BN, $ReLU$, $sparsity$, and $Deconv(7 \times 7)$. The classification accuracy is 95.68%.
2. The sequence of $sparsity$, and $Deconv(7 \times 7)$. The classification accuracy is 95.39%.

The UC of **Model 4-E** is constructed from three DRUs units, where each DRU comprises two DRBs. Each deconvolutional layer of the DRBs is preceded with a batch normalization and a ReLU. The sequence of the layers BN, $ReLU$, $sparsity$, and $Deconv(7 \times 7)$ is appended at the end of the UC. This model achieves a classification accuracy of 95.47% on a mean-normalized CIFAR-10.

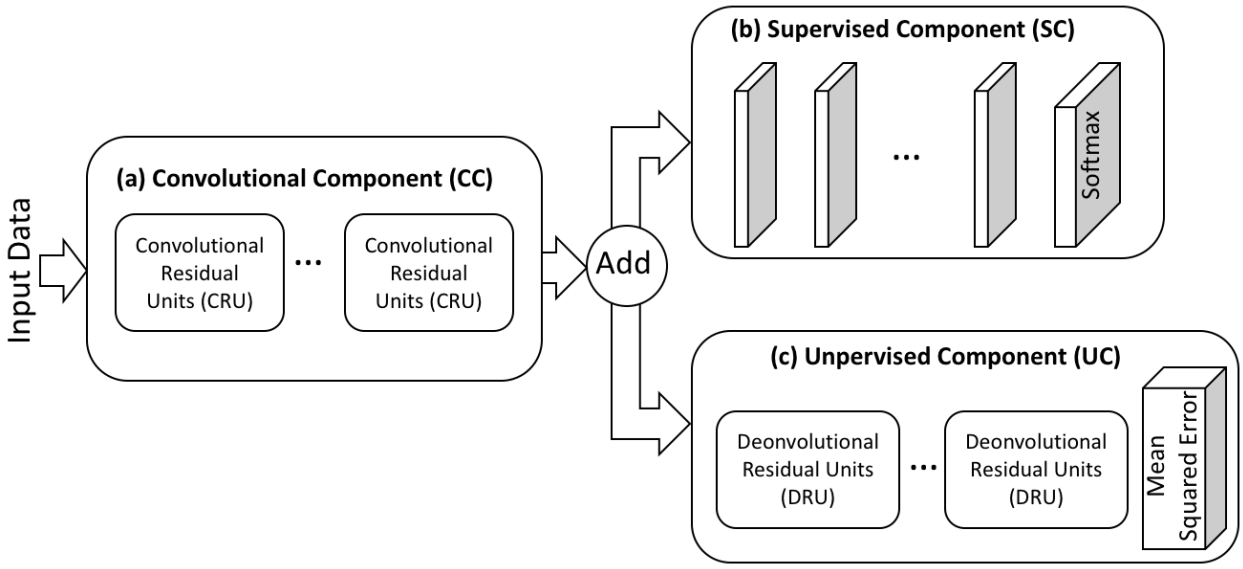


Figure 3.1: The standard structure of a Hybrid Residual Network Model (HyResNet) with the three main components: (a) Convolutional component (CC), (b) Supervised component (SC) and (c)Unsupervised component(UC)

Table 3.10: We evaluate the `Model 4-C1` with various kernel sizes for the last deconvolutional layer on mean-normalized CIFAR-10.

kernel size	Accuracy (%)
`Deconv`(3×3)	95.39
`Deconv`(5×5)	95.56
`Deconv`(7×7)	**95.88**
`Deconv`(9×9)	95.52
`Deconv`(11×11)	95.61

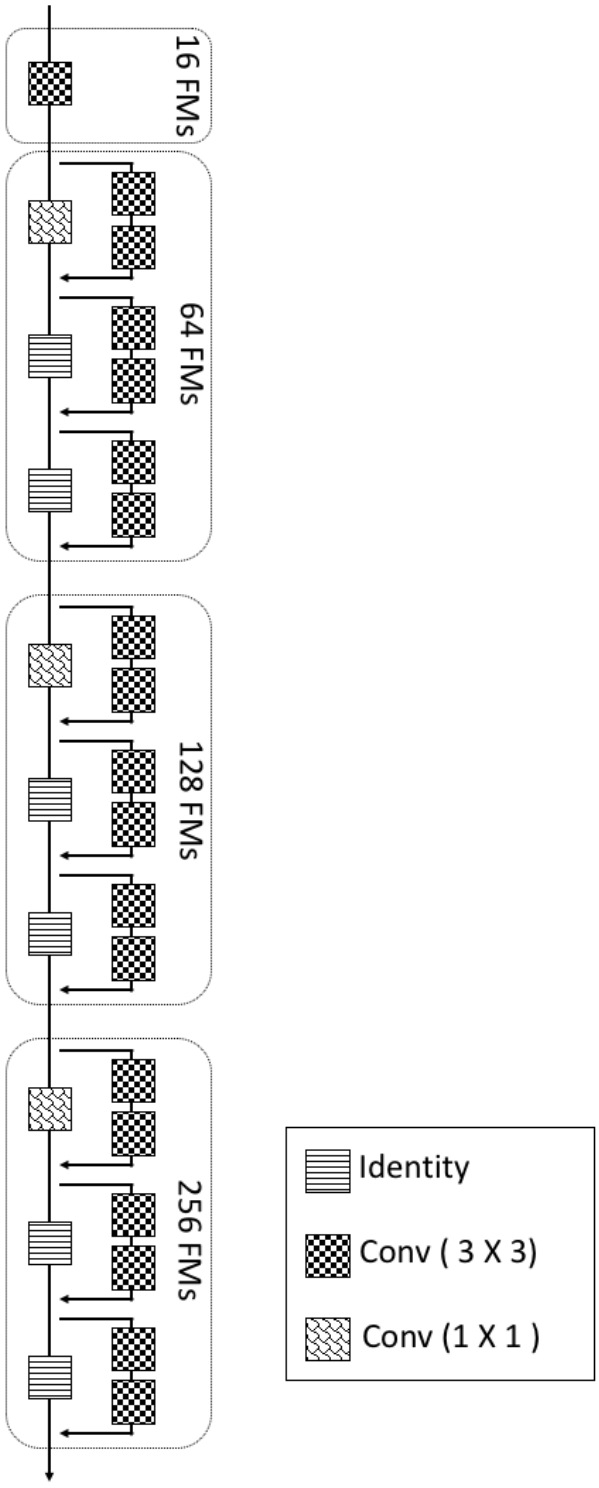


Figure 3.2: The convolutional component of `Model 4`. (HyResNet 22-4)

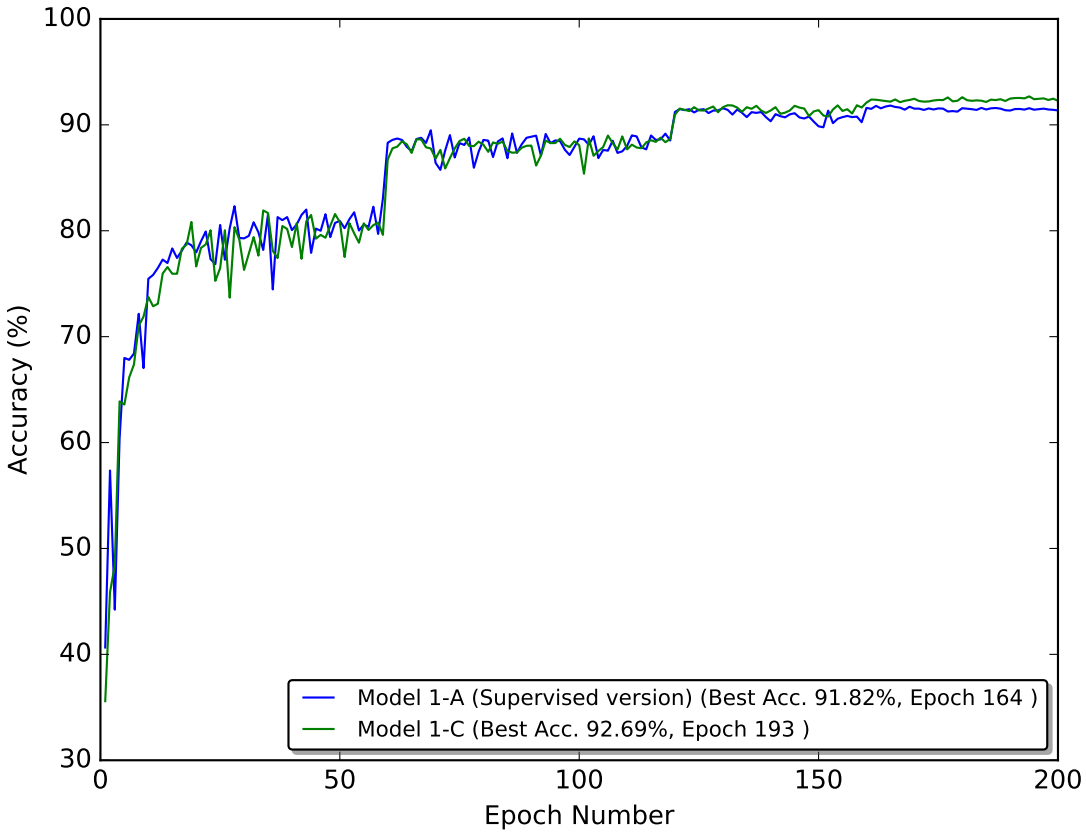


Figure 3.3: A comparison of CNN model `Model 1-A` and the HyResNet `Model 1-C`. The figure shows the performance of the models with a kernel size of (3×3) across all epochs on CIFAR-10.

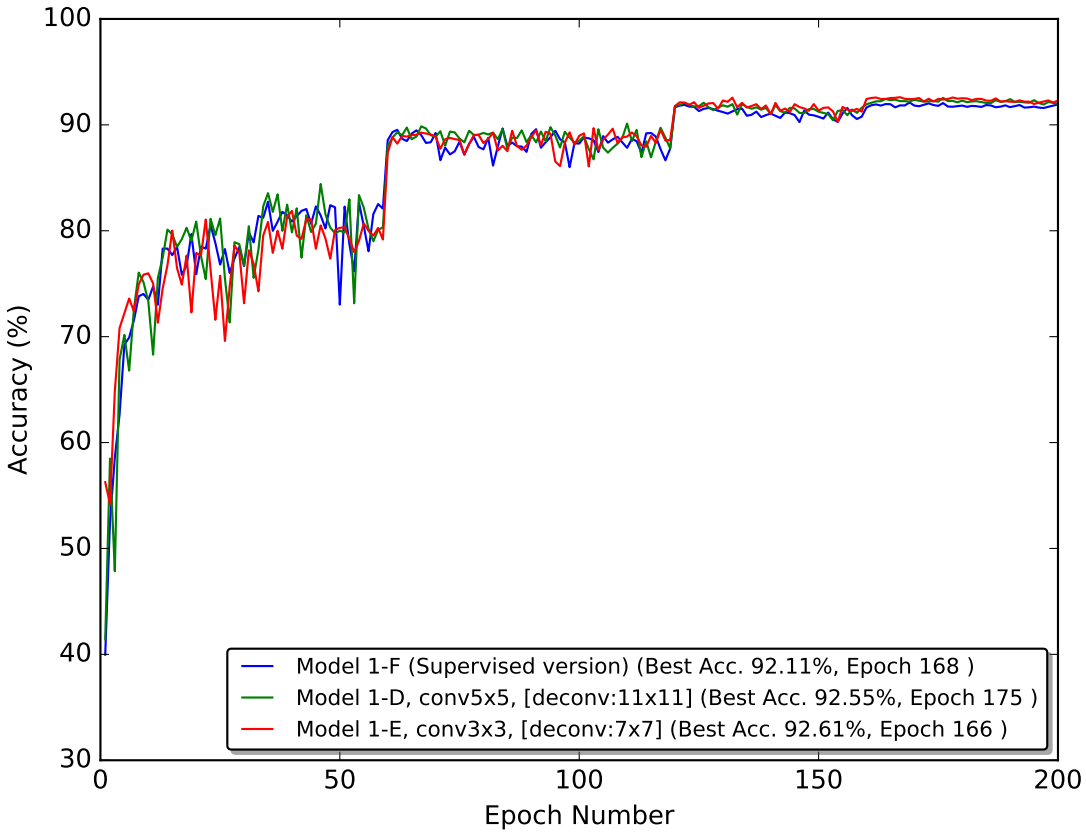


Figure 3.4: A comparison of `Model 1` variations. `Model 1-F` is a CNN model with a kernel size of (5×5). The figure shows the accuracies of each model across all epochs on CIFAR-10.

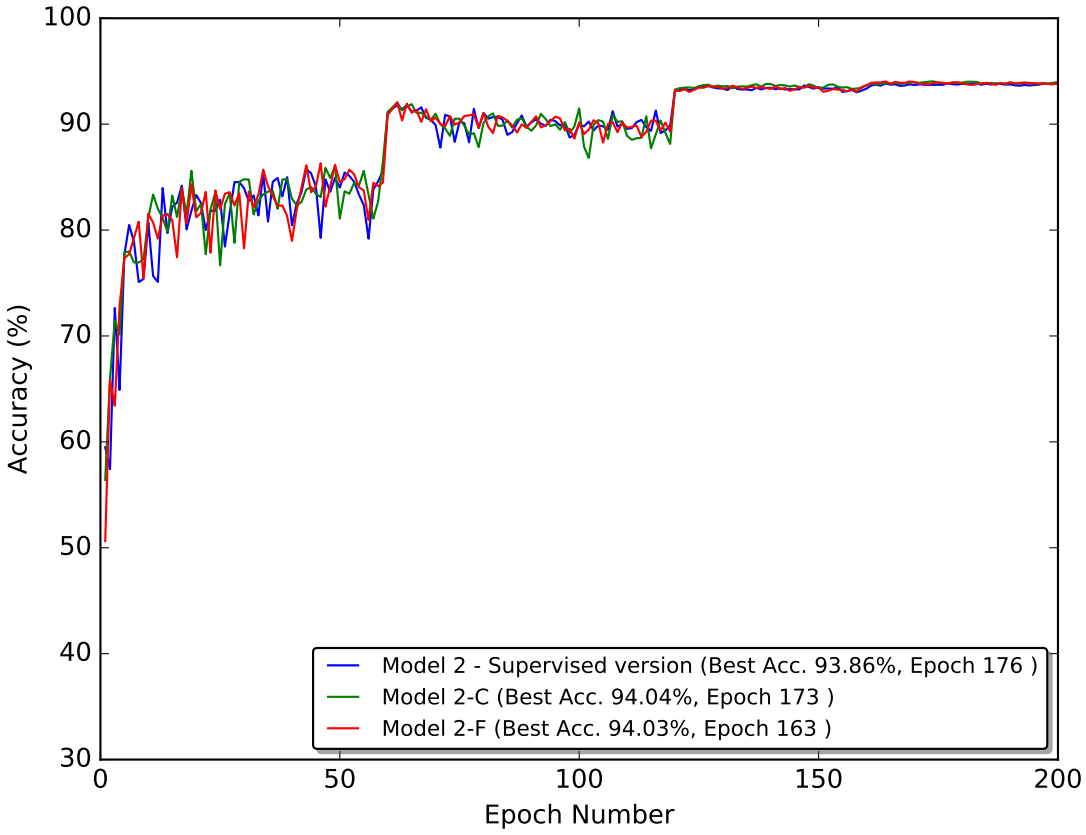


Figure 3.5: A comparison of the best HyResNet `Model 2` configurations on CIFAR-10.

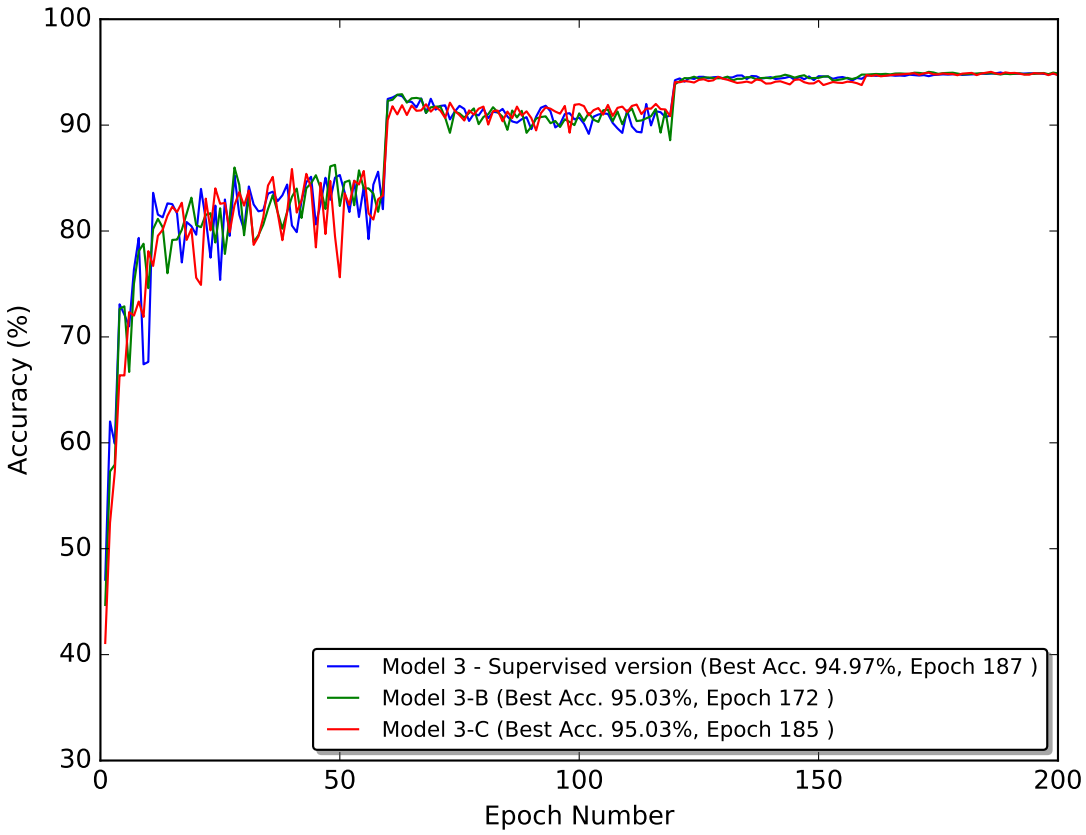


Figure 3.6: A comparison of the best performing variations of `Model 3`. The results are on CIFAR-10.

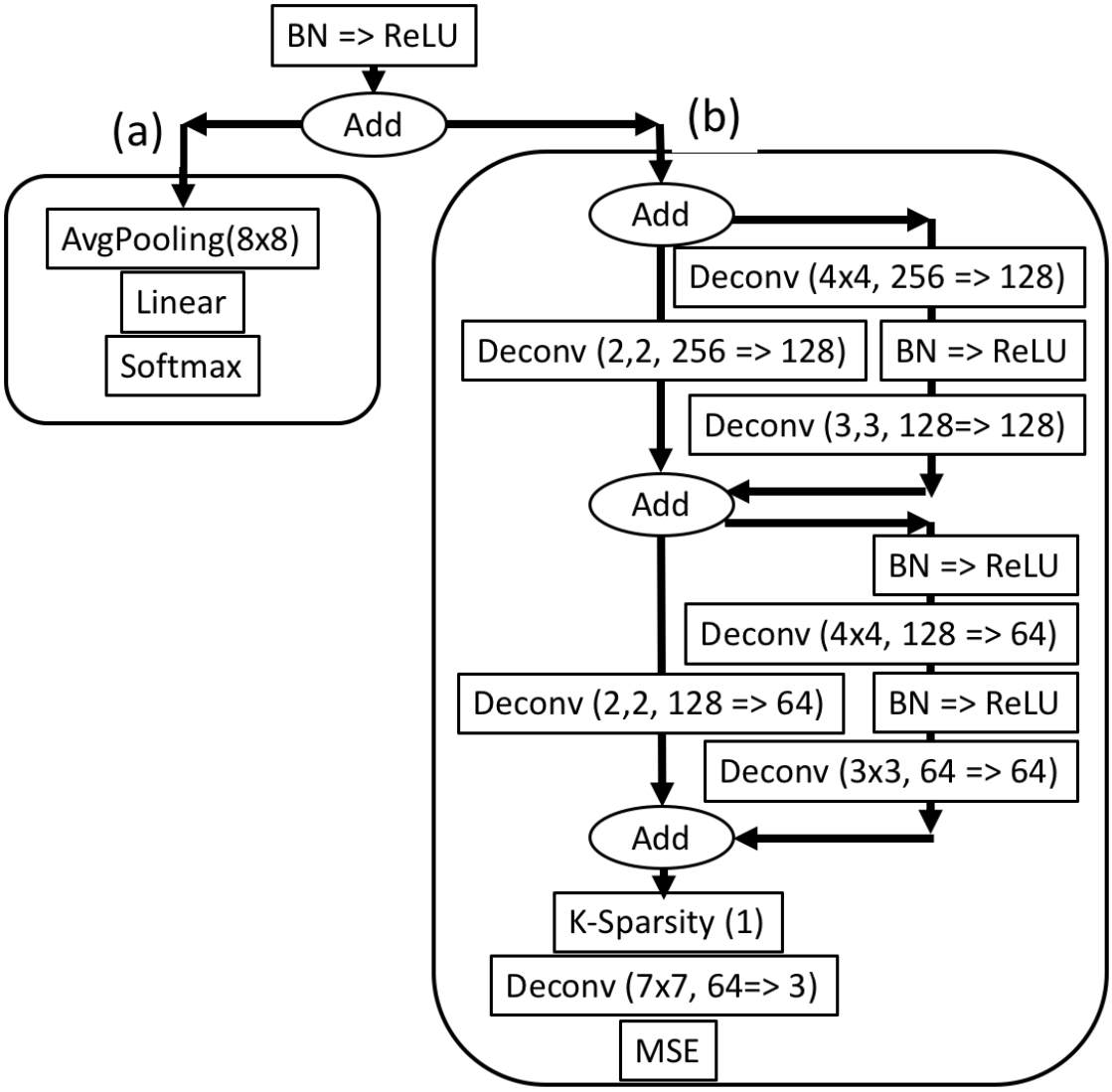


Figure 3.7: `HyResNet Model 4` architectural design. The figure shows the (a) supervised component with the layers of $MaxPooling \Rightarrow Linear \Rightarrow Softmax$ and (b) unsupervised component with two DRUs with a DRB each. The DRBs are with 128 and 64 feature maps, respectively.

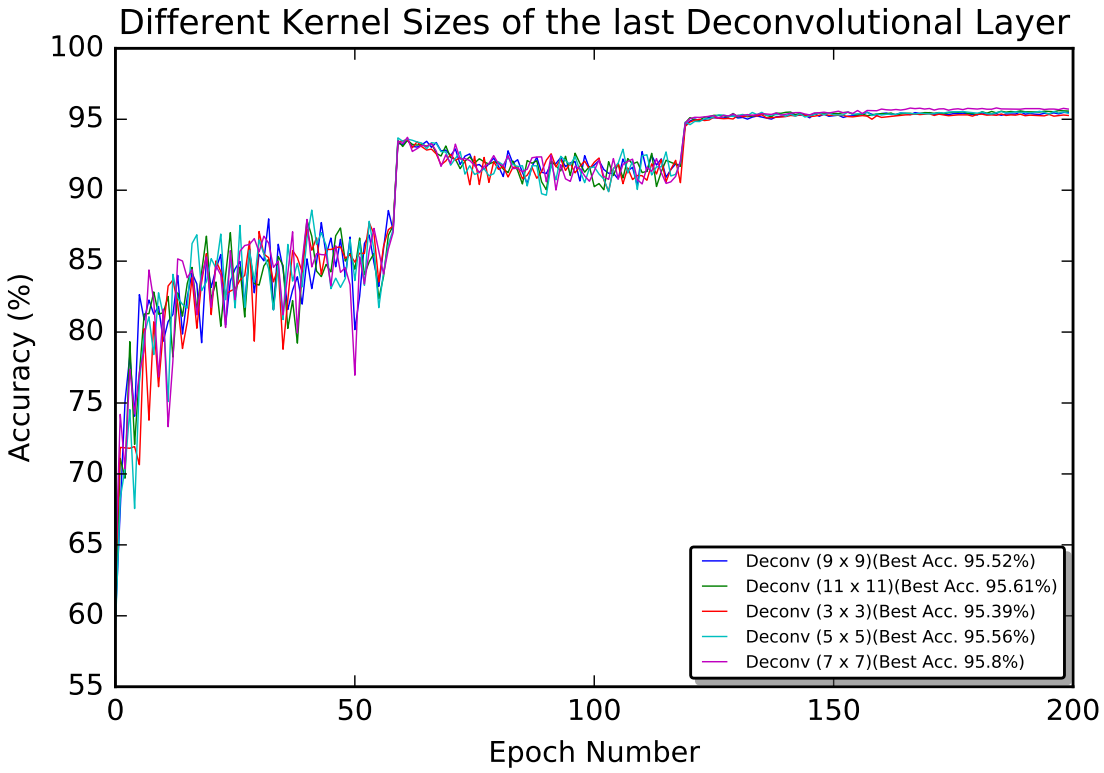


Figure 3.8: Compares the various settings of `Model 4-C1` with various kernel sizes of the last deconvolutional layer for a mean-normalized CIFAR-10.

In **Model 4-F**, the unsupervised component uses two DRBs. Each deconvolutional layer of the DRBs is preceded with batch normalization and ReLU. Then, we move the layers BN and $ReLU$ from the beginning of the second DRB and place them before the beginning of the block and after the CC fork. Therefore, the input for both the shortcut and the connection is normalized then passed through a $ReLU$. After that, we append the layers $sparsity$ and $Deconv(7 \times 7)$ to the end of the UC. We evaluate this model on mean-normalized CIFAR-10, and the classification accuracy is 95.56%.

3.3.5 The Effects of Dropout

Dropout in Convolutional Component

We use **Model 4-H** to investigate the effects of various dropout ratios. We evaluate this model twice with the dropout ratios of 0.3, and 0.5. This dropout applies to the convolutional components only, and we do not use any dropouts in the unsupervised component.

The unsupervised component of this model uses two DRBs. Each deconvolutional layer of the DRBs is preceded with a batch normalization and a ReLU. Then we append the layers $sparsity$ and $Deconv(7 \times 7)$ to the end of the UC. The accuracies of 0.3 and 0.5 dropout on a mean-normalized CIFAR-10 are 95.69%s 94.71%, respectively.

Dropout in both Convolutional and Unsupervised Components

In **Model 4-I**, we apply a dropout ratio of 0.3 to both the convolutional and unsupervised components. In this model, the unsupervised component uses two DRBs. Each deconvolutional layer of the DRBs is preceded with batch normalization and ReLU. We then append $sparsity$ and $Deconv(7 \times 7)$ to the end of UC. The evaluation of this model

on a mean-normalized CIFAR-10 achieves a classification accuracy of 95.63%.

Dropout or Sparsity in the Unsupervised Component

Model 4-J1 is similar to `Model 4-I` except that there is no dropout in the convolutional component. This method achieves an accuracy of 95.33%

Model 4-J2 is the same as `Model 4-J1` but we replaced the dropout in the unsupervised component with sparsity layers. The method achieves a classification accuracy of 95.75%.

Model 4-J3 is the same as `Model 4-J1` except that we replace the batch normalization layers in the DRBs with dropout layers. The accuracy is 95.58%

The unsupervised component of **Model 4-K** uses three DRBs, where the first block maintains the same input number of features maps and input signal sizes. Each deconvolutional layer of the DRBs is preceded with a batch normalization and a ReLU. The layers $sparsity$ and $Deconv(7 \times 7)$ are appended at the end of the UC. The accuracy of the model with a dropout is 95.51%.

3.3.6 The Effects of Batch Normalization

The unsupervised component of **Model 4-L** uses two DRBs. We removed the batch normalization from the deconvolutional component. Each deconvolutional layer of the DRBs is preceded with ReLU. After that, we evaluate one of the following scenarios by appending a sequence of layers to the end of the UC.

1. The sequence of layers $ReLU$, $sparsity$, and $Deconv(7 \times 7)$ achieves an accuracy of 95.59%.

2. The sequence of layers $sparsity$ and $Deconv(7 \times 7)$ achieves an accuracy of 95.62%.

3.3.7 The Unsupervised Component: Deconvolutional Layers

In this family of models we use the unsupervised component as a sequence of deconvolutional layers. The deconvolutional layers aim to reconstruct the original input. Figure 3.9 shows an example of (a) a supervised component of a `HyResNet Model 4` and (b) an unsupervised component composed of three deconvolutional layers. Table 3.11 shows the variations of the unsupervised component of **Model 4-N**, which were evaluated on mean-normalized CIFAR-10. The four scenarios are summarized below.

- `Model 4-N1`: the sequence $Deconv(2 \times 2)$, BN, $ReLU$, $Deconv(2 \times 2)$, BN, $ReLU$, $sparsity$, and $Deconv(7 \times 7)$ achieves an accuracy of **95.76**$\%$.

- `Model 4-N2`: the sequence $Deconv(2 \times 2)$, BN, $ReLU$, $Deconv(2 \times 2)$, BN, $sparsity$, and $Deconv(7 \times 7)$ achieves an accuracy of 95.65%.

- `Model 4-N3`: the sequence $Deconv(2 \times 2)$, $ReLU$, $Deconv(2 \times 2)$, $ReLU$, $sparsity$, and $Deconv(7 \times 7)$ achieves an accuracy of 95.42%.

- `Model 4-N4`: the sequence $Deconv(2 \times 2)$, BN, $ReLU$, $Deconv(2 \times 2)$, $sparsity$, and $Deconv(7 \times 7)$ achieves an accuracy of 95.42%.

Model 4-O is similar to `Model 4-N1` except that we apply a dropout of 0.3 in the convolutional component. This model achieves an accuracy of 95.53%.

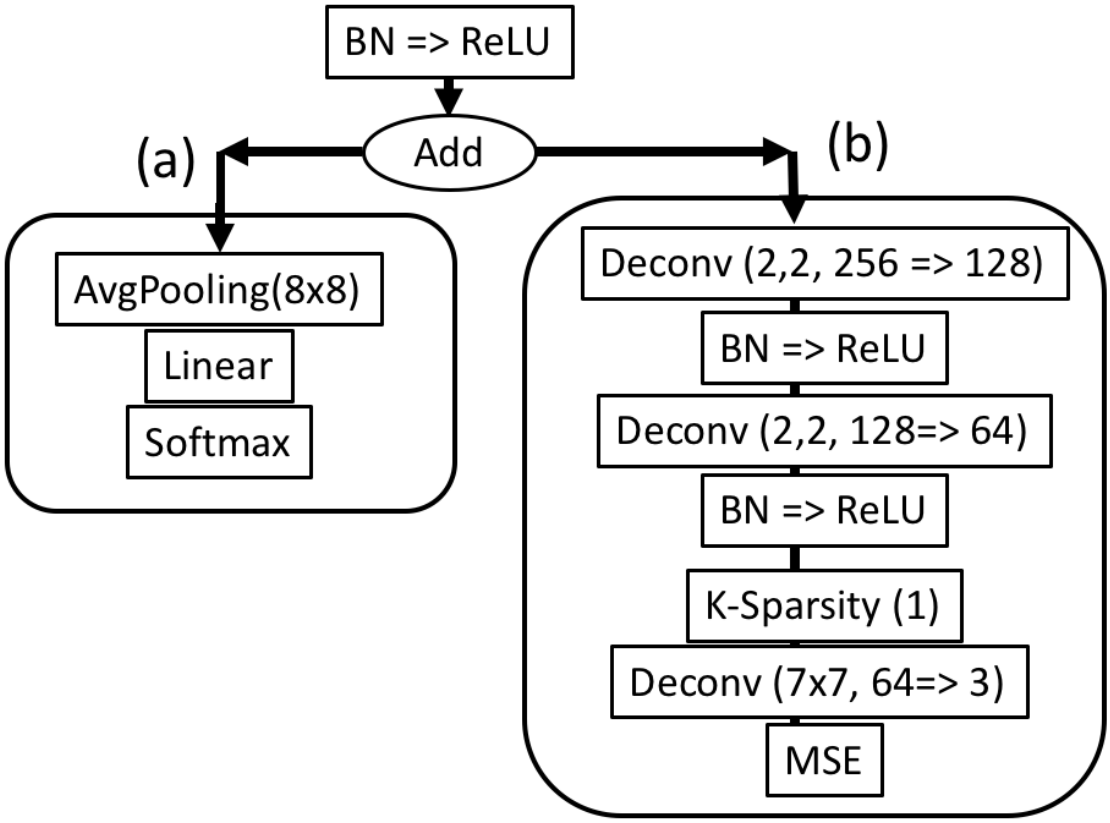


Figure 3.9: The architectural details of the `HyResNet Model 4`. The figure shows that (a) the supervised component is constructed with the sequence of $MaxPooling \Rightarrow Linear \Rightarrow Softmax$ and (b) the unsupervised component with two up-sampling, 128 and 64 feature maps, respectively.

Table 3.11: `Model 4-N` variations. Every column is a model. The layers are top-down (top one is the first layer in the model and so on).

	`Model 4-N1`	`Model 4-N2`	`Model 4-N3`	`Model 4-N4`

The Unsupervised Component Layers	`Deconv`(2×2)
	`BN`
	`ReLU`
	`Deconv`(2×2)
	`BN`
	`ReLU`
	`sparsity`
	`Deconv`(7×7)
		`Deconv`(2×2)
		`BN`
		`ReLU`
		`Deconv`(2×2)
		`BN`
		`sparsity`
		`Deconv`(7×7)
			`Deconv`(2×2)
			`ReLU`
			`Deconv`(2×2)
			`ReLU`
			`sparsity`
			`Deconv`(7×7)
				`Deconv`(2×2)
				`BN`
				`ReLU`
				`Deconv`(2×2)
				`sparsity`
				`Deconv`(7×7)

Accuracy (%)	**95.76%**	95.65%	95.42%	95.42%

3.3.8 The Effects of the Preprocessing Methods

We compare the performance of HyResNet `Model 4-C1` after three different preprocessing methods were employed. All experiments are evaluated on CIFAR-10. Each model is evaluated with 0.3 dropout and then without a dropout. Table 3.12 summarizes the results.

Table 3.12: The effects of various preprocessing methods on the performance of the proposed HyResNet `Model 4-C1` employed on CIFAR-10. Moreover, it shows the impact of dropout in the convolutional component.

Preprocessing	**Accuracy (%)**	
	No Dropout	**Dropout**
Mean	**95.64%**	**95.89%**
Mean/Std	95.55%	95.48%
ZCA Whitening	95.04%	95.23%

The results show that HyResNet achieves the best performance when the data is preprocessed by subtracting the means of the RGB channels with and without dropout with accuracies of 95.89% and 95.64%, respectively. Furthermore, when dropout is employed, HyResNet shows a better performance than removing the dropout.

3.3.9 Model 5 Variations

We adopt the `WRN 40-10` structure to construct our `HyResNet 40-10` model. The accuracy of `HyResNet 40-10` on mean-normalized CIFAR-10 without dropout is 96.32. Furthermore, this model learns $62.1M$ parameters. Table 3.13 shows the results.

The unsupervised component of `WRN 40-10` is constructed with two DRBs, where each deconvolutional layer in the DRB is preceded with a batch normalization and a ReLU. The UC ends with the layers $sparsity$ and $Deconv(7 \times 7)$.

Table 3.13: The error rate (%) of the `Models 5 & 6` on mean-normalized CIFAR-10. * as reported by [79] on mean/std normalization. ** The original paper [79] did not report a result.

		Dropout	No Dropout
Model 5 (40-10)	CNN	$96.2\%^{*}$	$-^{**}$
	HyResNet	**96.34%**	96.16%
Model 6 (28-10)	CNN	$96.11\%^{*}$	$96\%^{*}$
	HyResNet	**96.06%**	95.97%

3.3.10 Model 6 Variations

We adopt the `WRN 28-10` to construct our `HyResNet 28-10`. We evaluated `WRN 28-10` with and without dropout. `WRN 28-10` learns $42.7M$ parameters. We named the models with dropout as `Model 6-A` and the ones without dropout as `Model 6-B`. Table 3.13 shows the model performance on CIFAR-10. The unsupervised component of the model `HyResNet 28-10` is constructed from two DRBs. Each deconvolutional layer of the DRBs is preceded with a batch normalization and ReLU. Additionally, the layers $sparsity$ and $Deconv(7 \times 7)$ are appended at the end of the UC. We evaluated on mean-normalized CIFAR-10 3.13. We achieved accuracies of 96.06% and 95.97% for `Model 6-A` and `Model 6-B`, respectively.

3.4 Conclusion

We proposed the Hybrid Residual Network Method (HyResNet). HyResNet utilizes the power of both supervised and unsupervised learning in a single supervised model. We conducted an empirical study to measure the efficacy of the proposed HyResNet on visual object recognition tasks using the benchmark datasets CIFAR-10 and CIFAR-100. We evaluated HyResNet in various configurations and settings. HyResNet achieved comparable results to the state-of-the-art methods on CIFAR-10 with a classification error

Table 3.14: A summary of state-of-the-art supervised neural network methods performance on classifying the datasets CIFAR-10 and CIFAR-100. [1] We implemented and ran because it was not available in the original paper. [2] Normalized with mean/std, [3] summary of our best results.

			Test Error Rate (%)			
			CIFAR-10		CIFAR-100	
Method	Dim	# of param.	Dropout	No Dropout	Dropout	No Dropout
Network in Network [46]			8.81%		35.68%	
FitNet [60]	19	$2.5M$	8.39%		35.04%	
Deeply-Supervised Nets. [45]		$2.5M$	7.97%		34.57%	
Highway Networks [73]	19	$2.3M$	7.76%		32.39%	
All-CNN [71]	11	$1.4M$	9.08%			
ELU [13]	18		6.55%		24.28%	
Residual Networks						
Deep ResNet [27]	110	$1.7M$		6.43%		25.16%
	1202	$10.2M$		7.93%		27.82%
Pre ResNet [28]	110	$1.7M$		6.37%		
	164	$1.7M$		5.46%		24.33%
	1001	$10.2M$		4.92(4.64)%		22.71%
Stochastic Depth ResNet [34]	110	$1.7M$		5.23%		24.58%
	1202	$10.2M$		4.91%		
ResNet of ResNet [76]	$58-4$	$13.3M$		3.77%		
DenseNet-BC (k=40) [33]	190	$25.6M$		3.46%		17.18%
Wide ResNet [79]	$40-4$	$8.9M$		4.53%		21.18%
	$40-10$	$56.0M$	3.8%		18.3%	
	$16-8$	$11.0M$		4.27%		20.43%
	$28-10$	$36.5M$	3.89%	4.00%	18.85%	19.25%
	$22-4$[1,2]	4.3M	4.52%	4.45%		
HyResNet (ours)[3]	$22-4$	$5.3M$	4.31%	4.12%		22.06%
	$40-10$	$62.1M$	3.71%	**3.68%**		**18.85**%
	$28-10$	$42.7M$	3.94%	4.03%		

rate of 3.68%. Furthermore, HyResNet achieved comparable results on CIFAR-100. Table 3.14 compares the HyResNet results with the state-of-the-art results in the literature on the two aforementioned benchmark datasets.

CHAPTER 4 DEEP SEMI-SUPERVISED LEARNING

4.1 Introduction

Deep learning methods are among the best methods addressing various machine learning problems including object recognition, classification, and image segmentation. For example, CNNs have demonstrated great performance in handling computer vision problems, specifically the visual object classification tasks [41]. In particular, a key to the success of supervised deep learning methods is the availability of a sufficiently large labeled training data [11]. Unfortunately, creating a sufficiently large labeled training data with enough examples for each class (e.g., ImageNet [62], COCO [47], VGGFace2 [8]) is not an easy task. This task is time-consuming, expensive, susceptible to noise and mislabeled [38] examples, and requires experienced human effort. On the other hand, unlabeled data is easily available and inexpensive to collect. For example, a dataset of unlabeled images can be collected from online publicly available resources such as webpages and videos. Therefore, there has been a recognizable advance in exploiting the available large unlabeled data alongside with the limited labeled data to enhance the performance of deep learning methods.

Unlabeled data can be used to pre-train and initialize model parameters to achieve superior performance over random initialization. In turn, this pre-training can help reduce the total number of training epochs. In particular, Glorot et al. [21] introduced a smart random initialization to achieve superior performance in a smaller number of epochs without requiring any pre-training.

Generally speaking, supervised deep learning methods (e.g., CNN) learn class-specific sets of features whereas unsupervised deep learning methods (e.g., AE) learn general

detailed features [4]. Given the relative strengths of each approach, methods to combine both approaches have been forthcoming. Recently, several approaches were developed to combine supervised and unsupervised learning methods into a single model. Hinton et al. [29] proposed the use of unsupervised learning as a pre-training step for a supervised model. In other approaches [59, 58, 69, 23], supervised and unsupervised learning methods were combined into a single model and trained simultaneously. Moreover, other methods [10] used an unsupervised method to extract a general set of features that was later used to train a supervised model.

Self-training [56] is a semi-supervised learning (SSL) method that utilizes the access to large number of unlabeled examples and limited number of labeled data. Self-training has been proven to be promising in many machine learning applications [81]. However, it showed a few shortcomings: e.g., (1) its performance may fluctuate due to the fact that some unlabeled instances will remain unlabeled during the training phase; (2) its accuracy may be deteriorated because erroneous predictions will lead to adding mislabeled instances; this is especially true when the size of labeled data is small. Furthermore, the performance of self-training methods also depends on other factors such as the characteristic of data and confidence measures for adding instances during training: an erroneous confidence measure often leads to mislabeled instances being added to the labeled set.

In this chapter [26], we combine the strengths of both supervised and unsupervised neural networks into a single deep semi-supervised learning method (DSSL). DSSL exploits the availability of large-scale unlabeled data in conjunction with limited labeled data to learn both generic features (unsupervised) and discriminative features (supervised). It then combines the generic and discriminative features into a single set of features. DSSL

utilizes the state-of-the-art ResNet methods [27, 28, 79] to construct a residual-based convolutional component. This convolutional component then forks into two branches. The first branch is a supervised one ending with a supervised loss function. The second branch uses a new residual deconvolutional component ending with an unsupervised loss function. A DSSL model accepts a large number of unlabeled examples and a few labeled examples for training. The number of labeled examples can be as small as 5% of the overall input data. Some applications can tolerate even a smaller percentage of labeled training examples. As such, DSSL aims to reduce curating large labeled training data to only a few labeled examples while maintaining a high performance. This in turn helps reduce the overall cost and time needed. Furthermore, it aims to reduce the effect of mislabeled and noisy examples. Although DSSL is simple and easy to implement with existing machine learning libraries [37, 16, 1], it achieves state-of-the-art performance. Empirically, we show DSSL achieves state-of-the-art performance on several semi-supervised tasks.

4.1.1 Contribution

The primary contribution of this chapter is a semi-supervised learning method using the self-training ideas. The architecture of the proposed method combines the strengths of both supervised and unsupervised neural networks into a single semi-supervised deep learning method which we refer as DSSL. DSSL exploits the availability of large-scale unlabeled data in conjunction with limited labeled data to learn both generic features (unsupervised) and discriminative features (supervised) in a self-training fashion. To the best of our knowledge, the proposed method trained in the proposed procedure is among the first of its kind. The novelty of the proposed method is a generalization of self-training,

but differs standard self-training in three-fold: (1) The network architecture branches into two parallel tracks after the input goes through a series of deep convolutional blocks. One track is trained for the classification and the other is trained in an autoencoder fashion. The gradient from each track is aggregated and back-propagated to the initial deep convolutional unit. (2) The training algorithm uses a self-training to assign labels to all of the unlabeled samples and employs them later in the next epoch in the supervised training of the model. (3) The proposed method utilizes every example, both labeled and unlabeled, with both supervised and unsupervised methods.

4.1.2 Problem Formulation.

The goal in a classical supervised learning framework is to learn a decision model $\mathcal{M}$ from n available training examples. The training examples are denoted by $\mathcal{D}_n$, where n is the total number of training examples in $\mathcal{D}$. We denote a training example (e.g., image) by $x_i \in \mathbb{R}^d$, where $i \in \{1, \ldots, n\}$. Also, we denote the class label of x_i by $y_i \in \{1, \ldots, C\}$, where C is the total number of different classes in $\mathcal{D}_n$. In general, the training dataset is represented by $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^{n}$.

In SSL settings, the decision model is learned when the labels of a limited subset of $\mathcal{D}_n$ are available. The labeled subset of l examples is denoted by $\mathcal{D}_l$, where $\mathcal{D}_l \subset \mathcal{D}_n$. The labels of the remaining u examples are not available. The unlabeled data is denoted by $\mathcal{D}_u$, where $\mathcal{D}_u \subset \mathcal{D}_n$ and $u = n - l$. Consequently, the training data in a SSL framework is denoted by $\mathcal{D}_n = \{\mathcal{D}_l \cup \mathcal{D}_u\}$.

In particular, we employ self-training method, which is one of the common classes of SSL [81]. Self-training methods are iterative algorithms, where a model $\mathcal{M}$ is learned

from the available labeled subset $\mathcal{D}_l$ first. Then, the learned model $\mathcal{M}$ is used to predict labels for the unlabeled subset $\mathcal{D}_u$. The predicted labels with confidence scores more than a predefined confidence threshold p are retained and used in future steps. After that, the model $\mathcal{M}$ is retrained on the combined labeled and predicted data. This procedure is repeated until meeting a stop condition.

4.2 Related Work

A variety of approaches to develop semi-supervised learning methods to learn from large unlabeled data with a restricted number of labeled examples have been undertaken [81]. In fact, a major focus of current research is to achieve semi-supervised method performance comparable to the performance of state-of-the-art supervised methods.

The applications of self-training method include various machine learning problems such as natural language processing [51] and object detection [61]. One common problem of the self-training methods is that they are prone to error and poor prediction which can delude the model. The common solution to the problem caused by poor prediction is to set a predefined confidence threshold for using the labels with the confident score more than the threshold in each iteration. DSSL is a self-training semi-supervised method that considers standard prediction method. It predicts and uses the labels for all unlabeled data without any thresholds. The DSSL architecture and training procedure overcome the self-training problem of poor prediction. In the remaining part of this section, we focus on semi-supervised deep learning methods closely related to our proposed method.

Sajjadi et al. [64] proposed a transform/stability semi-supervised loss function that relies heavily on a learning model generating different outputs every time it is evaluated. The variations in output stem from different sources of randomization within the model such

as dropout [72] and augmentation. In the approach of Sajjadi et al. [64], each minibatch passes n times through heavy augmentation and model evaluation during training. The loss term is calculated as the sum of all pairwise mean squared distances between the n outputs. Additionally, they utilize a mutual-exclusivity supervised loss term [63]. The computational cost of evaluating the transform/stability loss function during training increases linearly as a function of the number of evaluations n. In the approach of Laine et al. [42], they presented Π-model, a temporal self-ensembling semi-supervised method, which is a special case of the transform/stability semi-supervised loss function of Sajjadi et al. [64] with $n = 2$. Our DSSL differs from the previous methods by employing supervised and unsupervised loss functions in a single model and evaluating the model on each training example only once for each epoch. This helps DSSL achieve superior computational efficiency as the number of evaluations is constant for each epoch.

Rasmus et al. [59] introduced the Γ-model, a subset of the ladder networks [78] that employed an encoder-decoder network architecture to tackle the semi-supervised learning problem. In the Γ-model, all ladder connections excluding the highest one are discarded. The highest connection is then used to construct two parallel, identical branches. One branch takes the original training input whereas the second branch accepts a noisy copy of the original training input. In this method, the unsupervised loss term is calculated as the squared difference between the pre-activation output of both noisy and clean branches. Our DSSL differs from the Γ-model by using one branch that splits into two supervised and unsupervised branches. In DSSL, a clean input passes through the network only once. Here, the unsupervised loss term is the mean squared difference between the the original training input and the corresponding output of the unsupervised branch. Fur-

thermore, DSSL classifies the unlabeled data every other epoch and uses the predicted labels to train the next epoch using both the supervised and unsupervised loss functions. This approach differs from the one proposed by Rasmus et al. [59], which uses only the unlabeled data with the unsupervised loss function.

Tarvainen et al. [77] proposed a technique to improve the semi-supervised learning methods that averages the weights of deep neural networks after each iteration as opposed to each epoch[42] to better predict labels. Here, the consistency loss function was used to estimate the distance between what they called a "student" model and a weight-averaged model called a "teacher" model.

Finally, Generative Adversarial Networks (GANs) [24] have been employed in various semi-supervised models [50, 70, 65, 39]. GANs learn generative models using game theory. In general, GANs set up a game between a generator and a discriminator. The discriminator is trained on a set of images. The generator creates images, which are assumed to come from the same distribution as the training images. The discriminator has to discriminate between fake and real images using a supervised loss function. The final goal of the generator is to deceive the discriminator. In the spirit of Generative Adversarial Networks, Salimans et al. [65] proposed a discriminative GAN model to tackle the semi-supervised problem.

4.3 Method

We name the DSSL models following the same naming scheme introduced by Zagoruyko et al. [79]. The name of a DSSL model shows the overall number of convolutional layers and the widening factor. The widening factor determines the number of feature maps in each layer.

DSSL is constructed from three components (Fig. 4.1 and Table 4.1). (a) A *Convolutional Component* (CC) consisting of sequence of deep convolutional blocks. (b) A *Supervised Component* (SC) comprising an average pooling layer, a linear layer, and a supervised loss function. We employ a supervised loss function that combines the negative log likelihood of softmax. The supervised loss function is described as:

$$\text{loss}_{\text{CE}}(z_{sc}, y) = -\sum_{j} log\left(\frac{e^{z_{sc_j}[y_j]}}{\sum_i e^{z_{sc_j}[i]}}\right). \tag{4.1}$$

where j is the example number in a minibatch b, z_{sc_j} is the output of the SC, which is a vector with a score for each class, y_j is the index of the target class of the example j. (c) An *Unsupervised Component* (UC) comprising a sequence of deconvolutional blocks followed by a sparsity layer, a deconvolutional layer, a `Tanh` layer, and finally an unsupervised loss function. The unsupervised loss function measures the mean squared error between the original input and the output of the UC, which is defined as:

$$\text{loss}_{\text{MSE}}(z_{uc}, b) = \sum_{k} ||z_{uc_k} - b_k||_2^2. \tag{4.2}$$

where b is an input minibatch to the model, b_k is the k^{th} example (e.g., image) in b, and z_{uc} is the output of the UC.

The convolutional component includes the following sequence. A single convolutional layer with a kernel size of (3×3). This convolutional layer accepts the original data as input (e.g., images) and produces an output of the same height and width as the input. Furthermore, the produced output has 16 feature maps. The convolutional layer

is followed with three convolutional residual units (CRUs). Each CRU is assigned to an initial number of features. The initial number of the feature maps for the three CRUs in a DSSL are 16, 32 and 64, respectively. The final number of feature maps of each CRU is the widening factor times the predefined constant factor of that CRU. Consequently, the number of feature maps for the three CRUs of DSSL 22-4 are $64\ (4 \times 16)$, $128\ (4 \times 32)$, and $256\ (4 \times 64)$, receptively. A CRU in DSSL is constructed from a sequence of convolutional residual blocks. The convolutional residual block is constructed from two parallel branches: a *residual branch* and a *shortcut branch*. The residual branch consists of two convolutional layers, each with kernel size (3×3). Each convolutional layer is succeeded with a batch normalization and a ReLU. The shortcut branch passes the same input signal (identity).

The first CRB in a CRU increases the number of feature maps (FMs) of its input and reduces the width and height. For example, assuming that the input size to the CRU1 is of the size (32×32) and 16 feature maps, the first CRB of the CRU1 reduces the input size to (16×16) and increases the number of FMs to 64. All other CRBs in the CRU preserve the same input size and number of FMs.

The supervised component (SC) comprises two layers. An average pooling layer with a region size of (8×8) and a stride of 1, and a linear layer (denoted by FC in Table 4.1). Finally, the SC ends with a supervised loss function.

Generally speaking, a network constructed with a convolutional component (CC) followed by a supervised component (SC) makes a wide residual network (WRN) proposed by [79].

The unsupervised component (UC) in DSSL includes a sequence of two deconvolu-

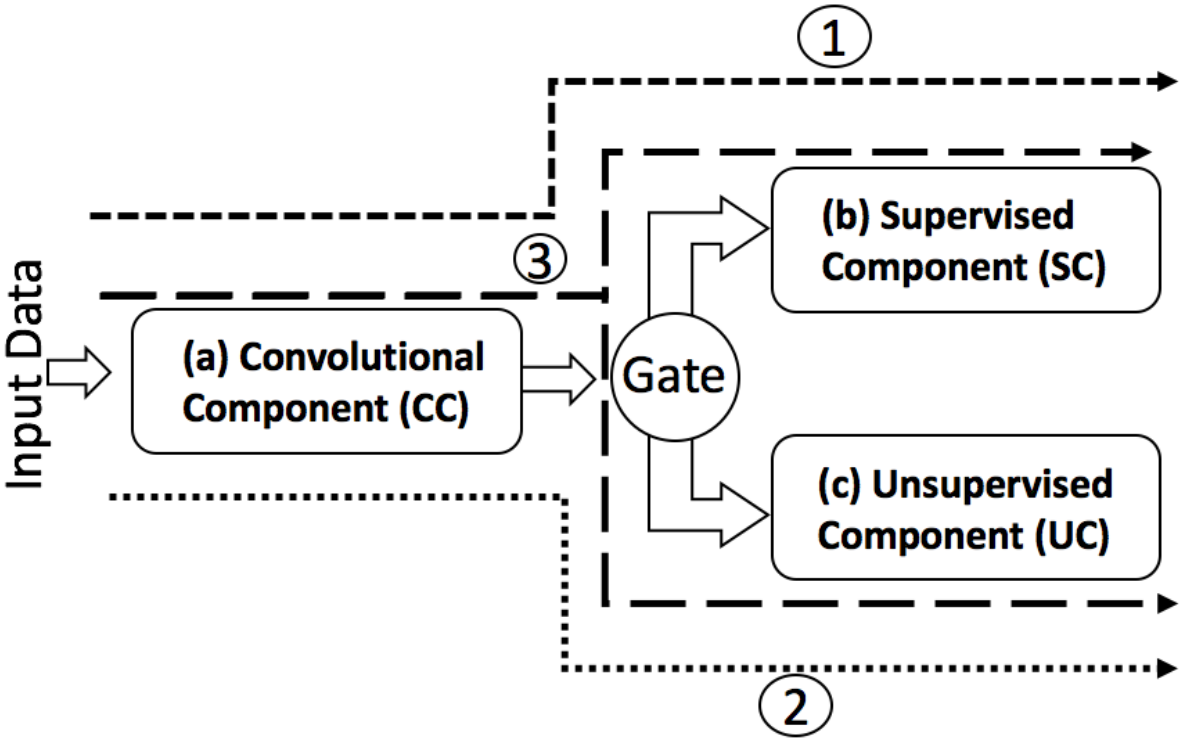


Figure 4.1: The structure of DSSL with the three main components: (a) Convolutional component (CC), (b) Supervised component (SC) and (c) Unsupervised component(UC). We identify three main paths in DSSL. (1) The short-dashed line path shows the CC followed by the SC, which can be viewed as a residual neural networks. (2) The dotted line path shows the CC followed by the UC, which can be viewed as a convolutional autoencoder. (3) The long-dashed line path, shows the CC forks to the SC and the UC.

tional residual units (DRUs), a batch normalization layer, a ReLU, a sparsity, a deconvolutional layer with a kernel size of (7×7), a `Tanh` layer, and finally an unsupervised loss function. Each DRU in DSSL is constructed from a single DRB block. The residual branch of the DRB includes two deconvolutional layers with kernel sizes (4×4) and (3×3), respectively. The shortcut branch is a deconvolutional layer with a kernel size of (2×2) and a stride of 2. The DRU reverses the CRU operations. In particular, a DRB reduces the number of feature maps, and increases the width and height of input. For example, in DSSL 22-4, the UC accepts an input of the width and height (8×8), and 256 FMs. The first DRU creates output of (16×16) and 128 feature maps.

The sparsity layer in UC can be any type of sparsity. DSSL utilizes the *spatial sparsity* proposed by Makhzani et al. [49]. In the feed-forward, the spatial sparsity sets all of the activation values in every feature map to zeros except the highest value is retained. Then, the error is back-propagated through the non-zero units to update the model parameters.

Similar to [79], the total number of convolutional layers in a DSSL is determined by the following formula:

$$m = 6c + 4. \tag{4.3}$$

where c is the number of CRB in a CRU, and m is the total number of convolutional layers. For example, the DSSL 22-4 model includes 3 CRBs in each CRU. Therefore, it includes 22 convolutional layers.

A residual neural network compiles a sequence of residual blocks [27]. A residual block is constructed from two parallel branches or connections (Fig. 2.2). We denote the top branch as *residual branch* and the bottom branch as *shortcut*. In general, a residual

Table 4.1: The detailed architectures of the models DSSL 22-4 and 28-10.

	Unit	Output size	DSSL 22-4	DSSL 28-10
CC	Conv1	(32×32)	[Conv(3×3)]$\times 1$, 16	[Conv(3×3)]$\times 1$, 16
	CRU1	(32×32)	[CRB (3×3)]$\times 3$, 64	[CRB (3×3)]$\times 4$, 160
	CRU2	$16 \times 16)$	[CRB (3×3)]$\times 3$, 128	[CRB (3×3)]$\times 4$, 320
	CRU3	(8×8)	[CRB (3×3)]$\times 3$, 256	[CRB (3×3)]$\times 4$, 640
SC			[(8×8) AveragePool, stride 1] $\Rightarrow$ FC	
UC	DRU1	(16×16)	[DRB (4×4),(3×3)]$\times 1$, 128	[DRB (4×4),(3×3)]$\times 1$, 320
	DRU2	(32×32)	[DRB (4×4),(3×3)]$\times 1$, 64	[DRB (4×4),(3×3)]$\times 1$, 160
			BN $\Rightarrow$ ReLU $\Rightarrow$ Sparsity $\Rightarrow$ Deconv(7×7) $\Rightarrow$ Tanh	

block is represented by the following formula:

$$x_i = \mathcal{F}(\mathrm{W}_{shortcut}, x_{i-1}) + \mathcal{G}(\mathrm{W}_{resid-branch}, x_{i-1}). \tag{4.4}$$

where x_{i-1} is the input to the residual block i, x_i is the output of a residual block i that is used as an input to the next component, $\mathcal{F}$ is the shortcut branch function and $\mathrm{W}_{shortcut}$ are the shortcut branch parameters (weights), and $\mathcal{G}$ is the residual branch function and $\mathrm{W}_{resid-branch}$ are the residual branch parameters (weights).

A residual branch includes at least one convolutional or deconvolutional [48] layer. Each layer is preceded with a set of layers such as batch normalization and ReLU. The shortcut can either be a *skip connection* that passes the same input signal (identity) or a single convolutional / deconvolutional layer that samples the data. The join after the two connections is an element-wise addition summing up the output of the branches. The output of the join is then passed to the next component. We discriminate between two types of residual blocks based on the use of convolutional or deconvolutional layers. A *convolutional residual block (CRB)* uses a convolutional set of layers whereas a *deconvolutional residual block (DRB)* uses deconvolutional layers.

Algorithm 2 The DSSL Algorithm

Require: $\mathcal{D}$ = training, $\mathcal{D}_l$ = labeled, and $\mathcal{D}_u$ = unlabeled datasets.
Require: f_{cc}, f_{sc} and f_{uc} = convolutional, supervised and unsupervised components, respectively.
Require: $loss_{\text{CE}}$ and $loss_{\text{MSE}}$ = cross-entropy (supervised) and mean-squared error (unsupervised) loss functions.
Require: *gate* = a split that decides which branches contribute to the model in every epoch.

for *epoch_no* $\in [1, max\_no\_epochs]$ **do**
  if *epoch_no is Odd* **then**
    gate $\leftarrow$ *on-off*
    $\mathcal{D} \leftarrow \mathcal{D}_l$
  else if *epoch_no* $== 2$ **then**
    gate $\leftarrow$ *off-on*
    $\mathcal{D} \leftarrow \{\mathcal{D}_l \cup \mathcal{D}_u\}$
  else ▷ *The epoch_no is even AND* > 2
    gate $\leftarrow$ *on-on*
    Classify $\mathcal{D}_u$ *and then update* $\mathcal{D}_u$ *labels*
    $\mathcal{D} \leftarrow \{\mathcal{D}_l \cup \mathcal{D}_u\}$
  end if
  for each minibatch $b \in \mathcal{D}$ **do**
    $w_b \leftarrow f_{cc}(aug(b))$
    if *gate* $==$ *on-on* OR *gate* $==$ *on-off* **then**
      $z_{\text{sc}} \leftarrow f_{\text{sc}}(w_b)$
      $loss_{\text{CE}}(z_{\text{sc}}, y) \leftarrow -\sum_j log\left(\frac{e^{z_{\text{sc}_j}[y_j]}}{\sum_i e^{z_{\text{sc}_j}[i]}}\right)$
    end if
    if *gate* $==$ *on-on* OR *gate* $==$ *off-on* **then**
      $z_{uc} \leftarrow f_{\text{uc}}(w_b)$
      $loss_{\text{MSE}}(z_{\text{uc}}, b) \leftarrow \sum_k ||z_{\text{uc}_k} - b_k||_2^2$
    end if
    Update weights using SGD with momentum
  end for
end for

The residual branch in DRB (Fig. 2.2) includes two deconvolutional layers with kernels (4×4) and (3×3), respectively. Each deconvolutional layer is preceded with a batch normalization and a ReLU. The shortcut branch includes a single deconvolutional layer with a (2×2) kernel and a stride of 2. Similar to Zagoruyko et al. [79], a residual branch of the CRBs includes a sequence of two (3×3) convolutional layers. Each convolutional layer is preceded with a batch normalization and a ReLU. A dropout layer separates the second convolutional layer from the previous ReLU layer.

We call a sequence of residual blocks of the same type (e.g., convolutional) and equal number of feature maps (FMs) a *residual unit* (RU). The RU can be either a *convolutional residual unit* (CRU) or a *deconvolutional residual unit* (DRU). The first residual block of the RU changes the number of FMs and the input size. All following blocks within an RU maintain the same number of FMs and data size. For instance, in a CRU, the first CRB decreases the input size (e.g., from 32 to 16) and increases the number of feature maps (e.g., from 64 to 128). Furthermore, in a DRU, the first DRB increases the input size (e.g., image size from 8 to to 16) and decreases the number of FMs (e.g., feature maps from 256 to 128).

DSSL is constructed from three components (Fig. 4.1). (a) A Convolutional Component (CC) that includes a sequence of CRUs. (b) A Supervised Component (SC) comprising an average pooling layer, a linear layer and a softmax layer. (c) An Unsupervised Component (UC) comprising of a sequence of DRUs followed by a sparsity layer, a deconvolutional layer, a `Tanh` layer, and finally an unsupervised learning criterion. A model constructed with a CC followed by an SC only is a ResNet. A model constructed with a CC followed by a UC can be viewed as a convolutional autoencoder (Coanv-AE). In

this project, we construct our CC based on the Wide Residual Network proposed by Zagoruyko et al. [79]. However, the CC can be replaced with any convolutional component such as deep ResNet [27], VGG [68], and GoogLeNet [75].

DSSL implements a fork with a gate (Fig. 4.1) that joins the CC from one end with the UC and SC from the other end. The gate is assigned one of three mode to decide which active components to learn in the feed-forward as well as contributing to the model in back propagation by directing the output of the convolutional component w to the right components. The gate mode is updated only at the beginning of every epoch and does not change until the beginning of the next epoch. If the gate mode is *on-on*, then in the feed-forward a copy of the CC output w is directed to both the SC and UC simultaneously. In the feed-backward, the gradients from both the SC and UC are back-propagated to update the weights of both components, respectively. Finally, the gradients are summed up and back-propagated to update the CC weights. All three components learn and contribute to the model in this mode. In the *on-off* gate mode, the SC is *on* which means it learns and contributes. The UC, on the other hand, is *off* and neither learns nor contributes to the model. The final gate mode is the *off-on* mode where the SC is *off* and the UC is *on*.

4.3.1 DSSL Algorithm

A DSSL (Algorithm 2) accepts an input of a dataset D. D consists of labeled data D_L, unlabeled data D_U, or both. Moreover, the labels of D_L never change whereas the labels of D_U are initially assigned to random values. Training a DSSL model starts the first epoch ($t = 1$) with mode of the $gate =$ *on-off* and $D = D_L$ (short-dashed path in Fig. 4.1). This step helps to learn CC and SC weights with all available labeled training data D_L. After

the first epoch is completed, the mode of the gate is switched such that $gate =$ *off-on* and the dataset consists of $D = D_U \cup D_L$. The second epoch ($t = 2$) is evaluated on $D = D_U \cup D_L$ (the dotted line in Fig. 4.1). This epoch aims to learn UC weights using all available training data regardless of the labels. In the third epoch ($t = 3$), the mode of the gate is switched once more such that $gate =$ *on-off* and $D = D_L$. Then, the model is evaluated to produce a supervised model M_t. After that, DSSL uses M_t to classify the unlabeled dataset D_U. These labels are saved and used accordingly in future epochs. Next, the mode of the gate is switched back such that $gate =$ *on-on* and $D = D_U \cup D_L$. It should be noted that this D now includes the new predicted labels from the previous epoch. Then, DSSL is evaluated on D for the fourth epoch ($t = 4$) (long-dashed path in Fig. 4.1). In this epoch, the supervised loss function uses the labels of both D_L and D_U. The model then alternates between the steps of epochs $t = 3$ and $t = 4$ with gate modes alternating between *on-off* and *on-on* until the maximum number of epochs is reached.

We study DSSL using various architectures and different settings of hyper-parameters. We report the best performing models. For example, we study DSSL with convolutional layer kernel (filter) sizes (3×3), (5×5), and (7×7). DSSL attains the best performance with a kernel size of (3×3). Therefore, all of the used models throughout this chapter have kernel sizes of (3×3) unless specified otherwise. Moreover, we tested various combinations of deconvolution layer kernel sizes with DSSL. Our experiments show that selecting the right deconvolutional layer kernel size plays a significant role in determining the overall performance of a DSSL model.

(1) First epoch ($t = 1$): $gate =$ *on-off* and $D = D_L$ (short-dashed path).

(2) Second epoch ($t = 2$): $gate =$ *off-on* and $D = D_U \cup D_L$ (the dotted line).

(3) Third epoch ($t = 3$): $gate =$ *on-off* and $D = D_L$.

- This epoch produces a supervised model M_t.
- Use M_t to classify D_U and save the new predicted labels.

(4) Fourth epoch ($t = 4$): $gate =$*on-on* and $D = D_U \cup D_L$ (long-dashed path).

(5) The model repeats the epochs $t = 3$ and $t = 4$ until it the maximum number of epochs.

4.3.2 Split Layer

The Input signal to the split layer includes also four dimensions $z_2^{N \times M \times D \times D}$; where N is the number of samples per minibatch, M is the number of input feature maps, $D \times D$ is the size of each input the sample.

- The network **Feed - Forward**: The output z of the common part of the model is copied twice to both branches (unsupervised and supervised).
- The network **Feed - Backward**: During back-propagation $grad_{sup}$ (gradients from supervised branch) and $grad_{unsup}$ (gradients from unsupervised branch) add to each other in element-wise fashion:

$$grad = grad_{sup} + grad_{unsup}. \tag{4.5}$$

Then calculated $grad$ back-propagates through the common part.

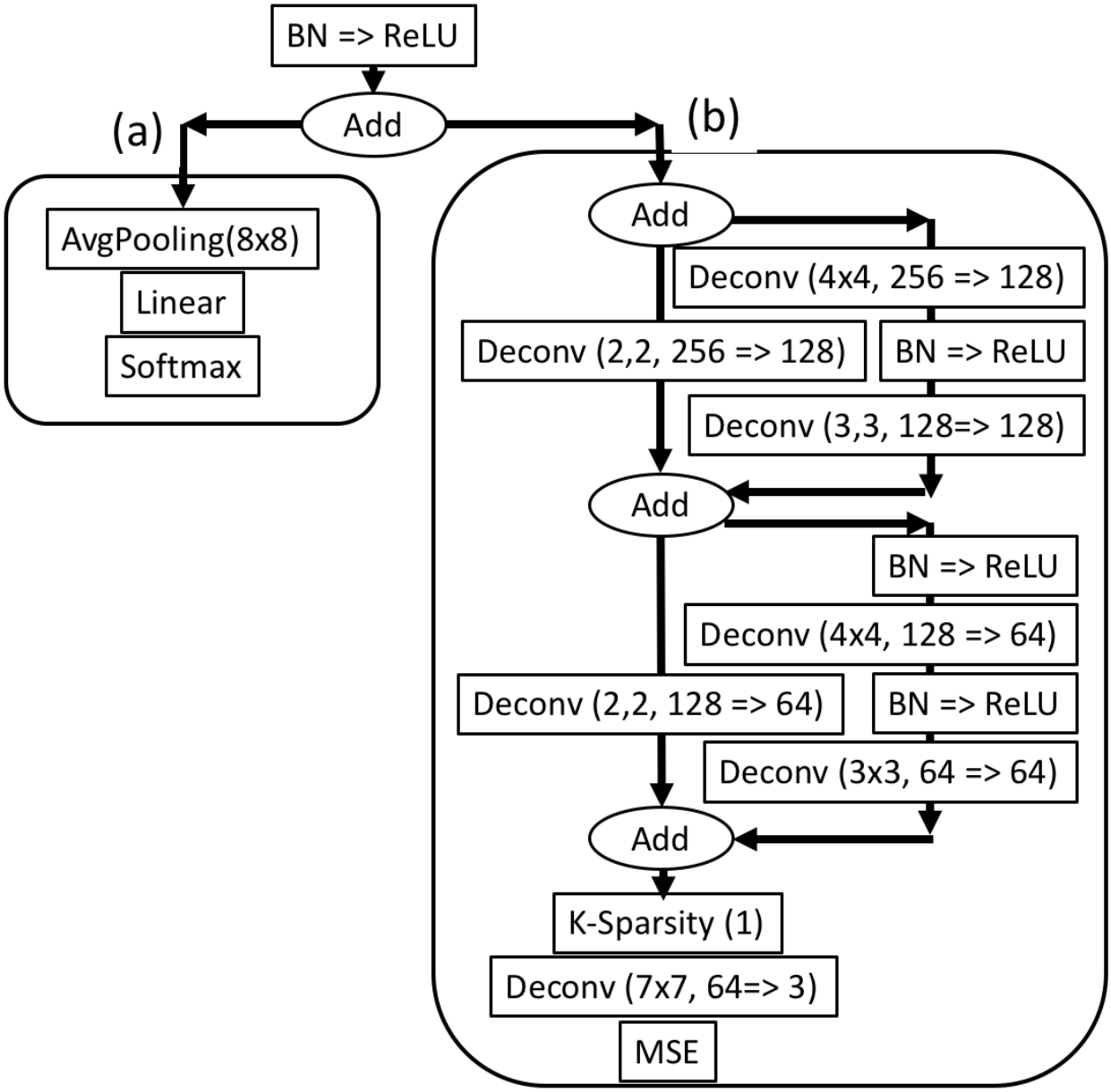


Figure 4.2: The DSSL model components (a) the supervised component with all layers (left), (b) the unsupervised component with all layers (right).

Table 4.2: Details and statistics of the evaluated datasets. The part‡ shows the number of unlabeled images. The size shows the width and heights of the images.

Dataset	#Classes	#Training	#Testing	Color	Size
CIFAR-10 [40]	10	50K	10K	✓	32×32
CIFAR-100 [40]	100	50K	10K	✓	32×32
STL-10 [15]	10	5K (100K‡)	8K	✓	96×96
MNIST [44]	10	60K	10K		28×28
SVHN [54]	10	73,257	26,032	✓	32×32

4.4 Experiment

In this section, we empirically show the efficiency of DSSL on the benchmark datasets CIFAR-10 [40], CIFAR-100 [40], STL-10 [15], SVHN [54], and MNIST [44]. Table 4.2 shows the statistics of the used datasets.

Experiment Setup

We train all models in this project from scratch and do not fine-tune any one of them. Unless otherwise specified, all of our experiments adopt the protocols used by Zagoruyko et al. [79]. For every dataset, we report the *mean* and *standard deviation* (mean $\pm$ std.) of the classification error rates. For each dataset, we generate multiple random partitions consisting of different ratios of labeled/unlabeled data. These error rates are calculated by taking the average and standard deviation of error rates of the random partitions. The state-of-the-art results are provided in the tables using a boldface font.

Unfortunately, the state-of-the-art residual network models use a massive number of parameters which require several days of continuous training on powerful machines equipped with multiple GPUs and a huge amount of RAM. For example, the state-of-the-art WRN model [79] (named in the original paper as WRN-40-10) that achieved the best results on CIFAR-10 uses $56M$ parameters. Therefore, we constructed our DSSL with a

fairly medium sized CC.

We limit preprocessing to the following steps only. For CIFAR-10 and CIFAR-100, we divide the images by 255 to scale them to the range of $[0, 1]$. We then subtract the means of the RGB channels. We normalize STL-10 by subtracting the means and dividing by the standard deviation of the channels. We normalize SVHN and MNIST by dividing the images by 255 to scale it in the range $[0, 1]$. We do not apply aggressive augmentation. Instead, we only apply the standard augmentation presented by Lee et al. [45] on CIFAR-10, CIFAR-100 and STL-10. This includes random horizontal flips and random crops after padding each side of the image by 4 pixels. For the SVHN and MNIST datasets, we only take random crops after padding each side of an image by 4 pixels each.

In all of our experiments, we use a minibatch size of 128 images except for the STL-10 experiments. For the STL-10 experiments we use a minibatch size of 32 since for this dataset the image sizes are large. We train our networks using SGD with Nestrov momentum. We also employ the cross-entropy and mean squared error for the supervised and unsupervised loss, respectively. DSSL shows the best performance when we use dropout [30] and batch normalization [36]. Furthermore, we employ the rectified linear units (ReLU) activation function [52]. Moreover, we find that turning off biases in all layers leads to better performance. Therefore, we turn off biases in all of our experiments. We initialize all layers with the method introduced by Glorot et al. [21]. We run the experiments in two settings. First, CIFAR-10, CIFAR-100, STL-10, and MNIST experiments are evaluated for 320 epochs. In this situation, the learning rate starts at 0.1 and then decreases after at epochs 120, 240 and 280 at a rate of 0.2 with a dropout of 0.3. Second, the SVHN experiments are evaluated for 280 epochs with a starting learning rate of 0.01.

The learning rate decreases at epochs 160 and 240 at a rate of 0.1 with a dropout of 0.4.

We study DSSL using various architectures and different settings of hyper-parameters. We report the best performing models. For example, we study DSSL with convolutional layer kernel (filter) sizes (3×3), (5×5), and (7×7). DSSL attains the best performance with a kernel size of (3×3). Therefore, all of the used models throughout this project have kernel sizes of (3×3) unless specified otherwise. Moreover, we tested various combinations of deconvolutional layer kernel sizes with DSSL. Our experiments show that selecting the right deconvolutional layer kernel size plays a significant role in determining the overall performance of a DSSL model.

4.4.1 CIFAR-10

The CIFAR-10 dataset consists of $60,000$ RGB images of size 32×32 pixels. The dataset is divided evenly into 10 different classes. Furthermore, it is available in two pre-defined parts that we adopt hereafter in our experiments. These parts consist of $50,000$ and $10,000$ examples used for training and testing, respectively. Figure 4.3 shows a sample from CIFAR-10 dataset.

We preprocess CIFAR-10 by dividing images by 255 to shift them to the range of $[0, 1]$. Then, we subtract the means of RGB channels. We use the means 0.49, 0.48, and 0.45 for red, green and blue channels, respectively.

We create five labeled datasets with $4,000$ labeled examples each. We construct each dataset by randomly selecting 400 examples per class from the CIFAR-10 training dataset (i.e., D_L). We retain their actual labels and then treat the rest of the training data as unlabeled (i.e., D_U). We evaluate DSSL 22-4 once on each dataset. Moreover,

Figure 4.3: Sample from CIFAR-10 dataset. CIFAR-10 compiles images from 10 different classes. All images are colorful and of the size 32×32.

we repeat the previous steps to construct and evaluate five other labeled datasets on DSSL 22-4, each consisting of $8,000$ labeled examples. Finally, we use all of the available training dataset as both labeled and unlabeled data and evaluate DSSL 22-4 five times (Table 4.3). We also use one $4,000$ labeled dataset from CIFAR-10 to evaluate a large DSSL 28-10 model for one time. It achieves an error rate of 9.77% and is within 1.06% of the overall performance improvement on the medium DSSL 22-4. Moreover, we compare the performance of a fully supervised WRN 22-4 CNN model using training data consisting of $4,000$ labeled examples that we run one time with our DSSL 22-4. WRN 22-4 achieves error rate of 20.81% [1] compared to our DSSL that achieves an error rate of 10.83%.

Table 4.3: Test results on CIFAR-10 and CIFAR-100. DSSL 22-4 results show the mean and standard deviation of the classification error rate (mean $\pm$ std.). We evaluate DSSL 28-10 once.

Method	**CIFAR-10**			**CIFAR-100**	
	4,000 labels	**8,000 labels**	**All labels**	**10,000 labels**	**All labels**
Γ-model [59]	20.40 ± 0.47				
Conv-CatGAN [70]	19.58 ± 0.58		9.38		
Improved GAN [65]	18.63 ± 2.32	17.72 ± 1.82			
Ensemble GAN [65]	15.59 ± 0.47	14.87 ± 0.89			
Π - model [42]	12.36 ± 0.31		5.56 ± 0.10	39.19 $\pm$ 0.36	26.32 $\pm$ 0.04
TE [42]	12.16 ± 0.31		5.60 ± 0.10	38.65 $\pm$ 0.51	26.30 $\pm$ 0.15
WAC [77]	12.31 ± 0.28		5.56 ± 0.03		
Sajjadi et al. [64]	11.29 ± 0.24				21.43 $\pm$ 0.16
DSSL 22-4	**10.83 $\pm$ 0.4**	**8.73 $\pm$ 0.24**	4.56 $\pm$ 0.08	**33.08 $\pm$ 0.23**	22.4 $\pm$ 0.24
DSSL 28-10	**9.77**			**30.26**	**18.33**

Table 4.4: The effect of augmentation and dropout on DSSL 22-4 on top of 4K labeled training examples from the CIFAR-10 dataset.

Dropout	Augmentation	Error Rate (%)
✓	✓	10.83%
	✓	11.4%
✓		16.96%
		24.49%

[1] We evaluate WRN 22-4 CNN using the training dataset consisting of $4,000$ labeled examples once. We run this experiment ourselves as the result is not available in the original paper.

Table 4.5: The mean and standard deviation (mean $\pm$ std.) of the classification error rates on STL-10.

Method	**1,000 labels**	**All labels**
Exemplar CNN [19]	27.2 $\pm$ 0.4	
Huang et al. [32]	23.2 $\pm$ 0.3	
CC-GAN [18]	22.21 $\pm$ 0.8	
SCI [31]		18.66 $\pm$ 0.1
DSSL 22-4	**19.88** $\pm$ **1.35**	**10.69** $\pm$ **0.09**

Additionally, we evaluate the effect of dropout and augmentation on DSSL 22-4 on one $4,000$ labeled CIFAR-10 dataset. We run every combination of enabling/disabling dropout and augmentation as an experiment with a dropout of 0.3. When we disable both dropout and augmentation, DSSL 22-4 achieves an error rate of 24.49%. The error rate decreases to 16.96%, when we enable dropout and disable augmentation. When we disable dropout while keeping augmentation the error rate decreases even further to 11.4%. Finally, DSSL 22-4 achieves the best error rate of 10.83%, when we enable both dropout and augmentation. From these results, we see that the augmentation has more effect than dropout on DSSL, but it benefits from having them both enabled.

Table 4.4 shows the effect of augmentation and dropout on DSSL on a dataset from CIFAR-10 dataset with 4,000 labeled examples. We run each experiment once.

4.4.2 CIFAR-100

Similar to CIFAR-10, CIFAR-100 is a collection of $60,000$ images with size 32×32 pixels. However, CIFAR-100 includes 100 classes compared to the 10 classes found in CIFAR-10. This creates the challenge of having a large number of classes with a smaller number labeled examples per class. In our experiments, we adopt the two publicly available parts of CIFAR-100, namely the $50,000$ and $10,000$ examples for training and testing,

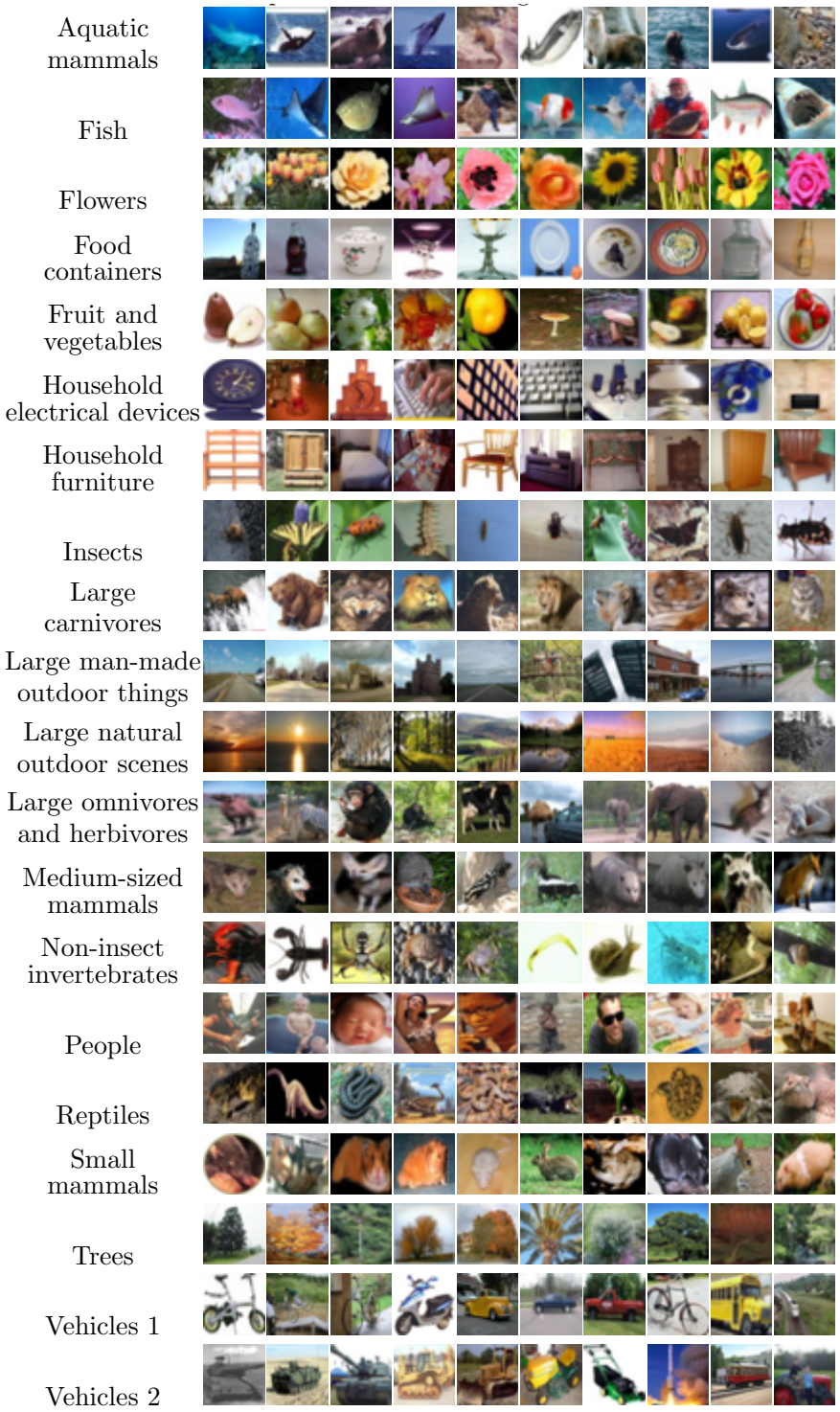


Figure 4.4: Sample from CIFAR-100 dataset. CIFAR-100 compiles images from 20 different coarse classes and that compiles 100 different fine classes. All images are colorful and of the size 32×32.

Table 4.6: CIFAR 100 dataset coarse and fine classes details.

Coarse classes	Fine classes
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 Electricals	Clock, Computer Keyboard, Lamp, Telephone, TV
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, Ocket, Streetcar, Tank, Tractor

respectively. We preprocess CIFAR-100 similar to CIFAR-10. We use the means of RGB channels 0.51, 0.49, and 0.44 for red, green and blue channels, respectively. Figure 4.4 shows a sample from CIFAR-10 dataset. Table 4.6 shows the coarse and fine classes of CIFAR-100.

We construct and evaluate DSSL 22-4 for five labeled training datasets. Each dataset is constructed by randomly selecting 100 examples per class (i.e., 20%) while keeping their actual labels. We then assign the rest of the training data to a random set of labels. Finally, we evaluate DSSL 22-4 on all available training examples as labeled and unlabeled data (Table 4.3).

We also evaluate the large DSSL 28-10 once on labeled training dataset consisting of $10,000$ training examples. The remaining training examples were treated as unlabeled. Table 4.3 shows a significant performance improvement when we use a larger model in

both cases.

4.4.3 STL-10

The STL-10 dataset includes 10 different classes with 500 and 800 colorful images per class for training and testing, respectively. STL-10 also includes $100,000$ unlabeled images that were selected from labeled examples on ImageNet. These images were extracted from a similar but broader distribution to the training labeled images. All images have the same size of 96×96 pixels. STL-10 is similar in spirit to CIFAR-10. However, STL-10 is more challenging because all STL-10 examples have higher resolution. Furthermore, STL-10 also includes a smaller number of labeled images per class compared to CIFAR-10. Finally, the unlabeled data comes from a wider distribution than the labeled data distribution. Figure 4.5 shows examples from STL-10 dataset.

Since the unlabeled dataset of the STL-10 was collected from a distribution different from the distribution of the labeled training and testing datasets. We preprocess the STL-10 training and testing dataset with one set of parameters different from the set of parameter that we use for the unlabeled data.

For the unlabeled data, we subtract the means of the GRB channels 112.35, 108.96, and 98.38 for the red, green and blue channels, respectively. Then, we divide images by standard deviations of channels 68.5, 66.62, 68.47 of red, green and blue channels, respectively.

Similar to the STL-10 unlabeled data, we preprocess the STL-10 labeled training and testing datasets by subtracting the means of the RGB channels 113.91, 112.15, and 103.70 for red, green and blue channels, respectively. Then, we divide by the standard deviations

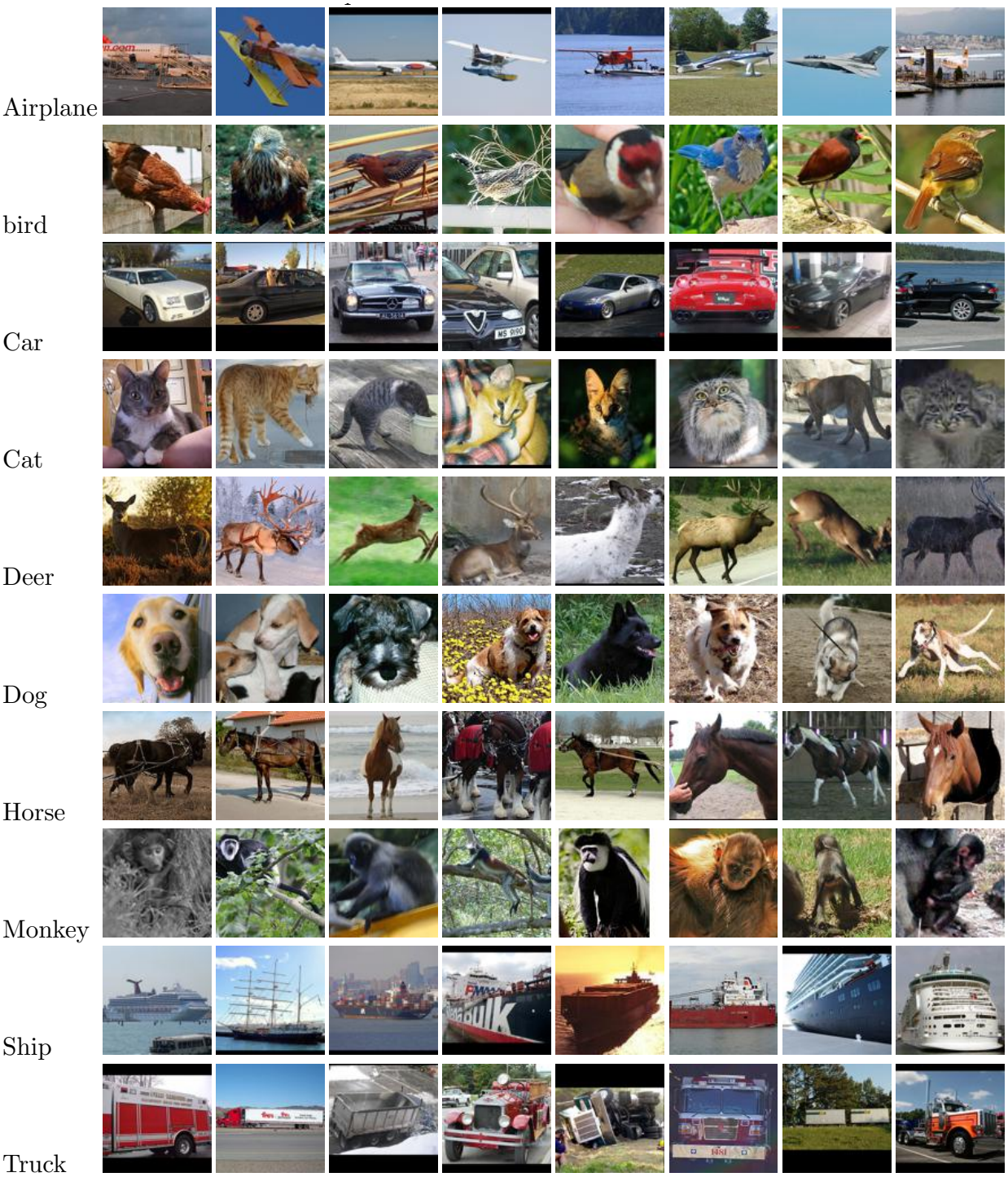


Figure 4.5: Sample from STL-10 dataset. STL-10 compiles images from 10 different classes. All images are colorful and of the size 96×96.

of the RGB channels of 66.39, 65.43, and 69.17 from the red, green and blue channels, respectively.

We evaluated DSSL 22-4 on five randomly selected training folds from the 10 folds provided with STL-10 dataset due to time limitations. Each fold includes 100 labeled training examples for each class. Given the increased image size, DSSL needs more training examples to avoid overfitting. Therefore, we replicate the labeled examples of each fold 10 times. We then randomly select $50,000$ examples from the unlabeled data for D_U. We do not use all available unlabeled data due to resource and time limitations. As a second experiment, we use all available $5,000$ training labeled examples as a D_L. D_U is as defined in the previous experiment. We report the performance of the aforementioned model on the $8,000$ labeled testing examples (Table 4.5).

4.4.4 MNIST

The MNIST dataset is a famous classification benchmark dataset of handwritten digits. It collects $70,000$ grayscale labeled images of size 28×28 pixels. We adopt the suggested parts of $60,000$ and $10,000$ images for training and testing, respectively. Figure 4.6 shows sample of MNIST dataset. We preprocess the MNIST datasets by dividing each image by 255 to make them in the range of [0,1].

We create five different labeled datasets from MNIST training dataset. For each dataset, we randomly select 5% labeled examples from the training dataset while using the rest as unlabeled. We follow the same steps mentioned before to construct five other datasets consisting of 10% labeled examples per class. We evaluate the performance of DSSL 22-4 on the datasets after we remove the `Tanh` layer from the UC. Finally,

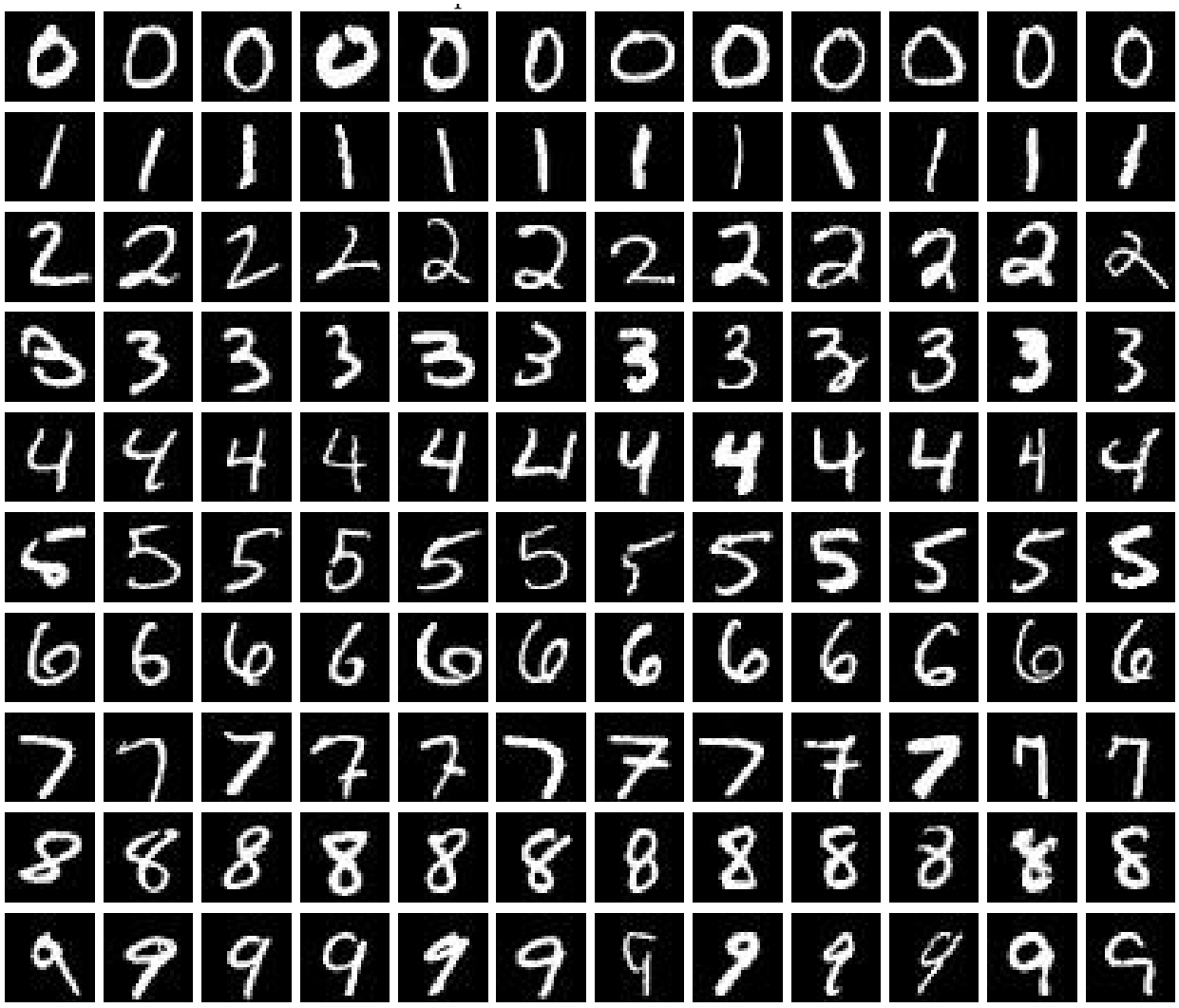


Figure 4.6: Sample from MINST dataset. MNIST dataset compiles images from 10 different classes. All images are gray scale and of the size 28×28.

we evaluate DSSL 22-4 five times on all available training dataset as labeled and unlabeled (Table 4.7).

4.4.5 SVHN

Table 4.7: The mean and standard deviation (mean $\pm$ std.) of the classification error rates on various labeled / unlabeled ratios from the training examples of MNIST.

Model	**5% labeled**	**10% labeled**	**All labels**
Conv-CatGAN [70]			0.48
Sajjadi et al. [64]			0.27 $\pm$ 0.02
Our DSSL 22-4	0.41$\pm$0.06	0.39$\pm$ 0.02	0.24 $\pm$ 0.01

The Street View House Numbers (SVHN) is a digit classification dataset that compiles $73,257$ primary training labeled images, $531,131$ extra labeled training images, and $26,032$ testing images. All images are colorful and are of size 32×32 pixels. The large variations in the images makes the SVHN dataset harder than MNIST to classify. We use the standard training images because it is the common in semi-supervised tasks [65, 42]. Figure 4.7 shows sample of SVHN dataset. We evaluate DSSL 22-4 after removing `Tanh` layer UC on various percentages of labeled/unlabeled data from SVHN. We preprocess the SVHN datasets by dividing each image by 255 to make them in the range of [0,1].

We construct five different datasets with 1% of labeled training examples. For each dataset, we randomly select 1% of the labeled examples and keep their labels and the rest of the training data as unlabeled. We repeat the previous steps with the proportions 5%, 10% and 20% of labeled examples, and create five datasets for each proportion. Additionally, we evaluated DSSL 22-4 on all available labeled training examples as labeled and unlabeled.

DSSL suffers from an overfitting problem for the 1% datasets because the number of

Figure 4.7: Sample from SVHN dataset. SVHN dataset compiles images from 10 different classes. All images are colorful and of the size 32×32.

labeled examples is very small and insufficient for training. Therefore, we replicate every 1% dataset 10 times. The mean of error rates of the 1% labeled dataset with 10 replicas is 4.56% (Fig. 4.8)

Table 4.8: The mean and standard deviation of the classification error rates (%) for evaluating the SVHN dataset with the DSSL 22-4 model (mean $\pm$ std.). We construct and test multiple labeled / unlabeled ratios from the SVHN dataset. We evaluate 1%, 5%, 10%, and 20% of the SVHN training dataset examples. Moreover, we evaluate on all available SVHN raining dataset as labeled and unlabeled. For each labeled / unlabeled ratio we create five different datasets. We evaluate all labels five times.

Model	1% labeled	5% labeled	10% labeled	20% labeled	All labeles
DSSL	**4.56 $\pm$ 0.4**	**3.72 $\pm$ 0.11**	**3.5 $\pm$ 0.08**	**3.14 $\pm$ 0.07**	**2.36 $\pm$ 0.08**

Hardware and Software

We adopt and modify the code released by [79] [2]. Then we constructed and ran all of this report experiments in `Torch 7` [16]. Moreover, the best pretrained models and code will be made available for public access.

We use our research lab server to conduct all reported results. The server is equipped with $3\times$ NVidia Tesla $K40$ GPUs, 256 GB of RAM, $4\times$ CPUs, with 16 cores a CPU. All of our experiments used a single GPU.

4.5 Discussion

The proposed DSSL method uses self-training, but alleviates the aforementioned self-training shortcomings by utilizing every training example (both labeled and unlabeled) during training phases. It adds every unlabeled example by predicting their labels without using any confidence thresholds. It also includes an unsupervised branch to fully exploit the information provided through both labeled and unlabeled data. The nature of DSSL

[2] The source code is publicly available on `https://github.com/szagoruyko/wide-residual-networks` (As of February 23, 2017)

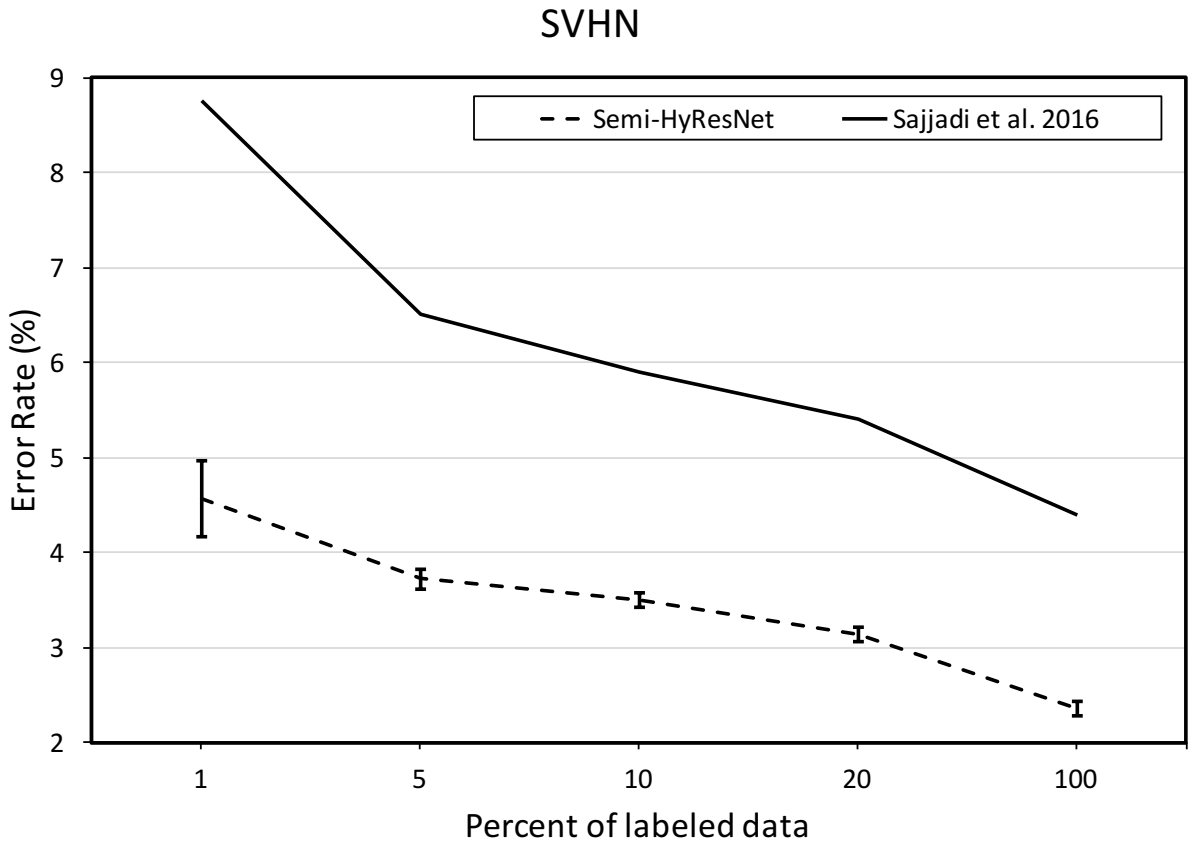


Figure 4.8: This chart compares the performance of DSSL 22-4 vs. Sajjadi et al. [64] on various ratios of labeled / unlabeled training datasets from the SVHN dataset. DSSL 22-4 results show the mean and standard deviation of classification error rates. Table 4.8 shows the details.

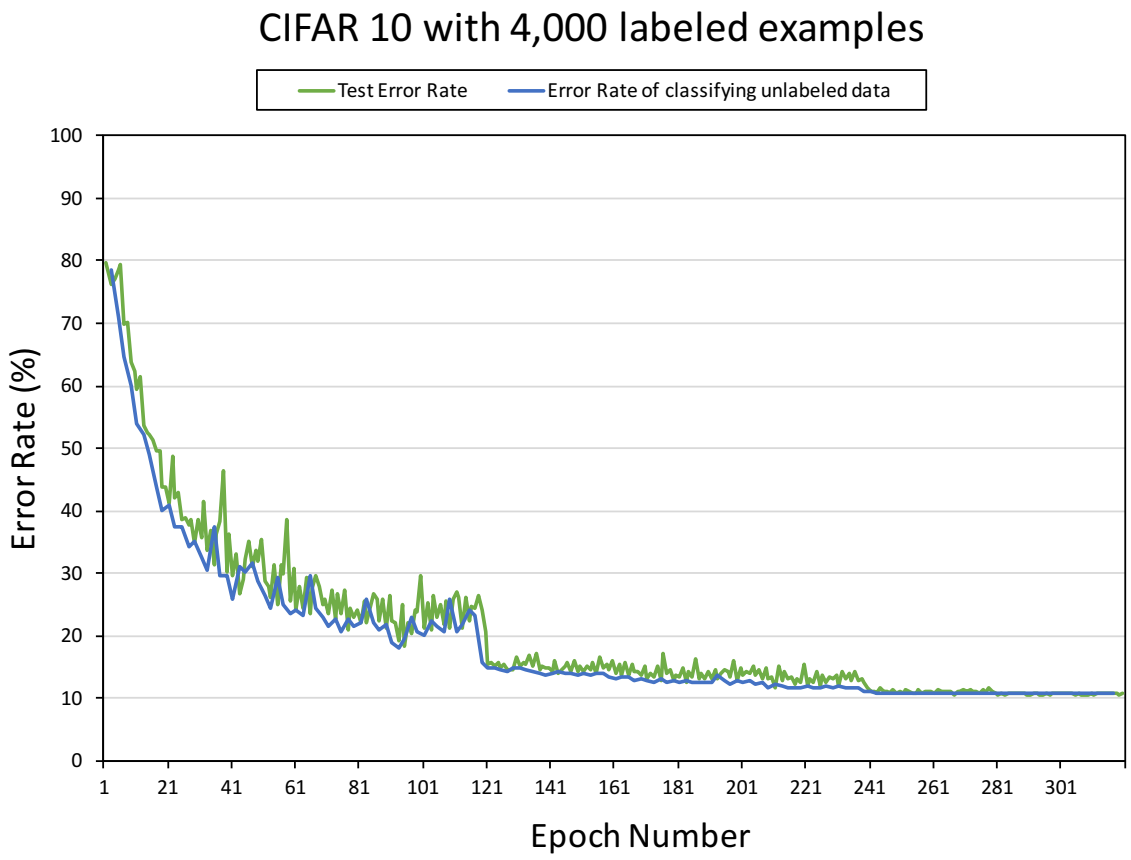


Figure 4.9: The performance of DSSL 22-4 on the $4,000$ labeled examples from the CIFAR-10 dataset. The figure compares the classification error rates of classifying the unlabeled data after every epoch for the 320 training epochs. The true labels of the unlabeled data are known and only used to create this plot. Moreover, for the same model, it shows the test classification error rates after every epoch.

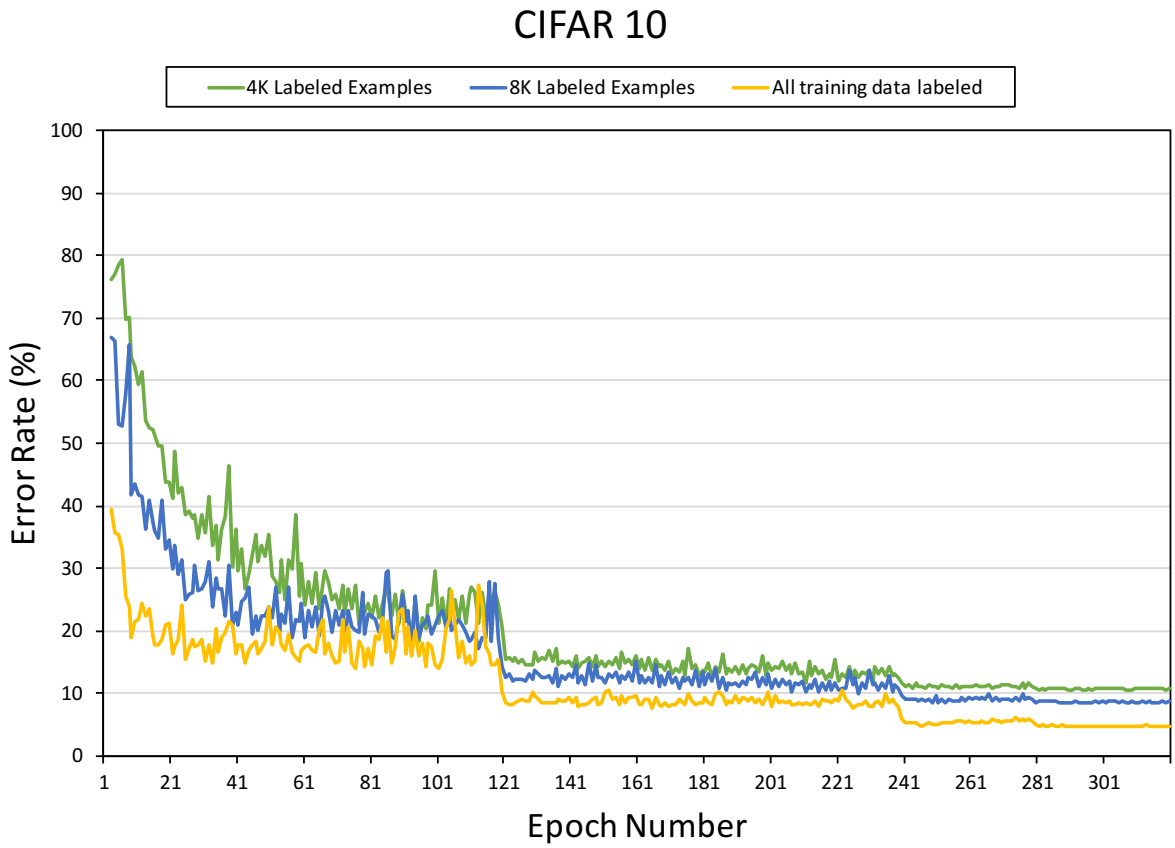


Figure 4.10: Compares the performance of DSSL 22-4 on three different ratios of labeled / unlabeled CIFAR-10 datasets. We show the results of evaluating the datasets consisting of 4K and 8K labeled examples. Moreover, all training data labeled shows the results of using all available training dataset as labeled and unlabeled. It shows the changes of the classification error rate for each dataset after every epoch for 320 epochs.

training prevents it from falling into the problem of poor prediction. Moreover, the DSSL architecture that includes the supervised and unsupervised components is one more factor that helps avoid this problem.

The unsupervised component in DSSL complements the supervised model in the absence of sufficiently large labeled training examples via the presence of large unlabeled examples. Additionally, DSSL benefits from unlabeled data by using it to train an unsupervised component that later increases the number of labeled examples per class used by the supervised branch. Furthermore, we note that the performance of DSSL improves as the number of labeled examples increases. This is consistent with other methods whose performance increase as the number of labeled examples increases such as [64].

CIFAR-10 results show that DSSL behaves similar to neural network methods where dropout and augmentation enhance the overall model performance. However, it does not rely on them to run. Moreover, augmentation influences DSSL more than dropout. Using both together improves the overall performance. Despite the fact that we use fairly medium sized models (e.g., DSSL 22-4) due to time and resources limits, we are able to set a new state-of-the-art record for most of the semi-supervised tasks demonstrated above. Moreover, the performance of DSSL can be improved even further by using larger models such as DSSL 40-10.

DSSL adds a new set of parameters compared to the original corresponding WRN models. For example, the models DSSL 22-4 and 28-10 add 19% and 14.8% of the overall parameters to the original WRN 22-4 and 28-10, respectively. The DSSL 22-4 and 28-10 models use $5.3M$ and $42.8M$ parameters, respectively.

The overall time complexity of a DSSL model is similar to the corresponding ResNet

model complexity except that the former runs a classifier every other epoch to classify the unlabeled data. For example, we train DSSL 22-4 on CIFAR-10 with $4K$ labeled examples for a total of 320 epochs. We classify the $46K$ unlabeled images 159 times. The average time of classifying all of the $46K$ images with a minibatch size of 100 is 30 seconds. Moreover, the time and number of basic operations that DSSL takes to train an epoch is not the same for all epochs. DSSL turns off the unsupervised branch in half of the epochs and evaluates only the labeled data. For example, evaluating DSSL on the $4K$ CIFAR-10 labeled dataset is very fast. In the second half of the epochs, only then DSSL runs on all available training dataset.

Fig. 4.9 shows the classification error rates of the unlabeled data classification during training across all epochs, as the true labels are known and used only to create this plot. We use the DSSL 22-4 model evaluated on a CIFAR-10 dataset with $4,000$ labeled examples. Moreover, for the same model, it shows the classification error rates on the testing data after every epoch. The DSSL trains for 320 epochs and classified the unlabeled data 159 times.

Fig. 4.10 shows the behavior of the DSSL model on CIFAR-10 dataset with multiple ratios of labeled / unlabeled examples.

4.6 Conclusion

In this chapter, we presented DSSL, a semi-supervised classification method that utilizes both supervised and unsupervised neural networks. DSSL uses a limited number of labeled training examples in conjunction with sufficiently large unlabeled examples to create a classification model. We empirically measured the performance of DSSL method on five benchmark datasets with various labeled / unlabeled ratios of training examples

Table 4.9: Summary of all DSSL results. The results show the mean and standard deviation of the classification error rates (%) (mean ± std.). The DSSL 28-10 results are indicated by ‡, where we evaluate it once on each dataset. All labels mean all available training data evaluated as labeled and unlabeled, which is evaluated five times except STL-10 is evaluated three times. The boldface results indicates new state-of-the-art record.

Dataset	**Labeled Data ratio**	**Error Rate**% (**mean** ± **std**.)
CIFAR-10	4K	**10.83**± **0.4** (**9.77**‡)
	8K	**8.73**±**0.24**
	All labeles	4.56 ±0.08
CIFAR-100	10K	**33.08** ± **0.23** (**30.26**‡)
	All labels	22.4±0.24 (**18.33**‡)
STL-10	1K	**19.88**± **1.35**
	All labels	**10.69** ± **0.09**
MNIST	5%	0.41±0.06
	10%	0.39±0.02
	All	0.24±0.01
SVHN	1%	**4.56**±**0.40**
	5%	**3.72**±**0.11**
	10%	**3.5**±**0.08**
	20%	**3.14**±**0.07**
	All labels	**2.36**±**0.08**

and then compared our results with state-of-the-art methods. The experiments show that DSSL set a new state-of-the-art record for CIFAR-10 with 4K and 8K labeled training examples, CIFAR-100 with 10K labeled training examples and STL-10 with 1K labeled training examples with mean of error rates of 9.77%, 8.73%, 30.08% and 19.96%, respectively. Furthermore, DSSL attains the state-of-the-art performance on the SVHN dataset using various ratios of labeled / unlabeled training examples. On the MNIST dataset, DSSL is competitive with other state-of-the-art methods. Table 4.9 shows a summary of all DSSL results on various semi-supervised tasks.

CHAPTER 5 TEACHER/STUDENT DEEP SEMI-SUPERVISED LEARNING FOR NOISY LABELS

5.1 Introduction

Supervised deep learning methods achieve excellent performance tackling a wide range of machine learning problems. In particular, convolutional neural networks (CNNs) attain state-of-the-art performance for applications such as including automated speech recognition [43, 3, 17], text classification [67, 80], object recognition [27, 25], spam detection, face recognition and verification [2, 4], and image segmentation [9, 5]. One major limitation, however, is the requirement for a sufficiently large number of labeled data [11] (e.g., ImageNet [62]). Furthermore, noisy label training datasets impair the performance of CNNs.

Unfortunately, curating a sufficiently large labeled dataset (e.g., ImageNet [62], COCO [47]) with a minimal number of noisy labels is not a simple task. Labeling the dataset in many domains is objective (e.g., texture data) [12] and varies by the background and expertise of the labeling person. In addition, labeling dataset in many cases requires domain experience such as in medical datasets (e.g., ultrasound images, X-Ray images) which is hard to acquire. In fact, many of the data labeling tasks are done by hiring people remotely over the Internet using frameworks such as Amazon Mechanical Turk, where the dataset is labeled in most cases by unknown individuals with a diverse set of experiences and backgrounds.

Generally speaking, the quality of the dataset examples affect the quality of the produced labels. For instance, labeling small images where two different objects share similar properties may confuse labeling individuals. Alternatively, automated labeling methods

(e.g., clustering, search engines) are more practical and can assist in curating labeled data. Fortunately, automated labeling is more accessible, more practical compared to human labor, and cheaper. However, high-quality labels from automated labeling methods are not guaranteed.

The combination of the aforementioned labeling problems and limitations increases the probability of producing inaccurate and noisy labels. Although learning from noisy label dataset has been presented in various machine learning methods such as KNN, SVM, and logistic regressions [53, 20], there is no sufficient attention to this direction. Recently, a few semi-supervised learning (SSL) methods such as self-training (ST) exploited training dataset that is contaminated with a level of noisy labels, where the noisy part of the dataset is unknown [38, 74, 81].

Typically, ST methods are iterative algorithms. In one common variation, the algorithm starts initially by learning a model from a noise-free small labeled dataset. Then, the learned model is used to predict labels for a larger noisy label (or unlabeled) training dataset. Examples with high confidence are added to the labeled training dataset, and the model is retrained on the new labeled dataset. The algorithm repeats the previous steps until the end of training.

Standard ST methods suffer several shortcomings and limitations. They are susceptible to poor prediction, which is in turn deceives the model during training toward deteriorating the model's performance. A common strategy to alleviate this problem is to set a predefined confidence threshold. This threshold is used to determine whether to accept the predicted labels and add them to the labeled training data. However, deciding the optimal threshold requires great efforts and several experiments.

Further, the frequent change of labels during training ST methods results in more frequent changes in the learned model. These changes produce an unstable model that has fluctuating performance during training and leads to degradation in performance later during the advanced training steps. Moreover, a few proposed self-training solutions assume a prior knowledge of the noise distribution [74], which in most of the practical life applications and datasets is not typical as the true labels are unknown.

5.1.1 Problem Formulation

The objective of a typical supervised learning method is to learn a decision model $\mathcal{M}$ from n available labeled training examples. The training examples are denoted by $\mathcal{T}_n$, where n is the total number of different training examples in $\mathcal{T}$. We denote a training example (e.g., image) by $\mathcal{X}_i \in \mathbb{R}^d$, where $i \in \{1, \ldots, n\}$. Also, we denote the true class label of $\mathcal{X}_i$ by $\mathcal{Y}_i \in \{1, \ldots, \mathcal{C}\}$, where $\mathcal{C}$ is the total number of different classes in $\mathcal{T}_n$. In general, we represent the labeled training dataset by $\mathcal{T}_n = \{(\mathcal{X}_i, \mathcal{Y}_i)\}_{i=1}^{n}$.

We use *noisy class label* (*noisy label* for short) *example* to indicate an example with a flipped class label, i.e., a label flipped to a label different from the true class label. For instance, an example $\mathcal{X}_m$ with a true class label $\mathcal{Y}_m$ is considered a noisy label example if it appears in the dataset with a class label $\widehat{\mathcal{Y}}_m$, $\mathcal{Y}_m \neq \widehat{\mathcal{Y}}_m$, and $\widehat{\mathcal{Y}}_m \in \{1, \ldots \mathcal{C}\}$. Additionally, a dataset with a level of $\nabla\%$ noisy labels includes $\nabla\%$ of the total number of examples are noisy label examples, and the rest of the $(100 - \nabla)\%$ of the total number of examples are true label examples, where $\nabla \in [0, 100]$. Generally speaking, we denote a dataset with noisy labels by $\widetilde{\mathcal{T}}_n = \{(\mathcal{X}_i, \widehat{\mathcal{Y}}_i)\}_{i=1}^{n}$.

In this chapter, we add one more set of labels $\mathcal{Y}^{(p)} \in \{1, \ldots \mathcal{C}\}$ to the noisy label

dataset. $\mathcal{Y}^{(p)}$ is a set of predicted labels during training, where initially $\mathcal{Y}^{(p)} = \widehat{\mathcal{Y}}$. We denote the training noisy label dataset hereafter by $\widehat{\mathcal{T}}_n = \{(\mathcal{X}_i, \widehat{\mathcal{Y}}_i, \mathcal{Y}_i^{(p)})\}_{i=1}^n$.

5.1.2 Contribution

In this chapter, we propose a new self-training *teacher/student deep semi-supervised learning* (TS-DSSL) method to train deep learning methods on noisy label dataset. In particular, TS-DSSL uses the common class of semi-supervised learning (SSL) methods; namely, self-training [81] (ST). TS-DSSL integrates two classifiers into a single semi-supervised learning (SSL) model. The first is a teacher classifier that gains knowledge during training. Then, it uses the knowledge to cleanse the noisy labels. The second is a student classifier that learns from the teacher during the training. The teacher and the student in the TS-DSSL model complement each other. The teacher classifier educates the student classifier, stabilizes the overall model, and cleans the noisy label in the training dataset. The student classifier exploits the knowledge gained by the teacher to enhance the overall model. Additionally, we propose a training procedure for TS-DSSL that overcomes the aforementioned shortcomings of ST. TS-DSSL sets state-of-the-art record on benchmark datasets with various levels of uniform and non-uniform noisy label training datasets.

The novelty of TS-DSSL originates from the following facts. First, we presume that TS-DSSL has access to only noisy labeled dataset to learn a decision model, and we do not assume the availability of any clean labels. Second, we assume that TS-DSSL does not have access to how much is the fraction of the noisy labels in the training dataset. Third, TS-DSSL does not have access to any information about which part of the training dataset

has the class labels and which one has the noisy labels. Fourth, it does not assume any prior information about the distribution of the noisy labels. In fact, it accepts any noisy label dataset regardless of the noisy labels information. Finally, TS-DSSL does not use any confidence thresholds as it uses all examples in the noisy label training dataset in each epoch of the training.

Algorithm 3 The algorithm for training TS-DSSL

Require: $\widehat{\mathcal{T}}_n = \{(\mathcal{X}_i, \widehat{\mathcal{Y}}_i, \mathcal{Y}_i^{(p)})\}_{i=1}^{n}$; $\widehat{\mathcal{T}}_n$ = training dataset consists of n training examples $\mathcal{X}$ associated with noisy labels $\widehat{\mathcal{Y}}$, $\mathcal{Y}^{(p)}$ = predicted labels of $\mathcal{X}$ classified during training. Initially $\mathcal{Y}^{(p)} = \widehat{\mathcal{Y}}$

Require: $\mathcal{G}$, $Teacher_{\mathcal{F}}$ and $Student_{\mathcal{F}}$ = convolutional layers, teacher branch, and student branch, respectively.

Require: $Teacher_{loss}$ and $Student_{loss}$ = loss functions for the teacher and the student, respectively.

Require: α=determines the contribution of teacher/student branches, $\alpha \in [0, 1]$.

Require: γ=determines the epoch when α and $(1-\alpha)$ will switch between the teacher and the student, respectively.

Require: β=determines the frequency of classifying the training dataset, and then updating $\mathcal{Y}^{(p)}$

Require: δ=determines the training dataset is first classified.

Require: $\mathcal{S}$=determines when to stop classification, $\mathcal{S} <$*max_ no_ epochs*.

$\mathcal{Y}^{(p)} \leftarrow \widehat{\mathcal{Y}}$

for *epoch_ no* $\in$ [1, *max_ no_ epochs*] **do**

  for *each minibatch* $b \in \widehat{\widehat{\mathcal{T}}}_n$ **do**

    $w_b \leftarrow \mathcal{G}($ *Augmentation* $(b))$

    $w_1 \leftarrow Teacher_{\mathcal{F}}(w_b)$

    $Teacher_{loss}\ (w_1, \widehat{\mathcal{Y}}) \leftarrow -\sum_j log\left(\frac{e^{w_{1_j}[\widehat{\mathcal{Y}}_j]}}{\sum_i e^{w_{1_j}[i]}}\right)$

    $w_2 \leftarrow Student_{\mathcal{F}}(w_b)$

    $Student_{loss}\ (w_2, \mathcal{Y}^{(p)}) \leftarrow -\sum_j log\left(\frac{e^{w_{2_j}[\mathcal{Y}_j^{(p)}]}}{\sum_i e^{w_{2_j}[i]}}\right)$

    if (*epoch_ no* $== \gamma$) **then** $\alpha = 1 - \alpha$ **end if**

    Update the weights using SGD with momentum, and $(\alpha *$ *Teacher*$_{loss})$ *and* $((1-\alpha) *$ *Student*$_{loss})$

  end for

  if $(epoch\_no\%\beta == 0)$ **AND** $(epoch\_no < \mathcal{S})$ **AND** $(epoch\_no \geq \delta$ **then**

    $\mathcal{Y}^{(p)} \leftarrow$ *Classify* $\mathcal{X}$ *with the model constructed from the sequence of* $\{\mathcal{G}$, *Teacher*$_{\mathcal{F}}$, *and Teacher*$_{loss}\}$

  end if

end for

5.2 Method

5.2.1 TS-DSSL Architecture

TS-DSSL is a single model that starts with a sequence of convolutional layers (Fig. 5.1). Then, the model forks into two branches. The two branches are the teacher and the student, respectively. Each branch is a classifier that is constructed from a pooling layer followed by a set of linear layers, and ends with a supervised criterion (e.g., cross entropy.) Note that both classifiers are constructed from an identical sequence of layers. We employ a supervised loss function that combines the negative log likelihoods of softmax. The supervised loss function is described by:

$$\mathrm{CE}_{loss}(w, \mathcal{Y}) = -\sum_{j} log\left(\frac{e^{w_j[\mathcal{Y}_j]}}{\sum_i e^{w_j[i]}}\right). \tag{5.1}$$

where j is the example number in a minibatch b, w_j is the output of the sequence of convolutional layers of the network consisting of a vector with a score for each class, and $\mathcal{Y}_j$ is the index of the target class of the example j.

TS-DSSL can be constructed from any learning method components (e.g., ResNet [27] and GoogLeNet [75]). In this chapter, we utilize the state-of-the-art wide residual network (WRN) [79] to construct the sequence of convolutional residual blocks (CRBs.) The size of a WRN model is represented by the number of convolutional layers (i.e., the network depth) followed by the widening factor (i.e., number of feature maps). For example, Fig. 5.1 shows a sequence of CRBs used from *WRN 10-2* to construct the TS-DSSL. The sequence includes 10 convolutional layers and a width of 2. In general, we represent a

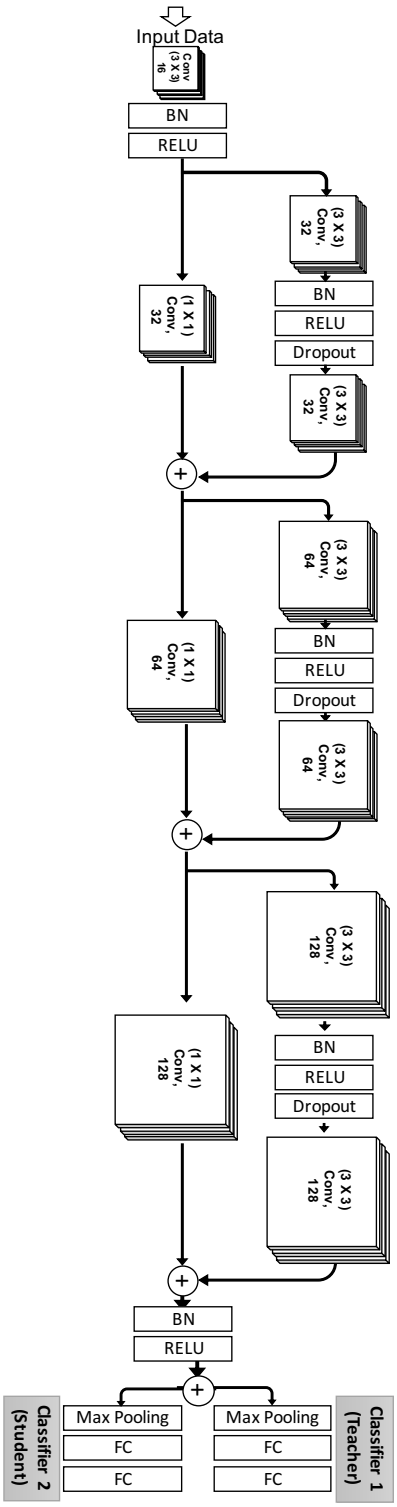


Figure 5.1: The architecture of TS-DSSL. It is composed of three branches. The first branch is a set of residual blocks that forks at the end into two supervised branches. Each branch compiles a sequence of max pooling, followed by two linear layers and finally a supervised learning criterion.

convolutional residual block by:

$$\mathcal{O}_t = \Gamma(\mathrm{W}_{\mathrm{shortcut}}, \mathcal{O}_{t-1}) + \Lambda(\mathrm{V}_{\mathrm{resid\_branch}}, \mathcal{O}_{t-1}). \tag{5.2}$$

where $\mathcal{O}_t$ is the output of the residual block t that is used as an input to the next component, $\mathcal{O}_{t-1}$ is the input to the residual block t, Γ is the shortcut branch function, $\mathrm{W}_{\mathrm{shortcut}}$ are the shortcut branch parameters (weights), Λ is the residual branch function, and $\mathrm{W}_{\mathrm{resid\_branch}}$ are the residual branch parameters (weights). In all of the experiments reported in this paper, we use a fairly medium sized WRN to construct our TS-DSSL. Basically, we construct the sequence of convolutional layers in all models from a WRN 10-2.

5.2.2 TS-DSSL Training

TS-DSSL training procedure aims to create a stable model. Moreover, it aims to reduce the computational burden by finding the optimal predication frequency and the best time to prune the prediction. The teacher and the student are trained simultaneously using the same training examples but with different sets of labels. In particular, the teacher training is limited to only the initially given noisy labels $\widehat{\mathcal{Y}}$, which never changes. On the other hand, the student is trained with the predicted labels $\mathcal{Y}^{(p)}$ that changes frequently during the training.

TS-DSSL accepts as input a noisy label training dataset $\widehat{\mathcal{T}}_n$. TS-DSSL starts initially by setting $\mathcal{Y}^{(p)} = \widehat{\mathcal{Y}}$. Training a TS-DSSL (Algorithm 3) starts the first epoch ($t = 1$) by training the model $\mathcal{M}$ with $\widehat{\mathcal{T}}_n$. The model is trained for ($t = \delta$) epochs with the same input. Then, at the end of δ^{th} epoch the model $\mathcal{M}_\delta$ is saved to be used in the next step. Next,

TS-DSSL uses the teacher of $\mathcal{M}_\delta$ to classify the training dataset $\widehat{\mathcal{T}}$ and then saves the predicted labels in $\mathcal{Y}^{(p)}$. After that, TS-DSSL repeats the previous steps but re-classify after every β epochs. After $(t = \mathcal{S})$ epochs, the model prunes the classification step and continues the training without the classification step until the maximum number of training epochs.

TS-DSSL exploits the stability and the knowledge of the teacher during training to clean the noisy labels of the training dataset. The student thereafter utilizes the cleansed labels to learn in the next epoch. In practice, the classification step in TS-DSSL can be performed in parallel for all training examples. Therefore, it can be fairly quick.

In each epoch of the training, the feedforward feeds the data into the sequence of convolutional layers. Then, the output is passed into the fork that passes a copy to each classifier. After that, the backpropagated gradients are passed back from the classifiers to the fork that sums gradients up then backpropagates it to the sequence of convolutional layers.

The performance of the teacher and student varies during training. In the early steps of the training, the teacher is more stable and has better performance than the student. Toward the end of training, the student is more stable and has better performance. TS-DSSL gives a weight $\alpha \in [0, 1]$ to the contribution of the teacher and a weight $(1 - \alpha)$ to the contribution of the student.

5.2.3 Synthesizing the Noisy Labels

We synthesize noisy label datasets from the benchmark datasets CIFAR10 [40] and MNIST [44]. The synthesized datasets simulate noisy labels in real-life practical applica-

tions. It is worthwhile to mention that we leave the labels of the test split of each dataset intact. We synthesize the noisy label training datasets using a probabilistic model of label noise. Initially, we presume that for any pair of examples $\mathcal{X}_t$ and $\mathcal{X}_z$ with the true labels $\mathcal{Y}_t$ and $\mathcal{Y}_z$, the noisy labels $\widehat{\mathcal{Y}}_t$ and $\widehat{\mathcal{Y}}_z$ are independent of each other. Additionally, we assume $p(\widehat{\mathcal{Y}}_t|\mathcal{Y}_t) = p(\widehat{\mathcal{Y}}_z|\mathcal{Y}_z)$ both have identical distribution. Moreover, we assume that each noisy label in our model (i.e., the change of the true label) $\widehat{\mathcal{Y}}$ is independent from the example itself $\mathcal{X}$ and depends only on the true label of the example $\mathcal{Y}$. We represent the probabilistic noisy label model Φ by a $\mathcal{C} \times \mathcal{C}$ probability transition matrix $\Phi \in \mathbb{R}^{\mathcal{C}\times\mathcal{C}}$. We define the noisy label $\widehat{\mathcal{Y}}$ distribution by:

$$p(\widehat{\mathcal{Y}} = z|\mathcal{Y} = t) = \phi_{z,t}. \tag{5.3}$$

where z, $t \in \{1, \ldots \mathcal{C}\}$, and $\phi_{z,t}$ is the element (z, t) in Φ.

TS-DSSL is independent of the noise distribution and does not use any information about the label noise distribution. Because TS-DSSL works for any label noise distribution Φ (Eq. 5.3), we use the noise model described by Eq. 5.3 to simulate and create two noise scenarios (close to practical life noisy labels) with various $\nabla\%$ levels of noisy labels ranging from $25\% - 80\%$ of the total number of training examples (Fig. 5.2). First, is a *non-uniform label noise* scenario denoted by

$$\Phi_{\text{non-uniform}} = (1 - \nabla)\mathbf{I} + \nabla\mathbf{U}. \tag{5.4}$$

where **I** is the identity matrix, ∇ is the noise level, and **U** is a matrix where all columns are

uniformly selected from a set of vectors with non-negative values that sum to one (i.e., unit simplex). Second, we use a *uniform label noise* scenario denoted by

$$\Phi_{\text{uniform}} = (1 - \nabla)\mathbf{I} + \frac{1}{\mathcal{C}}\nabla\mathbf{J}. \tag{5.5}$$

where **J** is a matrix where all elements are ones.

5.2.4 Baseline Methods

Although state-of-the-art methods used the same standard datasets we use to synthesize noisy label training datasets, their methods were evaluated on small models. Therefore, their results are incomparable to the results of TS-DSSL. We compare the performance of TS-DSSL with three baseline methods that have a similar architecture to TDS-DSSL. We implement, train, and evaluate the three baseline methods with consistent settings and parameters with TS-DSSL to make a fair comparison.

The three baselines methods are

1. *Standard WRN baseline method*. It is a standard WRN model trained on the noisy label dataset.

2. *Self-cleansing I*. It is a standard WRN with self-training. In this method, we first train the model for one epoch. Second, we classify and update the labels of the training dataset using the model produced from the previous steps. Third, we resume the training for the next epoch using the predicted labels from the previous step. We repeat the previous steps until the maximum number of epochs.

3. *Self-cleansing II*. A self-training method in which we train a WRN until the maximum number of epochs and save the final model. Then, we use the final model to classify and label the training dataset. After that, we use the new labels to train a new model. We repeat the previous steps three times and report the results.

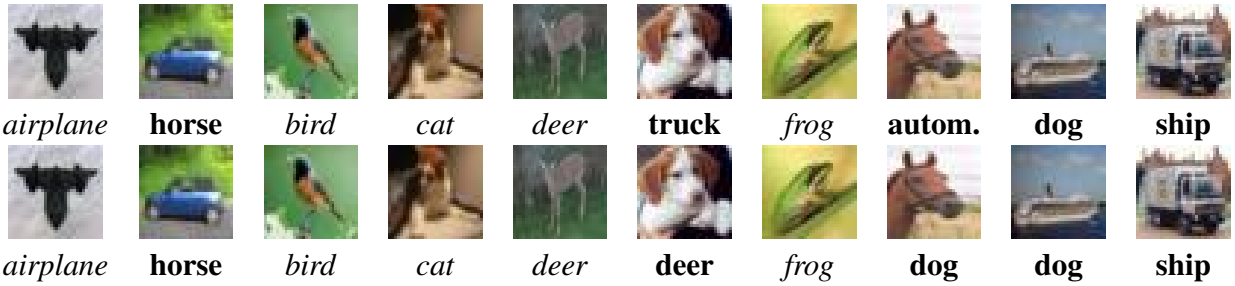


Figure 5.2: An example of a 50% noisy label data taken from CIFAR10. The first row shows a subset of 50% *uniform* noisy label dataset. The second row shows a subset of 50% *non-uniform* noisy label dataset. The labels in **boldface** are noisy labels, and the labels in *italic* are true labels.

5.3 Experimental Results

In this section, we investigate the performance of TS-DSSL on the datasets CIFAR10 and MNIST. All models in this chapter are trained from scratch. We generate a noisy label dataset from uniform and non-uniform noise distributions for each level of noise. We evaluate every dataset three times then we report the *mean* and *standard deviation* (mean$\pm$std.) of the classification error rates. The state-of-the-art results are provided in the tables using a boldface font.

5.3.1 CIFAR-10

CIFAR-10 dataset compiles $60,000$ colorful RGB images of size 32×32 pixels each image. The dataset is divided evenly into 10 different classes (airplane, automobile, bird, car, deer, dog, frog, horse, ship, and truck). Additionally, it is available in two predefined

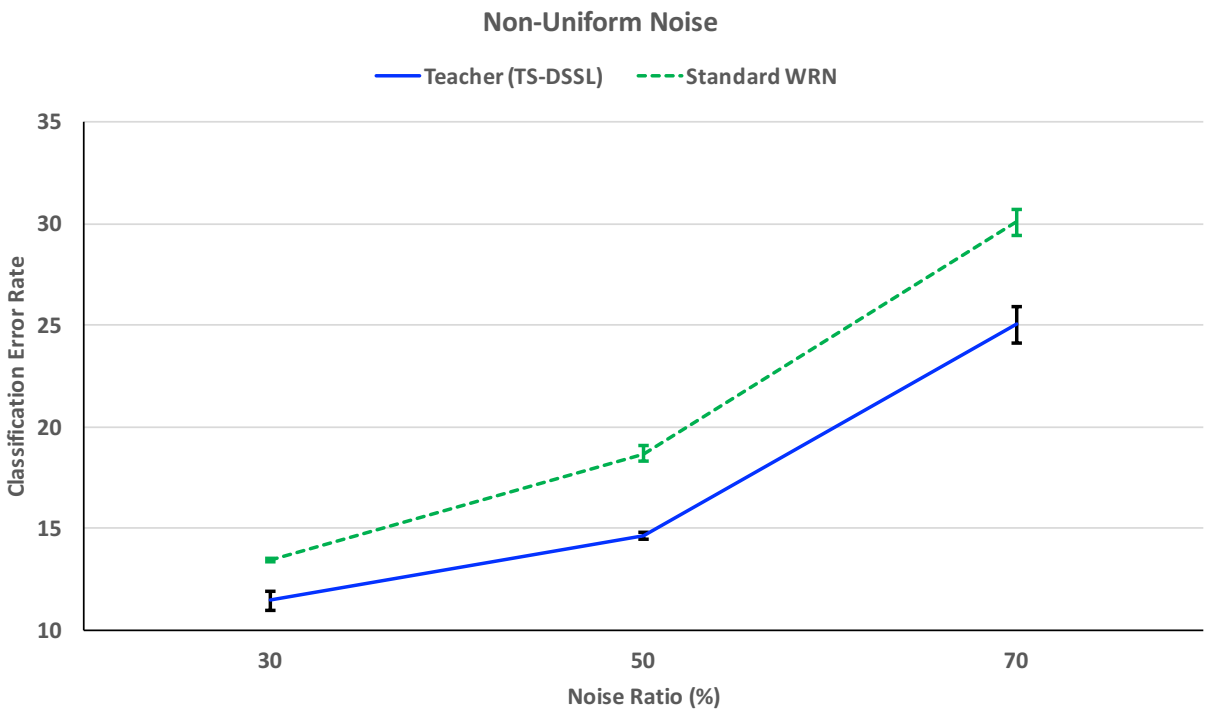
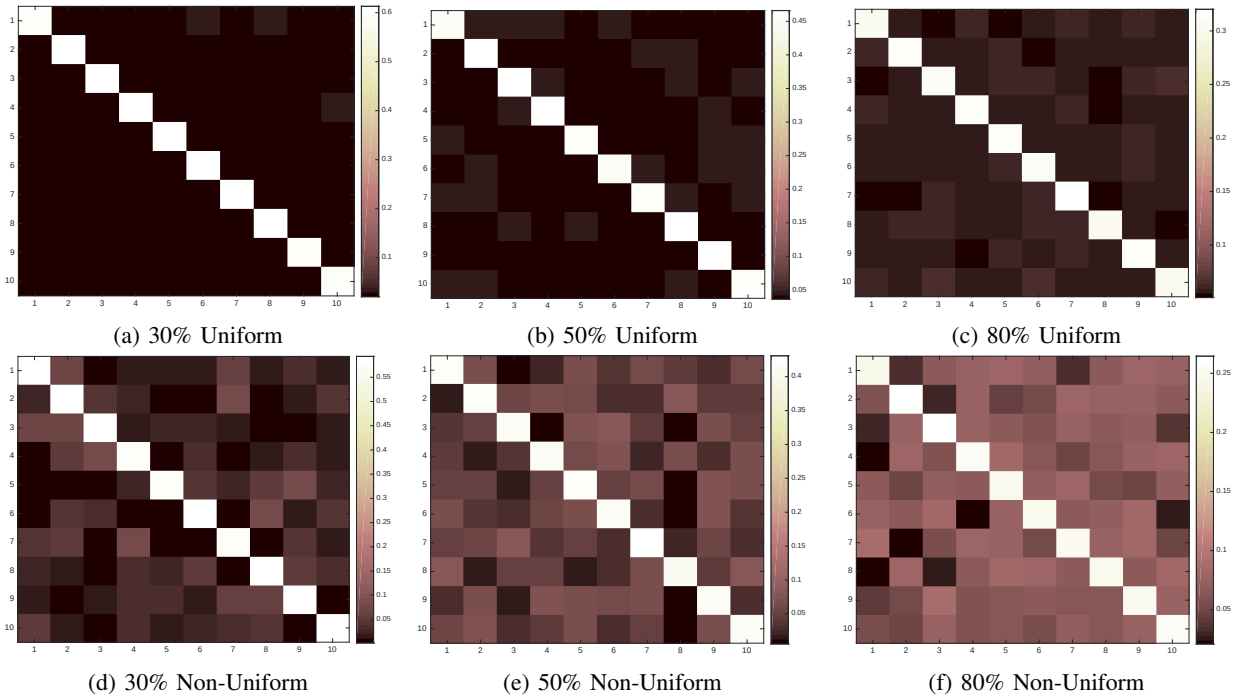


Figure 5.3: The (mean $\pm$ std.) of the classification error rates of TS-DSSL and WRN on various non-uniform levels of noise on CIFAR10. The results are measured on the standard test split provided with CIFAR10.

portions that we use hereafter in our experiments. These splits comprise $50,000$ and $10,000$ examples used for training and testing, respectively.

We evaluate TS-DSSL on two main types of noisy label training data synthesized from CIFAR10 (Fig. 5.4). First, we construct six different uniform noisy label datasets where the noise levels span $\nabla \in \{0.25, 0.3, 0.5, 0.7, 0.75\}$. Second,we synthesize three non-uniform noisy label datasets (Fig. 5.3) where the noise levels span $\nabla \in \{0.3, 0.5, 0.7\}$.



(a) 30% Uniform (b) 50% Uniform (c) 80% Uniform

(d) 30% Non-Uniform (e) 50% Non-Uniform (f) 80% Non-Uniform

Figure 5.4: The confusion matrices of various uniform/non-uniform noise levels from CIFAR10 training data.

Additionally, we use a noisy label dataset with 50% uniform noise from CIFAR10 to evaluate the following parameters in TS-DSSL. First, we investigate the optimal starting epoch of the classification step δ in TS-DSSL. We evaluate the values of $\delta \in \{1, 3, 5, 7, 14, 21\}$ epochs (Table 5.10). We also evaluate TS-DSSL when $\alpha \in \{0.95, 0.75, 0.65, 0.5, 0.35, 0.25, 0.05\}$ (Table 5.2). Moreover, we evaluate TS-DSSL when $\gamma \in \{20\%, 50\%, 55\%, 80\%, 100\%\}$ (Ta-

ble 5.3). Further, we experiment TS-DSSL with different frequency of classification β that determins haw many epochs pass before the self-cleaning step is performed to update $\mathcal{Y}^{(p)}$. We test $\beta \in \{3, 5, 7, 14, 21\}$ (Table 5.10). Additionally, we study the optimal prune of the classification step $\mathcal{S}$ after various percentages of the total number of training epochs $\mathcal{S} \in \{20\%, 50\%, 80\%, 100\%\}$ (Table 5.1). Finally, we evaluate TS-DSSL on various rates of dropout (Table 5.4).

TS-DSSL attains the best performance with a dropout of 30%, when the first classification starts after 7 epochs ($\beta = 7$) and repeats every 7 epoch ($\delta = 7$), prunes the classification at 80% of the total number of epochs $\mathcal{S} = 80\%$, the contributions of the teacher and the student at the first 80% of the total number of epochs are 0.65 ($\alpha = 0.65$) and 0.35, respectively. Additionally, when the contributions of the teacher and the students switches at 70% of the total number of epochs ($\gamma = 70\%$) to 0.35 ($\alpha = 0.35$) and 0.65, respectively.

Table 5.1: The (mean $\pm$ std.) of the classification error rates of TS-DSSL on a level of 50% uniform noise of CIFAR10 when we stop cleansing (predicting labels) of the training data after epoch $\mathcal{S}$.

Stop at epoch $\mathcal{S}$	**Teacher**	**Student**
20%	22.39±0.30	18.06±0.04
50%	16.29±0.53	15.83±0.27
55%	15.62±0.24	15.65±0.19
80%	**14.54±0.34**	15.37±0.33
No stop	14.82 ± 0.17	15.48±0.21

5.3.2 MNIST

MNIST dataset is a famous classification benchmark dataset of handwritten digits. It collects $70,000$ grayscale labeled images of size 28×28 pixels. We adopt the suggested splits of $60,000$ and $10,000$ images for training and testing, respectively. We synthesize

Table 5.2: The (mean $\pm$ std.) of the classification error rates of TS-DSSL on CIFAR-10 with 50% **uniform noise**. All results show (mean $\pm$ std.) for three runs. $\alpha \in [0, 1]$ is the weight assigned to the teacher's contribution and $(1 - \alpha)$ is the weight of the student's classifier contribution.

α	$1-\alpha$	**Teacher**	**Student**
0.95	**0.05**	18.22±0.13	17.53±0.09
0.75	**0.25**	14.49±0.16	15.35±0.16
0.65	**0.35**	**14.01±0.1**	15.05±0.23
0.50	**0.50**	16.76±0.68	17.16±0.66
0.35	**0.65**	17.53±0.27	17.34±0.16
0.25	**0.75**	21.03±0.34	20.81±0.44
0.05	**0.95**	39.93±2.58	39.51±2.52

Table 5.3: The (mean $\pm$ std.) of the classification error rates of TS-DSSL when we switch the values of α and $(1 - \alpha)$ after epoch γ, between the teacher and the student branches, respectively. The models evaluated on CIFAR10 with a level of 50% uniform noise, $\alpha = 75\%$, and γ is the percent of the total number of training epochs.

Switch after γ epoch	**Teacher**	**Student**
20%	14.58±0.33	15.56±0.23
50%	15.11±0.24	15.90±0.24
55%	14.72±0.39	15.35±0.16
70%	**14.49±0.16**	15.44±0.06

Table 5.4: The (mean $\pm$ std.) of the classification error rated of TS-DSSL with various dropout ratios on CIFAR-10 with a of 50% uniform noise.

Dropout %	**Teacher**	**Student**
No dropout	18.25 ± 0.82	18.30 ± 0.94
30%	**14.78 ± 0.23**	15.59 ± 0.21
50%	16.13 ± 0.49	16.12 ± 0.68
70%	23.26 ± 0.37	22.36 ± 0.29

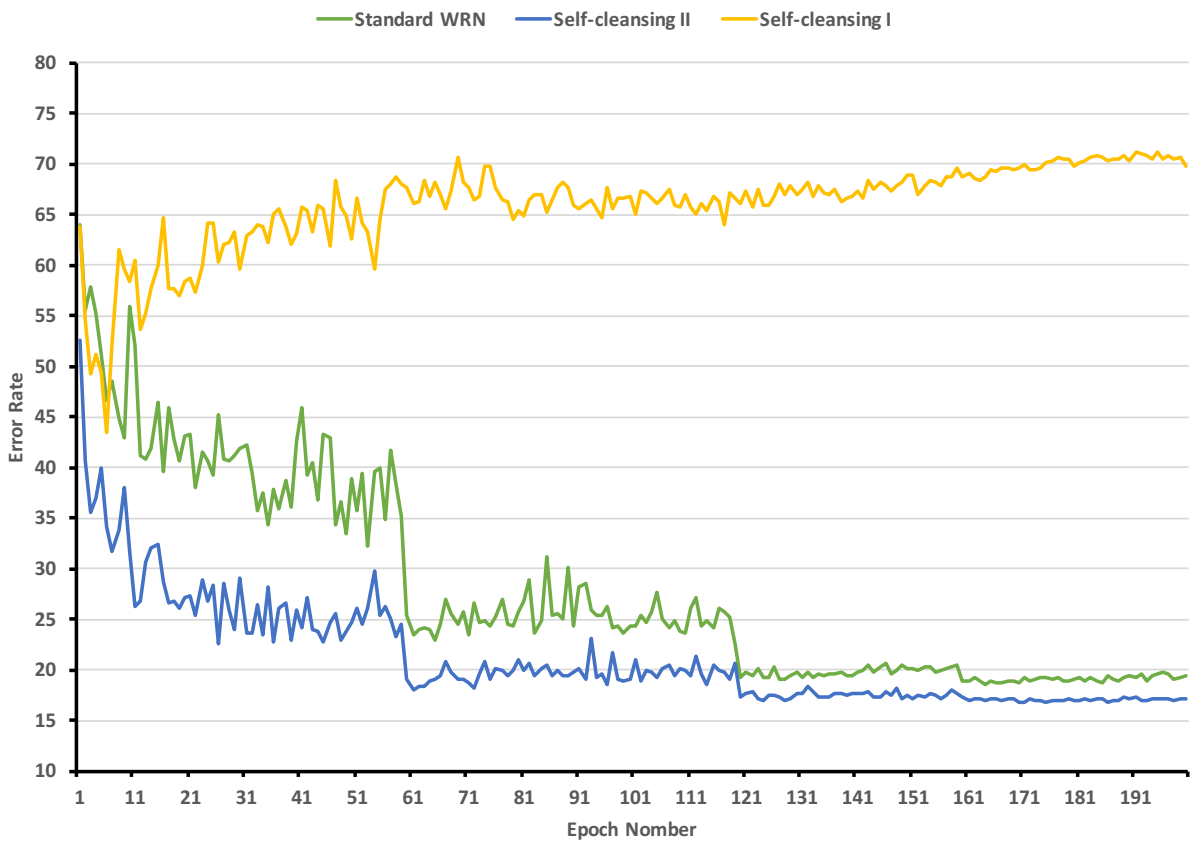
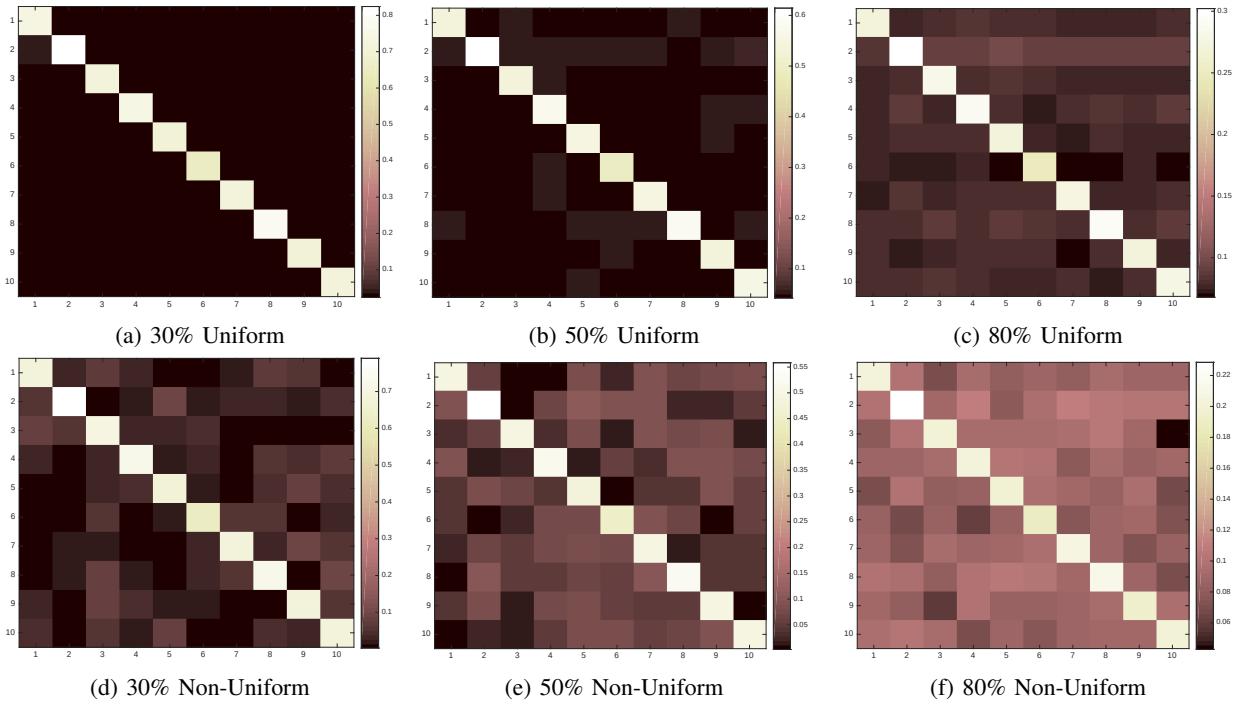


Figure 5.5: The error rates of three baseline models (*standard WRN*, *self-cleansing I*, and *self-cleansing II*) after each of 200 training epochs on CIFAR10 with a level of 50% uniform noise. The results are measured on the standard test split provided with CIFAR10.

uniform and non-uniform noisy label datasets from MNIST. For each noise distribution, we create six different noisy label datasets when noise levels of $\nabla \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ (Fig. 5.6). Table 5.5 and Table 5.6 show the evaluation results of TS-DSSL on various levels of uniform and non-uniform noisy label training datasets, respectively. Furthermore, we compare the performance of TS-DSSL with standard WRN on various levels of noise (Figures 5.8 and 5.7).



(a) 30% Uniform (b) 50% Uniform (c) 80% Uniform

(d) 30% Non-Uniform (e) 50% Non-Uniform (f) 80% Non-Uniform

Figure 5.6: The confusion matrices of various uniform/non-uniform noise levels from MNIST training data.

5.3.3 Data Preprocessing

The data preprocessing of the dataset is limited to the following steps. We normalize the MNIST by scaling the images in the range $[0, 1]$ by dividing them by 255. We preprocess CIFAR10 by dividing images by 255 to shift them to the range of $[0, 1]$. Then, we

Table 5.5: The (mean $\pm$ std.) of classification error rates on various uniform noise levels on MNIST. We also evaluated on MNIST with original labels (true labels), TS-DSSL teacher achieves (0.3 $\pm$ 0.02), student achieves (0.29 $\pm$ 0.02), and standard WRN achieves (0.3 $\pm$ 0.02).

Noise %	Teacher	Student	Standard CNN
30%	**0.57 $\pm$ 0.06**	0.62 $\pm$ 0.05	0.68 $\pm$ 0.04
40%	**0.54 $\pm$ 0.06**	0.61 $\pm$ 0.05	0.78 $\pm$ 0.01
50%	**0.74 $\pm$ 0.02**	0.82 $\pm$ 0.03	0.89 $\pm$ 0.03
60%	**0.81 $\pm$ 0.06**	0.89 $\pm$ 0.06	1.11 $\pm$ 0.04
70%	**0.84 $\pm$ 0.06**	0.90 $\pm$ 0.08	1.43 $\pm$ 0.05
80%	**1.28 $\pm$ 0.11**	1.56 $\pm$ 0.11	2.00 $\pm$ 0.09

Table 5.6: The (mean $\pm$ std.) of classification error rates on various non-uniform noise levels on MNIST.

Noise %	Teacher	Student	Standard CNN
30%	**0.62 $\pm$ 0.06**	0.67 $\pm$ 0.05	0.75 $\pm$ 0.05
40%	**0.63 $\pm$ 0.05**	0.77 $\pm$ 0.06	0.75 $\pm$ 0.11
50%	**0.82 $\pm$ 0.00**	0.97 $\pm$ 0.06	1.02 $\pm$ 0.05
60%	**0.87 $\pm$ 0.02**	1.02 $\pm$ 0.11	1.26 $\pm$ 0.07
70%	**1.60 $\pm$ 0.03**	2.07 $\pm$ 0.09	1.73 $\pm$ 0.09
80%	**3.37 $\pm$ 0.25**	3.98 $\pm$ 0.14	4.69 $\pm$ 0.18

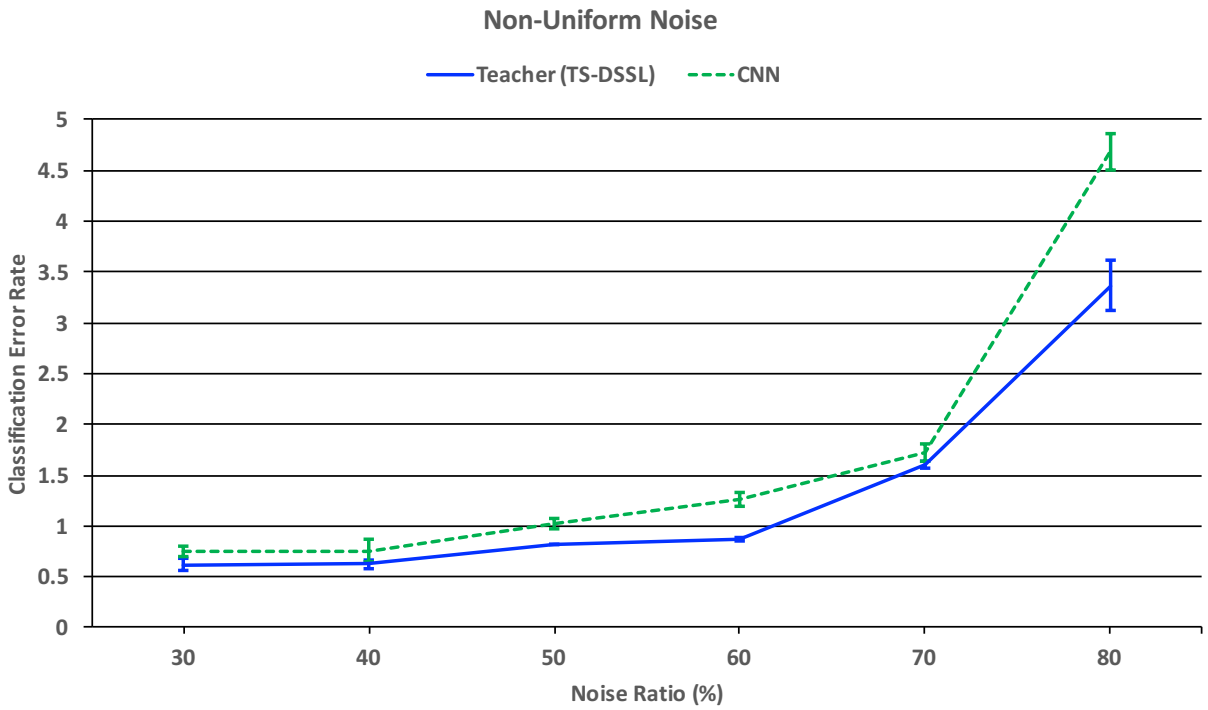


Figure 5.7: The (mean $\pm$ std.) of classification error rates on various levels of uniform noisy labels in MNIST. The results are measured on the standard test split provided with MNIST.

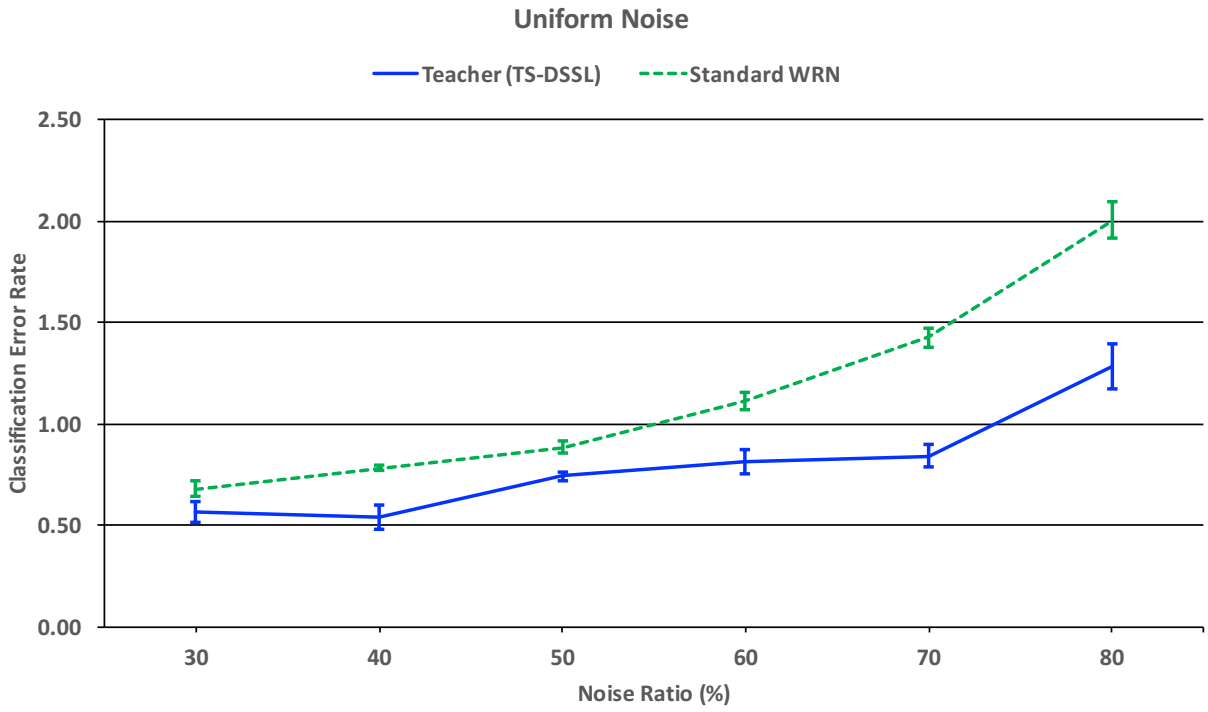


Figure 5.8: The (mean $\pm$ std.) of classification error rates on various uniform noise levels in MNIST. The results are measured on the standard test split provided with MNIST.

subtract the means of the RGB channels. We use the means 0.49, 0.48, and 0.45 for red, green and blue channels, respectively.

We do not apply aggressive augmentation. Instead, we only apply the standard augmentation presented by [45] on CIFAR-10. This includes random horizontal flips and random crops after padding each side of the image by 4 pixels. For the MNIST datasets, we only take random crops after padding each side of an image by 4 pixels each.

5.3.4 Experimental Setup

We study the TS-DSSL model with different levels of uniform and non-uniform levels of noisy labels. TS-DSSL state-of-the-art results are provided in the tables of results in boldface. In all of our experiments, we use a minibatch size of 128 images. We train TS-DSSL models using SGD with Nesterov momentum. TS-DSSL shows the best performance when we use dropout [30, 72] and batch normalization [36]. Furthermore, we employ the rectified linear units (ReLU) activation function [52]. Moreover, we turn off biases in all of our experiments. We initialize all layers with the method introduced by [21]. We run all TS-DSSL experiments for 350 epochs. The learning rate starts at 0.1 and then decreases after epochs 100, 200 and 250 at a rate of 0.2, and a dropout factor of 0.3. We train all of our models from scratch and do not fine-tune any model. We retain the given test partition of each dataset intact, and we use it to measure and report the performance of each method. We report the mean and standard deviation (mean $\pm$ std.) of the classification error rate for three runs for each model unless noted otherwise.

Unfortunately, the state-of-the-art WRN models use large number of parameters that require several days of continuous training. For example, the state-of-the-art WRN model [79]

with 40 convolutional layers and a width of 10 uses $56M$ parameters. Hence, we created our TS-DSSL models from a fairly medium sized WRNs so we are able to run more experiments and more detailed results.

5.3.5 Hardware and Software

We adopt and modify the code released by [79] [3]. Then, we construct, implement, and run all of the experiments in this chapter with `Torch 7` [16]. We use our research lab server to conduct all experiments. The server is equipped with $3\times$ NVidia Tesla $K40$ GPUs, 256 GB of RAM, $4\times$ CPUs, with 16 cores a CPU. All of our experiments use a single GPU.

Table 5.7: The (mean $\pm$ std.) of the classification error rates of TS-DSSL and three baseline methods on various **uniform noise** levels of CIFAR10.

Noise %	TS-DSSL Teacher	Standard WRN	Self-cleansing I (*baseline*)	Self-cleansing II (*baseline*)
original (true labels)	**7.61±0.06**	9.32±0.01		
25%	**11.11±0.15**	13.17±0.43		
30%	**11.61±0.17**	13.44±0.14	38.56±1.24	
50%	**14.63±0.18**	18.82±0.37	44.63±1.46	17.05±0.30
70%	**23.91±0.95**	31.16±0.10	59.12±2.98	
75%	**30.26±0.81**	35.72±0.61		

[3]. The source code is publicly available on `https://github.com/szagoruyko/wide-residual-networks` (As of June 10, 2018)

Table 5.8: The (mean $\pm$ std.) of the classification error rates of TS-DSSL and three baseline methods on various **non-uniform noise** levels of CIFAR10.

Noise %	Teacher	Student	Standard WRN	Self-learning I (*baseline*)
30%	**11.45±0.46**	12.26±0.33	13.44±0.10	38.29±1.11
50%	**14.65±0.18**	16.50±0.05	18.67±0.39	44.87±2.48
70%	**25.03±0.92**	26.82±0.55	30.08±0.63	59.66±3.21

Table 5.9: The (mean $\pm$ std.) of the classification error rates of the *self-cleansing I* (baseline) model on CIFAR10 with a uniform noise level of 50%. The results show various classification intervals and after what epoch the first classification was performed.

Interval (β)	$\delta = 1$	$\delta = \beta$
3	60.03 $\pm$ 1.20	51.88 $\pm$ 1.25
5	60.44 $\pm$ 1.11	49.59 $\pm$ 2.33
7	60.64 $\pm$ 1.17	44.63 $\pm$ 1.64

Table 5.10: The (mean $\pm$ std.) of the classification error rates of TS-DSSL on CIFAR-10 with different levels of **uniform** noise. The results show the performance of TS-DSSL on various N classification intervals and various classification starting point. The columns are when we first start the classification which is either after the first epoch or after N epochs. Then, we show the performance of the proposed method when we change the frequency of labels prediction.

		Teacher		**Student**	
Noise%	β	$\delta = 1$	$\delta = \beta$	$\delta = 1$	$\delta = \beta$
original (true labels)	3	**7.61±0.06**	7.70±0.09	7.45±0.14	7.53±0.24
	5	8.68±0.43	7.78±0.14	8.20±0.35	7.62±0.14
	7	7.75±0.27	7.82±0.18	7.50±0.21	7.59±0.16
25%	3	11.22±0.13	11.15±0.11	11.37±0.16	11.39±0.16
	5	11.21±0.20	11.24±0.16	11.31±0.22	11.37±0.13
	7	**11.11±0.15**	11.36±0.12	11.29±0.12	11.47±0.13
30%	3	11.73±0.20	**11.61±0.17**	11.95±0.35	11.69±0.20
	5	11.63±0.20	11.66±0.27	11.86±0.16	11.92±0.22
	7	11.88±0.32	11.58±0.37	12.09±0.27	11.86±0.41
50%	3	14.74±0.06	14.71±0.20	15.47±0.17	15.65±0.20
	5	14.72±0.51	14.80±0.28	15.74±0.26	15.60±0.25
	7	14.72±0.28	**14.63±0.18**	15.55±0.26	15.53±0.18
	14	15.12±0.08	15.17±0.35	16.10±0.19	16.15±0.31
	21	15.30±0.16	15.13±0.36	16.22±0.24	16.17±0.20
70%	3	24.79±0.82	24.38±1.21	27.06±0.66	26.23±1.05
	5	25.17±0.83	24.70±0.65	27.23±0.69	27.24±0.90
	7	**23.91±0.95**	25.19±0.70	26.09±0.80	27.27±0.81
75%	3	30.67±0.68	**30.26±0.81**	32.61±0.67	32.21±1.07
	5	31.72±1.34	31.10± 0.52	33.97±1.02	33.15±0.68
	7	30.68±0.85	32.45±0.61	32.69±0.53	34.32±0.90

Table 5.11: The (mean $\pm$ std.) of classification error rates of the *self-cleaning I* baseline model after each phase. The table shows the performance of the produced model on the testing data after completely training a WRN model. The first model was trained on CIFAR10 with a level of 50% uniform noise.

Phase	Error	Description
Phase 1	18.82 $\pm$ 0.37	The results of models trained on 50% uniform noise
Phase 2	16.79 $\pm$ 0.11	The results of models trained on dataset with predicted labels from the trained model from previous phase (phase 1)
Phase 3	17.05 $\pm$ 0.30	The results of models trained on dataset with predicted labels from the trained model from previous phase (phase 2)

5.4 Conclusion

In this chapter, we presented TS-DSSL, a semi-supervised classification method. TS-DSSL accepts as input a noisy training dataset and employs a self-training and self-cleansing techniques to train a model. The training protocol maintains the model stability and enhances the overall classification performance. TS-DSSL is constructed from a sequence of convolutional layers forked at the end into two classifiers. The primary classifier is called a teacher. It uses the initially given noisy label training data to build a knowledge that guides the secondary classifier through cleansing the noisy labels and maintain the model's stability. The secondary classifier is the student. It uses the knowledge learned by the teacher in previous training steps to clean the noisy labels in the training data, and then it uses the cleaned labels to train the following training steps.

Moreover, we measured the performance of TS-DSSL on the benchmark datasets CIFAR10 and MINIST with different levels of uniform and non-uniform noises in the labels of the training dataset. We also compared the efficiency of TS-DSSL with three baseline methods, as the state-of-the-art methods use small models with low performance. The experiments show that TS-DSSL sets a new state-of-the-art record for CIFAR-10 and MNIST datasets with different percents of noisy label training examples.

CHAPTER 6 CONCLUSION

We now summarize our main contributions in this dissertation in the following three main facets. First, we presented in chapter 3 a new Hybrid Residual Network Method (HyResNet) that exploits the power of both supervised and unsupervised deep learning methods into a single deep supervised learning model. We tested HyResNet via empirical studies on visual object recognition tasks using benchmark datasets with various configurations and settings. HyResNet showed comparable results to the state-of-the-art methods on the benchmark datasets.

Second, we proposed a deep semi-supervised learning method (DSSL) in chapter 4. DSSL utilizes both supervised and unsupervised neural networks. The novelty of DSSL originates from its nature in employing a limited number of labeled training examples in conjunction with sufficiently large unlabeled examples to create a classification model. The combination of DSSL architecture and self-training has a joint impact on the performance over the DSSL. We measured the performance of DSSL method on five benchmark datasets with various labeled / unlabeled levels of training examples and then compared our results with state-of-the-art methods. The experiments show that DSSL sets a new state-of-the-art record for various benchmark tasks.

Finally, we introduced in chapter 5 a new teacher/student semi-supervised deep learning methods (TS-DSSL). TS-DSSL accepts as input a noisy training dataset and employs a self-training and self-cleansing techniques to train a deep learning model. The integration of TS-DSSL architecture with the proposed training protocol maintains the stability of the TS-DSSL model and enhances the overall model performance. TS-DSSL is constructed from a sequence of convolutional layers forked at the end into two classifiers. The

primary classifier is called a teacher and another helper classifier is called the student. Moreover, we evaluated the performance of TS-DSSL on benchmark semi-supervised learning tasks with different noisy labels distributions. The experiments showed that TS-DSSL sets new state-of-the-art records for on the benchmark tasks.

APPENDIX A: LIST OF PUBLICATIONS

- **Zeyad Hailat**, Xuewen Chen, “Teacher/Student Deep Semi-Supervised Learning,” *Submitted and under review.*

- **Zeyad Hailat**, Artem Komarichev, Xuewen Chen, “Deep Semi-Supervised Learning,” *24th International Conference on Pattern Recognition (ICPR)*, 2018.

- **Zeyad Hailat**, Artem Komarichev, Xuewen Chen, “HyResNet: A Hybrid Residual Network for Supervised Deep Learning,” *Wayne State University, Technical Report*, 2017.

- Tarik Alafif, **Zeyad Hailat**, Melih Aslan, Xuewen Chen, “On Classifying Facial Races with Partial Occlusions and Pose Variations.” *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017.

- Melih Aslan, **Zeyad Hailat**, Tarik K. Alafif, and Xuewen Chen, “Multi-channel multi-model feature learning for face recognition.” .*Pattern Recognition Letters* 85 (2017): 79-83.

- Tarik Alafif, **Zeyad Hailat**, Melih Aslan, Xuewen Chen, “On Detecting Partially Occluded Faces with Pose Variations,” *I-SPAN* (2017).

- Zhe Zhang, **Zeyad Hailat**, Marni J. Falk, and Xuewen Chen, “Integrative analysis of independent transcriptome data for rare diseases.” textitMethods 69, no. 3 (2014): 315-325.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning."

[2] T. Alafif, Z. Hailat, M. Aslan, and X. Chen, "On detecting partially occluded faces with pose variations," in *The 14th International Symposium on Pervasive Systems, Algorithms, and Networks (I-SPAN)*. IEEE Computer Society Conference Publishing Services (CPS), 2017.

[3] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[4] M. S. Aslan, Z. Hailat, T. K. Alafif, and X.-W. Chen, "Multi-channel multi-model feature learning for face recognition," *Pattern Recognition Letters*, vol. 85, pp. 79–83, 2017.

[5] S. Bell, P. Upchurch, N. Snavely, and K. Bala, "Material recognition in the wild with the materials in context database," *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[6] Y. Bengio *et al.*, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[7] C. Bishop, C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[8] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on*. IEEE, 2018, pp. 67–74.

[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[10] X.-w. Chen, M. S. Aslan, K. Zhang, and T. S. Huang, "Learning multi-channel deep feature representations for face recognition," in *Proceedings of The 1st International Workshop on Feature Extraction: Modern Questions and Challenges, NIPS*, 2015, pp. 60–71.

[11] X.-W. Chen and X. Lin, “Big data deep learning: challenges and perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014.

[12] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, “Describing textures in the wild,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3606–3613.

[13] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *arXiv preprint arXiv:1511.07289*, 2015.

[14] A. Coates, H. Lee, and A. Y. Ng, “An analysis of single-layer networks in unsupervised feature learning,” *Ann Arbor*, vol. 1001, no. 48109, p. 2, 2010.

[15] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[16] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[17] R. Collobert, C. Puhrsch, and G. Synnaeve, “Wav2letter: an end-to-end convnet-based speech recognition system,” *arXiv preprint arXiv:1609.03193*, 2016.

[18] E. Denton, S. Gross, and R. Fergus, “Semi-supervised learning with context-conditional generative adversarial networks,” *arXiv preprint arXiv:1611.06430*, 2016.

[19] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, “Discriminative unsupervised feature learning with convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2014, pp. 766–774.

[20] B. Frénay and M. Verleysen, “Classification in the presence of label noise: a survey,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869, 2014.

[21] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks.” in *Aistats*, vol. 9, 2010, pp. 249–256.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[23] I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio, "Multi-prediction deep boltzmann machines," in *Advances in Neural Information Processing Systems*, 2013, pp. 548–556.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[25] Z. Hailat, A. Komarichev, and X. Chen, "Hyresnet: A hybrid residual network for supervised deep learning." Wayne State University, 2017, Technical Report.

[26] Z. Hailat, A. Komarichev, and X. Chen, "Deep semi-supervised learning," in *24th International Conference on Pattern Recognition (ICPR)*, 2018.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 630–645.

[29] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

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

[31] E. Hoffer, I. Hubara, and N. Ailon, "Deep unsupervised learning through spatial contrasting," *arXiv preprint arXiv:1610.00243*, 2016.

[32] C. Huang, C. Change Loy, and X. Tang, "Unsupervised learning of discriminative attributes and visual representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5175–5184.

[33] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[34] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*. Springer, 2016, pp. 646–661.

[35] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411–430, 2000.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[38] I. Jindal, M. Nokleby, and X. Chen, "Learning deep networks from noisy labels with dropout regularization," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 967–972.

[39] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3581–3589.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

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

[42] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," *arXiv preprint arXiv:1610.02242*, 2016.

[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[44] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[45] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, “Deeply-supervised nets.” in *AISTATS*, vol. 2, no. 3, 2015, p. 5.

[46] M. Lin, Q. Chen, and S. Yan, “Network in network,” *arXiv preprint arXiv:1312.4400*, 2013.

[47] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[48] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[49] A. Makhzani and B. Frey, “A winner-take-all method for training sparse convolutional autoencoders,” in *NIPS Deep Learning Workshop*. Citeseer, 2014.

[50] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.

[51] D. McClosky, E. Charniak, and M. Johnson, “Reranking and self-training for parser adaptation,” in *COLINGâĂć ACL 2006*, 2006, p. 337.

[52] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[53] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, “Learning with noisy labels,” in *Advances in neural information processing systems*, 2013, pp. 1196–1204.

[54] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning.”

[55] A. Ng, “Sparse autoencoder,” *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[56] K. Nigam and R. Ghani, “Analyzing the effectiveness and applicability of co-training,” in *Proceedings of the 9th International CIKM*. ACM, 2000, pp. 86–93.

[57] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: A strategy employed by v1?” *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[58] M. Ranzato and M. Szummer, “Semi-supervised learning of compact document representations with deep networks,” in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 792–799.

[59] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko, “Semi-supervised learning with ladder networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 3546–3554.

[60] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *arXiv preprint arXiv:1412.6550*, 2014.

[61] C. Rosenberg, M. Hebert, and H. Schneiderman, “Semi-supervised self-training of object detection models,” in *2005 7th IEEE Workshops on Applications of Computer Vision (WACV/MOTION’05)-Volume 1*.

[62] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[63] M. Sajjadi, M. Javanmardi, and T. Tasdizen, “Mutual exclusivity loss for semi-supervised deep learning,” in *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1908–1912.

[64] M. Sajjadi, M. Javanmardi, and T. Tasdizen, “Regularization with stochastic transformations and perturbations for deep semi-supervised learning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 1163–1171.

[65] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2226–2234.

[66] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.

[67] F. Sebastiani, “Machine learning in automated text categorization,” *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

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

[69] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning, “Semi-supervised recursive autoencoders for predicting sentiment distributions,” in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 151–161.

[70] J. T. Springenberg, “Unsupervised and semi-supervised learning with categorical generative adversarial networks,” *arXiv preprint arXiv:1511.06390*, 2015.

[71] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net,” *arXiv preprint arXiv:1412.6806*, 2014.

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

[73] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” *arXiv preprint arXiv:1505.00387*, 2015.

[74] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, “Training convolutional networks with noisy labels,” *arXiv preprint arXiv:1406.2080*, 2014.

[75] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[76] S. Targ, D. Almeida, and K. Lyman, “Resnet in resnet: generalizing residual architectures,” *arXiv preprint arXiv:1603.08029*, 2016.

[77] A. Tarvainen and H. Valpola, “Weight-averaged consistency targets improve semi-supervised deep learning results,” *arXiv preprint arXiv:1703.01780*, 2017.

[78] H. Valpola, “From neural PCA to deep unsupervised learning,” *Advances in Independent Component Analysis and Learning Machines*, pp. 143–171, 2015.

[79] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.

[80] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Advances in neural information processing systems*, 2015, pp. 649–657.

[81] X. Zhu, “Semi-supervised learning literature survey,” *world*, vol. 10, p. 10, 2005.

ABSTRACT

DEEP LEARNING METHODS FOR VISUAL OBJECT RECOGNITION

by

ZEYAD HAILAT

August 2018

Advisor: Dr. Xuewen Chen

Major: Computer Science

Degree: Doctor of Philosophy

Convolutional neural networks (CNNs) attain state-of-the-art performance on various classification tasks assuming a sufficiently large number of labeled training examples. Unfortunately, curating sufficiently large labeled training dataset requires human involvement, which is expensive, time-consuming, and susceptible to noisy labels. Semi-supervised learning methods can alleviate the aforementioned problems by employing one of two techniques. The first approach os to utilize a limited number of labeled data in conjunction with sufficiently large unlabeled data to construct a classification model. The second approach is to utilize sufficiently large noisy label training data to learn a classification model. In this dissertation, we proposed new methods to mitigate the aforementioned problems. We summarize our main contributions below.

First, we presented a new Hybrid Residual Network Method (HyResNet) that exploits the power of both supervised and unsupervised deep learning methods into a single deep supervised learning model. Our experiments show the efficacy of HyResNet on visual object recognition tasks. We tested HyResNet on benchmark datasets with various configurations and settings. HyResNet showed comparable results to the state-of-the-art methods on benchmark datasets.

Second, we proposed a new deep semi-supervised learning method (DSSL). DSSL utilizes both supervised and unsupervised neural networks. The novelty of DSSL orig-

inates from its nature in employing a limited number of labeled training examples in conjunction with sufficiently large unlabeled examples to create a classification model. The combination of DSSL architecture and self-training has a joint impact on the performance over the DSSL. We measured the performance of DSSL method on five benchmark datasets with various labeled / unlabeled ratios of training examples and then compared our results with state-of-the-art methods. The experiments show that DSSL sets a new state-of-the-art record for various benchmark tasks.

Finally, we introduced a new teacher / student semi-supervised deep learning method (TS-DSSL). TS-DSSL accepts as input a noisy labeled training dataset and then it employs a self-training technique to train a deep learning model. The integration of TS-DSSL architecture with the proposed training protocol maintain the stability of the method and enhance the overall method performance. We evaluated the efficiency of TS-DSSL on benchmark semi-supervised learning tasks with different levels of noisy labels that we synthesized from uniform and non-uniform noise distributions. The experiments showed that TS-DSSL sets a new state-of-the-art record on the benchmark tasks.

AUTOBIOGRAPHICAL STATEMENT

Zeyad Hailat is currently a Ph.D. candidate in the Department of Computer Science at Wayne State University, Detroit, MI, USA. He works under the supervision of Dr. Xuewen Chen. Before joining Wayne State University, Mr. Hailat has received a Bachelor's and a Master's degrees in Computer Science from Yarmouk University, Jordan, in 2006 and 2009, respectively.

The academic experience of Mr. Hailat includes teaching courses in diverse computer topics for various undergraduate levels. On the practical computer side, he is exposed to various state-of-the-art computer technologies, general purpose programming languages, artificial intelligence tools, and a wide range of computer hardware and software.

His research interests broadly include artificial intelligence, data sciences, deep learning, machine learning, data mining, computer vision, and big data. Specifically, his research focus mainly on developing and implementing new machine learning (supervised, unsupervised, and semi-supervised deep learning) methods to solve real-life challenging problems as well as problems in computer vision, natural language processing, speech recognition, and bioinformatics.