A Study of Self-training Variants for Semi-supervised Image Classification

A thesis

submitted in fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Computer Science

at

The University of Waikato

by

Attaullah Sahito

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2021

To my parents.

Abstract

Artificial neural networks achieve state-of-the-art performance when trained on a vast number of labelled examples. Still, they can easily overfit training examples when few labelled examples are available. The requirement to have labels for all training examples is a strong limitation of standard supervised machine learning. This can be addressed by applying semi-supervised learning methods that extend supervised learning and use unlabelled examples. Self-training is the most basic and generic semi-supervised approach. In self-training, a model is trained iteratively on both labelled and pseudo-labelled examples obtained from previous iterations. This thesis focuses on the task of investigating different variants of self-training by applying metric learning, transfer learning, and self-supervised learning.

The first part of this thesis investigates how metric learning can be applied to self-training. This is achieved by applying several metric learning losses for the training of feedforward neural networks. Experimental results show that triplet loss – a metric learning loss – can achieve better results than cross-entropy loss with simple neural networks.

For improving the performance of self-training, the second part of the thesis investigates applying large neural networks and pretraining on various image sizes of ImageNet with different loss functions. Experimental results show that pretraining always improves the predictive performance of the model. Pretraining on smaller image sizes with cross-entropy loss provides the highest performance.

In the third part of this thesis, several self-training methods are developed using self-supervised learning. Geometric transformation-based self-supervised learning is applied to unlabelled examples. The experimental results indicate that applying self-supervised learning for only the first iteration achieves better performance than using it in all iterations of self-training.

Acknowledgements

I want to thank my supervisors, Bernhard Pfahringer and Eibe Frank, for the patient guidance, encouragement, and advice they have provided throughout my time as their student. I have been fortunate to have supervisors who cared so much about my work and promptly responded to my questions and queries. They always motivated me, especially during the lockdown period. I am also thankful to Bob Durrant for guiding me at the start of my PhD.

Completing this work would have been all more difficult were it not for the support and friendship provided by the other members of the Machine Learning group of the Department of Computer Science at the University of Waikato. Especially, Vithya, Chen, and Hongyu for interesting discussions. I am indebted to them for their help.

I am thankful to Te Pūnaha Matatini Whānau for organising several retreats and providing a chance to visit various cities and explore the beauty of New Zealand.

I would like to thank the Pakistani community of Hamilton. They made my stay here in New Zealand memorable and organised many events and gatherings so that I did not feel homesick.

I would like to thank Higher Education Commission (HEC) Pakistan for funding my doctoral study. Finally, I would like to thank the financial support of the University of Waikato and the machine learning group that funded all my conference and summer school trips.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data is of ever-increasing abundance in the modern world. Such a vast amount of data provides opportunities to exploit. However, much of this data is complex, noisy, and lacks obvious structure. Therefore, explicit modelling of, for example, its distribution is too challenging for a human agent. On the other hand, a human can specify an explicit procedure, i.e., an algorithm, for constructing a suitable model. Machine learning (ML) is concerned with algorithms that enable computers to learn from the data. The majority of the algorithms that have been developed in this area of research perform what is known as supervised learning. These algorithms take a set of training examples and produce a predictive model as output. Typically, each training example is a vector of feature values and an associated label provided by some labelling mechanism—the "supervisor". The learning algorithm aims to find a model that will generalise to examples that are not available during the training process, so-called test examples, and correctly assign labels to these new examples. The requirement to have labels for all training examples is a strong limitation of standard supervised machine learning. There is another related class of algorithms that perform unsupervised learning, and these algorithms do not require label information for the training of the model. Unsupervised learning algorithms search for structures and patterns in data. They are useful for knowledge discovery from data but not necessarily immediately suitable for

predictive modelling. Given the scarcity of labelled data in many applications, an important question in machine learning research is whether unlabelled data can be used in conjunction with labelled data to improve the accuracy of predictive models. Semi-supervised learning (SSL) lies somewhere between supervised and unsupervised learning and can use labelled and unlabelled examples. SSL methods are designed to work with labelled and unlabeled examples.

In the last decade, much of the boom in machine learning has been seen around deep learning – techniques for training large artificial neural networks. Due to these techniques, we now can construct systems that can recognise thousands of everyday objects in photos (Russakovsky et al., 2015). Previous attempts to solve these tasks relied on hand-designed features such as bag of visual words (Perronnin et al., 2010) or Fisher vectors (Sánchez et al., 2013). Artificial neural networks apply end-to-end learning from data designed to extract features from the raw signal available in the data. They do not rely on predefined human feature engineering techniques.

Modern artificial neural networks are also known as deep neural networks, where the qualification "deep" relates to the large number of layers involved in these models. Particular types of layers may be suitable for certain applications. For example, convolutional layers introduced in (LeCun et al., 1989) are core building blocks of neural networks for image data. Deep models have a large number of parameters that need to be estimated from the training data, and suitable initial parameter values can substantially influence the outcome of learning. For example, the parameters of neural networks can be initialised randomly or transferred pretrained from other related tasks—this is a form of transfer learning. Generally, some form of the stochastic gradient-based optimisation algorithm is used to fit the parameters of the network by minimising a function that measures the predictive error of the model on the training data—the so-called "loss function". For supervised training of neural networks, the cross-entropy loss function is widely used for optimising the parameters. Cross-entropy loss is estimated by comparing the labels in the

training data and the model's predictions.

Neural networks of sufficient size are universal function approximators (Hornik et al., 1989) and can even perfectly fit a dataset with randomised labels (Zhang et al., 2021). However, for the training of a complex neural network, a large number of labelled examples may be required, and these networks can easily overfit when trained on a small number of labelled examples: prediction on training data may be much more accurate than predictions on test examples not seen during the training process.

In many real-world scenarios, substantially fewer labelled examples are available than unlabelled examples. Manual annotation of data is costly and time-consuming, and for applications in areas such as medicine, it may not be feasible to collect more labelled examples. In these situations, SSL approaches may be useful: the idea is that training can be improved by using a limited number of labelled examples along with a large number of unlabelled examples. In the simplest case, SSL methods can be trained iteratively on labelled examples and unlabelled examples with predictions from previous iterations of model training used to label the unlabeled examples. This is known as self-training. This widely-used yet straightforward SSL strategy is the focus of this thesis. We will investigate the effect of metric learning, transfer learning, and self-supervised learning on the performance of SSL using self-training.

Considering metric learning, we will specifically focus on using so-called Siamese networks for this task. For a small number of labelled examples, Siamese networks (Bromley et al., 1993) are considered very efficient, especially when few labelled examples are available per class (Koch et al., 2015). Hence, they can be very useful when labelled examples are scarce. The training of Siamese networks is performed using a loss function designed to enable the networks to learn a distance function—a metric— over training examples using label information.

Just as metric learning, transfer learning is an appealing method to reduce the amount of labelled data required for successful learning and a promising

tool to investigate in the context of semi-supervised learning using self-training. Transfer learning is commonly performed by taking pretrained parameters obtained from a related domain of data for which a large amount of labelled data is available.

If transfer learning cannot be used, or the transferred parameters are not well-suited to the target domain, then a promising alternative approach can be employed to create an artificial classification problem based on *unlabeled data* from the *target domain* and obtaining parameters by training on this *auxiliary* training task—for example, detecting whether an image has been rotated or not. This is known as self-supervised learning and a fairly recent development in the literature on deep learning.

This thesis investigates ways in which metric learning and self-supervised learning can be applied to self-training employing transfer learning. Three avenues of research are followed:

- Applying metric learning for self-training to learn a similarity function motivated by the hypothesis that this will yield better generalisation performance than self-training using cross-entropy.

- Exploring the effect of transferring pretrained parameters for the initialisation of deep neural networks in self-training, based on the observation that transfer of knowledge is particularly useful when small amounts of labelled data are available for training.

- Adapting self-supervised learning for self-training to exploit unlabelled examples in a manner that goes beyond what self-training does by labelling unlabeled examples, thus improving the predictive performance of the model.

These three research directions are motivated by the goal of designing learning algorithms that can efficiently employ labelled and unlabelled examples to improve the model's predictive performance. The first point is based on using metric learning on labelled examples and propagating label information to

unlabelled examples based on the metric that has been learned. The second point investigates the effect of applying transfer learning from related tasks with various sizes of inputs. The third line of enquiry takes the alternative approach of exploiting unlabelled examples by creating auxiliary tasks from this data, rather than resorting to pre-trained parameters obtained from some other classification task performed on a similar but different domain.

The rest of this chapter briefly discusses the main concepts introduced above and summarises the contributions and structure of the remainder of the thesis.

1.1 Semi-supervised Learning

Many tasks in image classification require learning from a small number of labelled examples. Semi-supervised learning and transfer learning aim to achieve fast generalisation from a small number of labelled examples by leveraging unlabelled data or labelled data from other domains, respectively. SSL methods are designed to work with labelled instances $L = \{(x_1, y_1), (x_2, y_2), ..., (x_{|L|}, y_{|L|})\}$ and unlabelled instances $U = \{x_1^{'}, x_2^{'}, ..., x_{|U|}^{'}\}$, where X and Y relate to an input space and output space, $x_i, x_j^{'} \in X (i = 1, 2, ..., |L|, j = 1, 2, ..., |U|)$ are feature vectors with d dimensions (e.g., the RGB pixel values of an image) and $y_i \in Y$ are labels of x_i. Usually, these methods assume a much smaller number of labelled than unlabelled instances, i.e., $|L| \ll |U|$, because unlabelled instances are often easy to acquire. SSL has proven to be useful, especially when we are dealing with anti-causal or confounding problems (Peters et al., 2017).

A generic technique for semi-supervised learning is self-training. In the simplest instantiation of this technique, a predictive model is first trained on the labelled data, and, once this has been done, this trained model is used to predict labels for the unlabelled data—these labels are called ”pseudo labels” because they are not actual labels available in the original data. The unlabelled

data with its pseudo labels is then merged with the original labelled data, and the model is retrained on this mixture of data. The whole process can be iterated by modifying the pseudo labels using predictions of the latest model trained until a satisfactory model has been obtained.

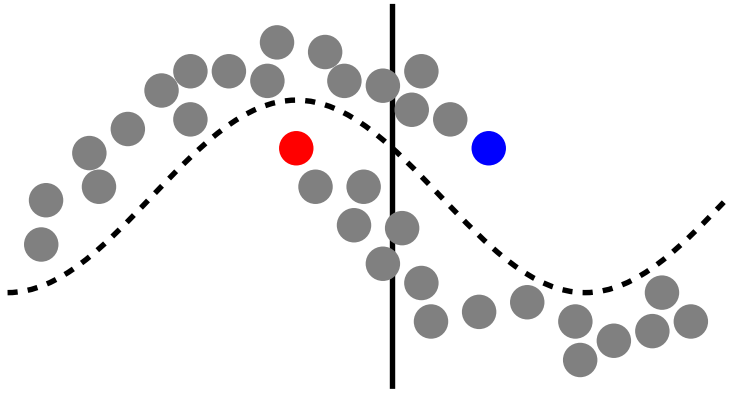


Figure 1.1: Binary classification problem depiction using labelled and unlabelled instances of toy dataset.

Figure 1.1 shows a binary class toy dataset. Labelled instances are shown in colour, while unlabelled are in grey. A reasonable decision boundary obtained using purely supervised learning is shown as a vertical line; the optimal decision boundary is shown in the dashed curved line, which could potentially be approximated by a semi-supervised learning algorithm that can make use of the unlabeled data to inform the location and shape of the decision boundary. This figure shows the potential of using unlabelled instances by applying semi-supervised learning.

Semi-Supervised Learning: Assumptions

Without making any assumptions on how the inputs and outputs are related, it is impossible to justify semi-supervised learning as a principled approach. One common set of assumptions for SSL was proposed by (Chapelle et al., 2006). These assumptions are:

1. **Smoothness assumption**: If two points x_1, x_2 are close in a high-density region, then their corresponding outputs y_1, y_2 should also be

close.

2. **Cluster assumption**: If points are in the same cluster, they are likely to be of the same class.

3. **Low-density separation**: The decision boundary between classes should lie in a low-density region of input space.

4. **Manifold assumption**: The high-dimensional data lie on a low-dimensional manifold.

For the SSL paradigm, the smoothness assumption helps us link unlabelled data to labelled data. The smoothness assumption also holds for supervised learning. The cluster assumption does not suggest that the number of clusters will be equal to the number of classes. There can be multiple smaller clusters for a particular class. The low-density separation assumption follows the cluster assumption. For example, if we cut through a high-density region of a cluster belonging to a particular class, we might assign points of a single class to two classes. From the manifold assumption, it can be inferred that there must exist a way for projecting data onto the manifold.

1.2 Metric Learning

Metric learning aims to find a distance function that can establish the similarity and dissimilarity between data points. It operates on the relationships between the data points and tries to find a metric that groups data with the same label and pulls data with different labels apart. Once an accurate metric has been learned, it can be used to obtain predictions using a simple similarity-based strategy, e.g., the classic nearest neighbour classifier. When applied to neural networks, metric learning implicitly learns discriminative features of the data. In situations where the usual classification loss, i.e., cross-entropy, is not feasible, metric learning can be applied. For instance, in similarity search retrieval, where there is only one ”example” of each class in the data: face

verification (Schroff et al., 2015) and person re-identification (Hermans et al., 2017). Metric learning is normally based on special-purpose loss functions such as triplet loss or contrastive loss instead of cross-entropy loss. These will be discussed in more detail in Chapter 2.

1.3 Transfer Learning

In ML, transfer learning is concerned with techniques for applying the knowledge gained from one task to another task. Intuitively and in practice, this is most useful when insufficient data is available to train an accurate model for the target task. The simplest way of performing transfer learning for deep learning is to reuse the neural network parameters trained on a related task with abundant data to apply them as a starting point for the target task. For instance, in deep image classification, the neural network parameters fitted on ImageNet (Russakovsky et al., 2015) are widely used in object detection problems (Girshick et al., 2014). The motivation behind using pretrained parameters of neural networks is to obtain better generalisation performance than using randomly initialised parameters by providing a better starting point for parameter optimisation using gradient descent-based methods, especially when few labelled examples from the target task are available for training.

1.4 Self-supervised Learning

Self-supervised learning is a type of unsupervised learning in the sense that it can be applied to suitable forms of data, such as image data, when no labels are available for the target task. It is based on the ingenious idea of creating a pretext learning task from the unlabelled data. The hope is that features learned in this task will be relevant for the target task that needs to be addressed. For example, a possible way to create a pretext task for images is to rotate them and use ML to learn to identify the angle of rotation. In this way, self-supervised learning applies supervised training on

unlabelled examples for learning features that form a useful representation of the data. Thus, it is a form of representation learning in which a human labeller does not provide labels; they come from the data itself by applying algorithmic transformations. The assumption motivating the application of this type of technique is that representations learned by self-supervised training carry semantic and structural meaning and can be helpful for a variety of tasks. Indeed, self-supervised learning has been successfully applied to image classification tasks (Gidaris et al., 2018) for improving predictive performance.

1.5 Contributions and Thesis Outline

The main thesis hypothesis is that the judicious use of metric learning, transfer learning, and self-supervised learning improves the accuracy of deep neural networks for image classification learned using self-training. Based on the hypothesis, the three primary lines of investigation are applying (a) metric learning, (b) transfer learning, and (c) self-supervised learning with self-training. The main contributions of this thesis are as follows:

- Application of metric learning losses for self-training and development of two confidence measures for selection of confidently labelled unlabelled examples for self-training. Metric learning using triplet loss is experimentally shown to produce similar or higher predictive performance than using cross-entropy loss.

- Evaluation of pretrained deep neural networks on various sizes of ImageNet and using different losses, and experiments showing that transfer learning with pretraining on smaller image sizes produces high predictive performance.

- An investigation of self-supervised learning with self-training and empirical results showing that self-supervised learning is more beneficial when applied only in the first iteration of self-training.

The structure of the thesis is as follows:

Chapter 2 introduces the basic building blocks of neural networks, loss functions for the training of neural networks, optimisation methods used for updating parameters of neural networks, and the datasets used in this thesis for evaluation of the proposed methods.

Chapter 3 presents a taxonomy of semi-supervised learning approaches and reviews the related literature.

Chapter 4 presents a self-training approach based on metric learning and various confidence measures for pseudo-label selection in self-training. The proposed approaches are evaluated empirically using a variety of datasets.

Chapter 5 investigates the use of transfer learning in the self-training paradigm. As part of the experiments, a set of three new deep neural networks is pretrained on ImageNet from scratch on three different sizes of images: 224 by 224, 64 by 64, and 32 by 32, and the performance of the self-training algorithm using these special networks is compared to that obtained using a publicly available pretrained network for images of size 224 by 244.

Chapter 6 introduces the use of self-supervised learning to self-training in three different ways. The proposed approaches are evaluated empirically using randomly initialised and pretrained parameters of the neural networks on various datasets.

Chapter 7 provides a summary of the contributions of this thesis and speculates about future research directions that could result from the work that has been undertaken.

A substantial amount of the work presented in this thesis has been published already, but additional experimental results have been included. The

self-training approach described in Chapter 4 is based on a paper that was presented at the 2019 edition of the Australasian Joint AI Conference in Adelaide, Australia. For the thesis, two additional losses and one extra dataset have been added to evaluate the algorithm. The content of Chapter 5 was first presented at the 2020 Australasian Joint AI Conference in Canberra, Australia. For the thesis, three additional deep neural networks are pretrained from scratch on ImageNet. Moreover, smaller image sizes are also considered to evaluate proposed approaches for one of the datasets considered. A manuscript relating to the content of Chapter 6 is accepted at the 2021 Australasian Joint Conference on Artificial Intelligence in Sydney, Australia and under printing process at the time of writing this thesis.

1. Attaullah Sahito, Eibe Frank, and Bernhard Pfahringer. Semi-supervised learning using siamese networks. In Jixue Liu and James Bailey, editors, AI 2019: Advances in Artificial Intelligence, pages 586–597, Cham, 2019. Springer International Publishing. ISBN 978-3-030-35288-2

2. Attaullah Sahito, Eibe Frank, and Bernhard Pfahringer. Transfer of pretrained model weights substantially improves semi-supervised image classification. In Australasian Joint Conference on Artificial Intelligence, pages 433–444. Springer, 2020

3. Attaullah Sahito, Eibe Frank, and Bernhard Pfahringer. Better self-training for image classification through self-supervision. arXiv preprint arXiv:2109.00778, 2021

Chapter 2

Background

This chapter begins by reviewing common basic building blocks of deep neural networks to provide background for the techniques introduced in this thesis. This is followed by a description of the different loss functions used to train networks. The chapter concludes with a high-level overview of optimisation methods and a description of the datasets used in the experiments.

2.1 Neural Network Architectures

The simplest class of artificial neural networks consists of the so-called feedforward neural networks. When applying supervised learning or semi-supervised learning, these networks are trained to approximate some function f. For instance, a classifier maps an input x to a category y. The function being learned is parameterised by a large set of parameters that can be adjusted based on the training data. The aim is to find values for the parameters that minimise the value of the loss function on the training data. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the values of the parameters θ by backpropagating gradients with respect to loss function at the current point in parameter space and adjusting θ based on the information in the gradient (and, potentially, previous gradients from other points in parameter space). Feedforward networks are called networks because they are typically represented by composing many different functions together in the form of a

network of functions. For example, the three functions $f^{(1)}, f^{(2)}$, and $f^{(3)}$ can be connected in a chain to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. In this case, $f^{(1)}$ is called the first layer of the network, $f^{(2)}$ is called the second layer of the network, and so on. The final layer of the network is called the output layer, and intermediate layers are known as hidden layers.

2.1.1 Fully-connected Layers

Feedforward networks can contain a simple fully-connected layer with parameters θ given by W, b. A fully-connected layer can be expressed as

$$f(x; W, b) = x^T W + b \tag{2.1}$$

Usually, W is known as the weight matrix and b as the bias vector. The size of W is dependent on the input dimensions and output dimensions of the layer specified by the user. The weight matrix W can be initialised using different approaches; for random initialisation, the initialisation approach proposed in (Glorot and Bengio, 2010) is widely used.

The fully-connected layer can also contain an activation function. These activation functions introduce nonlinearities. The most frequently used activation function in hidden fully-connected layers is the rectified linear unit (ReLU). The motivation for using the ReLU as an activation function is that it does not saturate as the weights in the network become larger. It can be stated as

$$\phi^{relu}(x) = max(0, x) \tag{2.2}$$

where the max function is applied elementwise. The activation function used for the output layer of a network is determined by the task for which the network is being used. For instance, classification tasks will typically use a softmax activation function in the output layer, where the k-th output is given by:

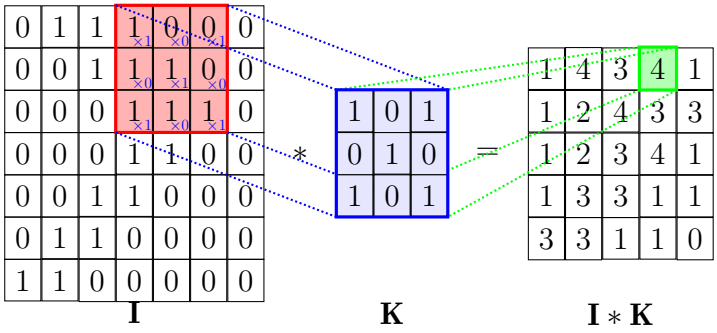


Figure 2.1: Illustration of a single convolutional layer.

$$\phi_k^{softmax}(x) = \frac{e^{x_k}}{\sum_{j=1}^{C} e^{x_j}} \tag{2.3}$$

and C is the total number of classes. This gives the estimated class probabilities, and a label is normally assigned to a test example by picking the class with maximum probability.

2.1.2 Convolutional Layers

Convolutional layers are another layer used in feedforward networks that are particularly useful for models that operate on image data. As images have a two-dimensional local structure that is spatially correlated, it is helpful to exploit this by enabling functions to operate locally in each neighbourhood. The functions can be viewed as filters in this scenario. In this way, a two-dimensional filter can extract local features instead of using fully-connected layers. With the local receptive fields, filters can extract elementary visual features such as edges, endpoints, and corners. An input is organised in planes within which all units share the same filter parameters. The application of a filter on a plane produces a feature map. For a feature map, the same operation is performed on different parts of the image. Figure 2.1 shows a single convolutional layer applied to an image and producing a feature map.

A pooling layer for downsampling usually follows convolutional layers. These layers reduce the output dimensions at a certain location with a summary statistic of the nearby outputs. For example, a) the max pooling opera-

tion reports the maximum output within a rectangular neighbourhood, b) the average pooling produces the mean within a rectangular neighbourhood.

For accelerating the training of the neural network, Ioffe and Szegedy (2015) proposed batch normalisation, which can be implemented in the form of additional batch normalisation layers. Batch normalisation is simple and easy to implement. It works by normalising the layer activations on each minibatch of data that is used for gradient descent (this will be discussed in more detail in Section 2.3). For d-dimensional input consisting of the activations of a layer, the normalised output can be calculated as:

$$\begin{aligned} \hat{x}^{(k)} &= \frac{x^{(k)} - \mu^{(k)}}{\sqrt{\sigma^{(k)^2} + \varepsilon}} \\ x &= \gamma \hat{x}^{(k)} + \beta \end{aligned} \tag{2.4}$$

where $k \in [1, d]$, and $\mu^{(k)}$, $\sigma^{(k)^2}$ are the per-dimension batch mean and variance respectively while the scaling γ and the shifting parameters β are learned during neural network training. A small value ϵ is used for numerical stability. Batch normalisation enables the use of higher learning rates, less careful parameter initialisation, and saturating nonlinearities.

These different types of layers can be combined to construct a fully functional neural network. Initially, neural networks were based on fully-connected layers. LeCun et al. (1989) introduced the use of convolutional layers for handwritten digit recognition. Furthermore, LeCun et al. (1998) proposed the LeNet family of neural networks. These networks consist of a series of blocks based on convolutional layers with a sigmoid activation function and pooling layers followed by several fully-connected layers. These models' success is due to the reduced number of parameters and the (limited) translation invariance of the activations of pooled outputs of convolutional layers.

The ImageNet Large Scale Visual Recognition Challenges has played a major role in advancing neural network architecture search. Krizhevsky et al. (2012) successfully applied a deep neural network on the ImageNet dataset, known as AlexNet. This network used convolutional layers with larger feature maps than the LeNet, and ReLU as an activation function.

2.1.3 VGG16 Architecture

Simonyan and Zisserman (2015) introduced the VGG family of networks based on smaller filters in convolutional layers, with a resolution of only 3×3, which results in better generalisation. They show that one can construct deeper networks using several convolutional layers between pooling layers for better predictive performance. The architectural diagram of VGG-16 is shown in Figure 2.2. Blocks of convolutional layers (CONV), with varying sizes of filters, e.g., 64, 128, 256, and 512, are followed by pooling layers. Fully-connected layers (FC) produce the final output of the network. ReLU is used as an activation function in all layers except the output layer. The total number of trainable parameters is 138,357,544. VGG networks can be used for feature extraction by removing fully-connected layers, which reduces the total number of parameters to 14,846,016.

2.1.4 Residual Networks

The residual networks proposed by He et al. (2016) improved performance on the ImageNet dataset while reducing the parameter count compared to VGG networks by having fewer feature maps in each layer and increasing the network depth. However, the main contribution was introducing residual connections, which enable the construction of residual blocks that use skip connections. The output from the previous layer is added to the output of the next residual block. These skip connections help remedy shrinking gradients and enable the training of much deeper neural networks. In a further twist, Zagoruyko and Komodakis (2016) have shown that wide residual networks with depths similar to VGG networks perform as well as the much deeper residual networks trained by (He et al., 2016), with the bonus of being more computationally efficient. The architectural diagram of a Wide Residual Network (WRN) with depth 28 and widening factor 2 is shown in Figure 2.2. The depth determines the number of convolutional layers, while the widening factor determines the convolutional layers' filter size. Batch normalisation is employed in all convolutional layers,

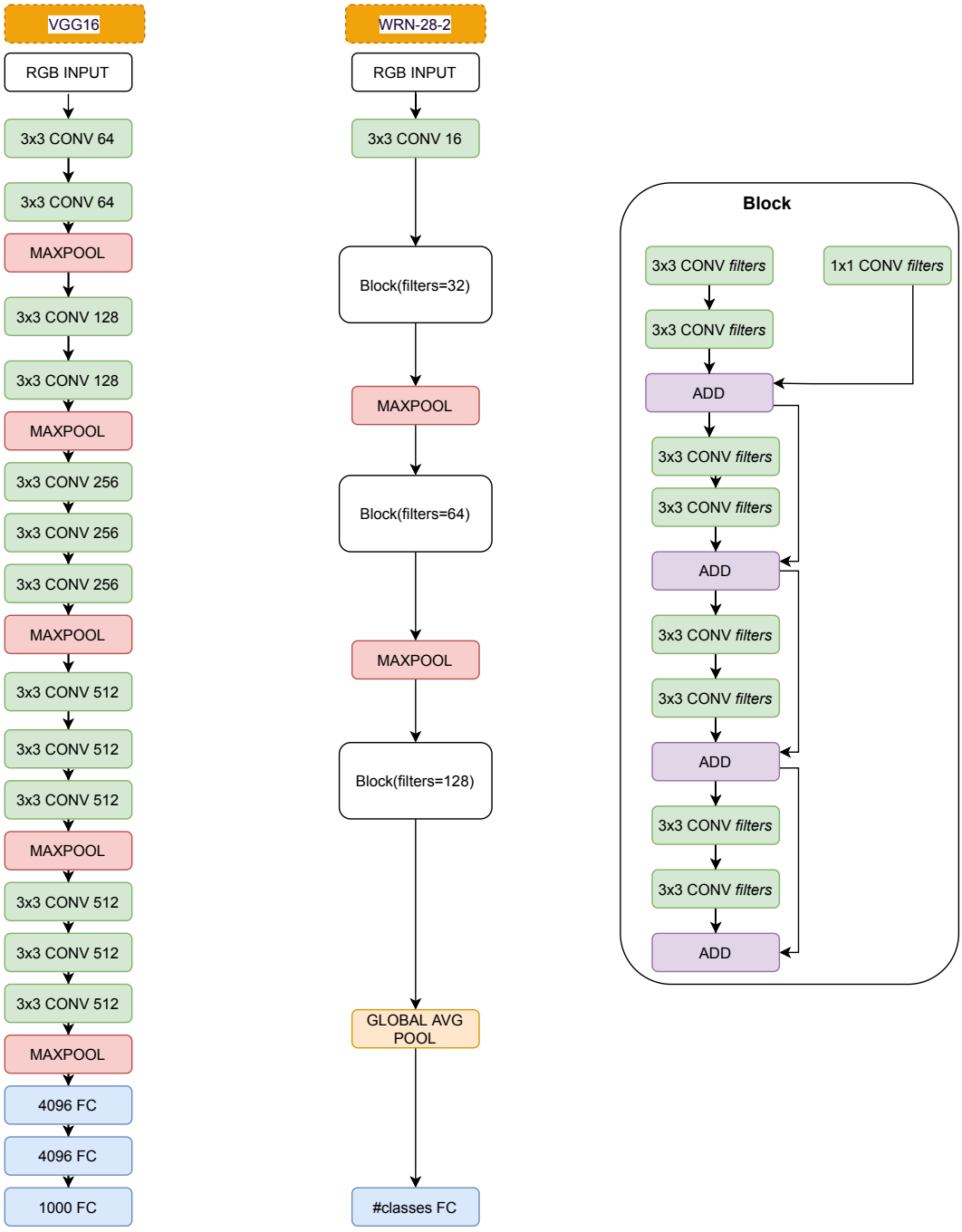


Figure 2.2: VGG16 and Wide Residual Network-28-2 architectural diagram.

followed by ReLU as an activation function. This network contains four blocks of convolutional layers. The first block contains a single convolutional layer with 16 filters. The remaining three blocks share the same block structure with varying sizes of filters, i.e., 32, 64, and 128. The block structure shown on the right side of Figure 2.2 consists of convolutional layers and residual connections. Finally, a fully-connected layer follows the global average pooling. The total number of trainable parameters is 1,474,576, which is very much smaller than VGG-16.

2.2 Loss Functions

This section discusses the most widely used classification loss function, softmax cross-entropy, and the three metric learning loss functions used in this thesis.

2.2.1 Softmax Cross-entropy Loss

The most frequently used classification loss function is softmax cross-entropy loss, which is a measure of the difference between the desired probability distribution and the predicted probability distribution. Softmax cross-entropy loss can be calculated as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(z_i) \tag{2.5}$$

where z_i is the output of the softmax activation function for the i^{th} sample, y_i the corresponding ground truth, and C is the number of classes.

2.2.2 Contrastive Loss

The contrastive loss (Hadsell et al., 2006) is a pair-based loss that attempts to bring similar examples closer to each other and push dissimilar examples farther away with respect to a minimum margin m. Examples of the same class are considered similar. Contrastive loss for embedded pairs of examples (x_1, x_2) can be calculated using the Euclidean distance $d(.,.)$ as:

$$\mathcal{L} = yd(x_1, x_2) + (1 - y)max(0, m - d(x_1, x_2)) \tag{2.6}$$

where $y = 1$ if x_1 and x_2 are from the same class, and $y = 0$ otherwise. The contrastive loss aims to bring embeddings from similar examples closer and push embeddings farther away from dissimilar examples according to the user-specified hyperparameter specifying the margin m.

2.2.3 Triplet Loss

The triplet loss (Weinberger and Saul, 2009) has been used for face recognition (Schroff et al., 2015). A triplet contains an anchor example a, positive example p, and a negative example n, where a, and p are from the same class and n must be from a different class. The triplet is provided to the network as a training example for learning suitable embeddings. During the optimisation of the network parameters, valid triplets (i, j, k) are selected where $label[i] = label[j], i \neq j$ and $label[i] \neq label[k]$ for calculation of the loss. The loss is calculated according to the following equation using the Euclidean distance $d(.,.)$ between the embeddings of triplets:

$$\mathcal{L} = max(d(a, p) - d(a, n) + m, 0) \tag{2.7}$$

where m is the so-called "margin" and constitutes a hyperparameter.

The triplet loss attempts to push away the embedded negative example n from the embedded anchor example a based on a given margin m and the given positive example p. As illustrated in Figure 2.3, depending on the negative example's location relative to the anchor and the positive example, it is possible to distinguish between hard negative examples, semi-hard negative examples, and easy negative examples. The latter are effectively ignored during optimisation because they yield the value zero for the loss.

- **Easy triplets**: have a loss of 0.

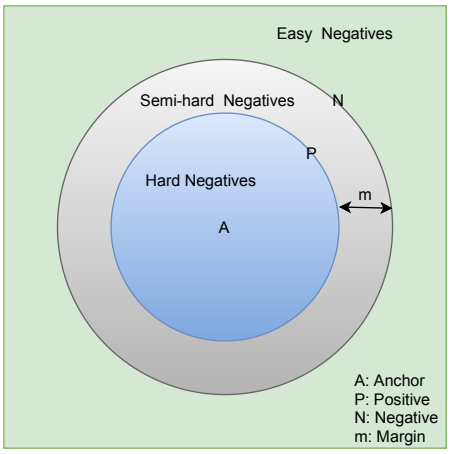


Figure 2.3: The three corresponding regions of the embeddings space for the negatives in Triplet loss.

- **Hard triplets**: the negative n is closer to the anchor a than the positive p.
- **Semi-hard triplets**: the negative n is not closer to the anchor a than the positive p, but n is within the margin, thus still returning a positive loss.

In this thesis, only semi-hard triplets are employed to evaluate triplet loss during neural network training as per suggestion from Schroff et al. (2015).

2.2.4 ArcFace Loss

ArcFace loss (Deng et al., 2019) is a modified cross-entropy loss with angular margins in the softmax expression, which is claimed to result in improved discriminative capacity for metric learning.

The ArcFace transforms the logits so that $W_j^T x_i = ||W_j||\ ||x_i|| \cos\phi_j$, where ϕ_j is the angle between features x_i and weight W_j. The weight is $l2$-normalized, giving $||W_j|| = 1$. The feature x_i is also $l2$-normalized and rescaled to s. The normalisation on weights and features makes predictions dependent only on the angle between weights and features. The learned embeddings are distributed on a hypersphere with radius s. An additive angular margin penalty m is added to the angle ϕ_{y_i} for a given label y_i. ArcFace loss can be calculated as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\frac{e^{s\cdot cos(\phi_{y_i}+m)}}{e^{s\cdot cos(\phi_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{C} e^{s\cdot cos\phi_j}} \tag{2.8}$$

where N is the training sample size and C is the number of classes.

2.3 Optimisation Methods

The training of neural networks proceeds by backpropagating (Rumelhart et al., 1986) gradient information consisting of partial derivatives of the loss function with respect to network parameters and then using an optimisation algorithm to update the parameters to optimise the objective. The optimisation method determines how the weights of the network are modified. The majority of the optimisation methods are based on only first-order gradient information. Ideally, we want to minimise the loss function, which represents the objective. Supervised learning based on a loss $\mathcal{L}$ of a function f parametrised by θ can be expressed as

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N}[\mathcal{L}(f(x_i;\theta), y_i)] \tag{2.9}$$

N is the number of training instances, x_i is the feature vector of ith training instance, and y_i is the corresponding label.

A simple update rule based on gradient descent for modifying the network parameters θ_{t+1} at iteration t can be described as:

$$\theta_{t+1} = \theta_t - \alpha\nabla\mathcal{L}(f(x;\theta_t), y) \tag{2.10}$$

where α is the user-defined learning rate (also referred to as step size) for updating the parameters, and $\nabla\mathcal{L}(f(x;\theta_t), y)$ is the gradient consisting of the vector of partial derivatives of the loss function with respect to the network parameters. In vanilla (or "batch") gradient descent, all training samples' gradients are needed; this becomes computationally infeasible when the training set is very large. In practice, stochastic optimisation methods are used for updating network parameters. Generally, it is common to sample a small

number of instances from a dataset known as a minibatch. The gradient is then calculated based on the data in the minibatch only, and the gradient descent update rule is applied after each minibatch to adapt the parameters. This is known as Stochastic Gradient Descent (SGD). This method converges at a sublinear rate (Sun et al., 2019). This rate can be improved by introducing momentum. Nesterov (1983) proposed the Nesterov accelerated gradient descent (NAG) method based on momentum. NAG works by accumulating the previous gradient as momentum and updates the network parameters by employing momentum. The update rule can be formulated as:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \alpha \nabla \mathcal{L}(f(x; \theta_t - \gamma v_{t-1}), y) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{2.11}$$

The momentum term γ is often a large value such as 0.9.

Manually setting a suitable global learning rate for optimisation algorithms can be difficult. In these situations, adaptive learning rate optimisation methods can be helpful. AdaGrad (Duchi et al., 2011) adaptively adjusts the learning rate according to the sum of the squares of all historical gradients. The update rule for AdaGrad can be expressed as:

$$\begin{aligned} V_t &= \sqrt{\sum_{i=1}^{t} \nabla \mathcal{L}(f(x; \theta_t), y)^2 + \epsilon} \\ \theta_{t+1} &= \theta_t - \alpha \frac{\nabla \mathcal{L}(f(x; \theta_t)}{V_t} \end{aligned} \tag{2.12}$$

where V_t is the accumulated historical gradient of the parameters at iteration t and ϵ is a smoothing term that avoids division by zero (usually set to 10^{-8}).

As the training progresses, the accumulated gradients will become larger, making the denominator larger, resulting in ineffective parameter updates. To overcome smaller updates at the late stage of training, AdaDelta (Zeiler, 2012) and RMSProp[1] introduced an exponential moving average for the accumula-

[1] `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

tion of gradients V_t.

$$V_t = \sqrt{\beta V_{t-1} + (1-\beta)\sum_{i=1}^{t} \nabla \mathcal{L}(f(x;\theta_t), y)^2} \tag{2.13}$$

where β is an exponential decay parameter.

Adam (Kingma and Ba, 2015) combines the adaptive and momentum-based methods. Adam employs an exponentially decaying average of past squared gradients V_t like AdaDelta and RMSProp, as well as an exponentially decaying average of past gradients m_t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\sum_{i=1}^{t} \nabla \mathcal{L}(f(x;\theta_t), y) \\ V_t &= \sqrt{\beta_2 v_{t-1} + (1-\beta_2)\sum_{i=1}^{t} \nabla \mathcal{L}(f(x;\theta_t), y)^2} \\ \theta_{t+1} &= m_t - \alpha \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{m_t}{V_t + \epsilon} \end{aligned} \tag{2.14}$$

where β_1 and β_2 are exponential decay rates. In this thesis, the values of β_1, β_2 and ϵ are set as 0,9, 0.999, and 1e-7 as per default in the deep learning library Tensorflow (Abadi et al., 2016).

2.4 Datasets

Five different image classification datasets are used to evaluate the performance and robustness of the proposed methods in this thesis. For introducing diversity, datasets containing greyscale images and three colour channel images are used, with the number of classes ranging from 10 to 38 and the resolution ranging from 28 by 28 to 96 by 96. Summary information on the datasets is shown in Table 2.1.

2.4.1 MNIST

The classic MNIST (LeCun et al., 1999) dataset consists of greyscale 28 by 28 images of handwritten digits from 0 to 9. The dataset contains 60,000 training

Figure 2.4: A sample image from each class of the MNIST dataset.

Figure 2.5: A sample image from each class of the Fashion-MNIST dataset.

instances and 10,000 test instances. A sample image from each class is shown in Figure 2.4. All classes have the same number of images in the data.

2.4.2 Fashion-MNIST

The Fashion-MNIST dataset (Xiao et al., 2017) consists of 28 by 28 greyscale images of ten different fashion items. These items are T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle Boot. The dataset contains 60,000 training images and 10,000 test images. Figure 2.5 shows an image from each class of the dataset. As in MNIST, the classes are equally distributed.

2.4.3 SVHN

SVHN comprises 32 by 32 RGB images of house numbers taken from the Street View House Numbers dataset (Netzer et al., 2011). A single image can contain multiple digits, but only the digit in the centre determines the label. The dataset contains 73,752 training images and 26,302 test images. Figure 2.6 depicts a single image from each class.

SVHN has an imbalanced number of examples per class for training and test images. Figure 2.7 shows the frequency of all ten classes in descending order. The class of digits is shown along the x-axis. The count of training images is shown at the bottom in blue coloured bars, while the count of test

Figure 2.6: A sample image from each class of SVHN dataset.

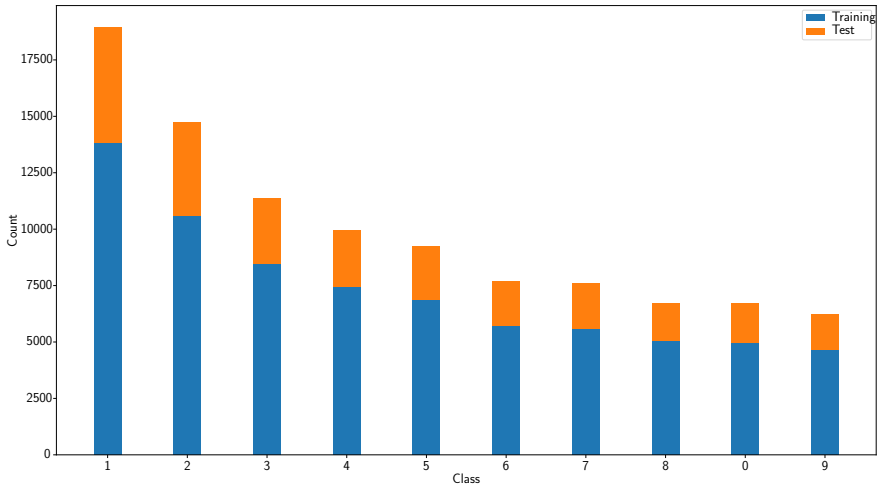


Figure 2.7: Class distribution of SVHN dataset.

images is stacked above the count of training images in orange coloured bars. Digit one is the most frequent, while digit nine is rarely occurring.

2.4.4 CIFAR-10

The CIFAR-10 (Krizhevsky and Hinton, 2010) dataset contains 32 by 32 natural colour images of ten different classes. The classes are aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset contains 50,000 training and 10,000 test images. Figure 2.8 depicts a single image from each class. All classes have the same number of images in the data.

Figure 2.8: A sample image from each class of the CIFAR-10 dataset.

Figure 2.9: A sample image from each class of the PlantVillage dataset.

2.4.5 PlantVillage

The PlantVillage (Hughes and Salathé, 2015) dataset consists of healthy and infected plant leaves of 14 different species. There are 38 classes, of which 12 correspond to healthy plants and 26 to infected plants. For some of the plant species, the dataset does not contain healthy leaves, while for some of the species, only infectious leaves are available. For more details, see (Hughes and Salathé, 2015). The dataset[2] contains 43,456 training and 10,849 test RGB images with a resolution of 256 by 256. The higher image size requires bigger convolutional neural networks for better predictive performance. In practice, very big neural networks are used for the PlantVillage dataset (Atila et al., 2021), which are computationally expensive. To overcome this, the images are resized to 96 by 96, 64 by 64, and 32 by 32. A sample image for each class is shown in Figure 2.9.

The PlantVillage dataset has an imbalanced number of images per class.

[2]Dataset is available at `https://github.com/attaullah/downsampled-plant-disease-dataset`

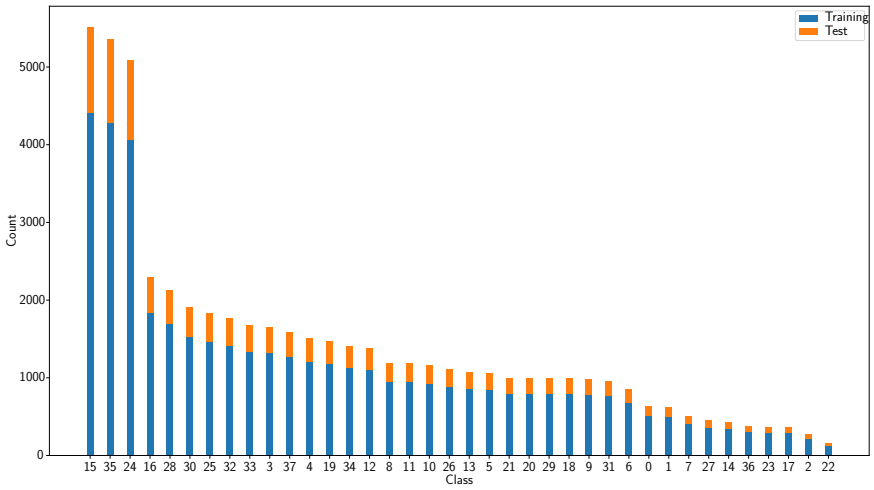


Figure 2.10: Class distribution of PlantVillage dataset.

Figure 2.10 shows the class distribution for all 38 classes sorted according to the frequency in descending order. The class index is shown along the x-axis. The count of training images is shown at the bottom in blue coloured bars, while the count of test images is stacked above the count of training images in orange coloured bars. There is a lot of variation in the total number of training images per class, ranging from a few thousand to a few hundred.

Table 2.1: Summary information of all five datasets.

Name	Training images	Test images	Size	Total classes
MNIST	60000	10000	(28,28,1)	10
Fashion MNIST	60000	10000	(28,28,1)	10
SVHN	73752	26032	(32,32,3)	10
CIFAR-10	50000	10000	(32,32,3)	10
PlantVillage32	43456	10,849	(32,32,3)	38
PlantVillage64	43456	10,849	(64,64,3)	38
PlantVillage96	43456	10,849	(96,96,3)	38

Chapter 3

An Overview of Semi-supervised Learning

Semi-supervised learning involves training on labelled as well as unlabeled data. Different approaches have been proposed in the literature based on the underlying assumptions about data to efficiently use unlabelled data to improve the model's predictive performance.

This chapter discusses a taxonomy of semi-supervised learning algorithms. This is followed by a description of the different types of methods and a discussion of previous attempts at applying them.

3.1 Taxonomy

Semi-supervised learning has been under study since the 1970s (McLachlan, 1975). Since then, a variety of methods have been proposed. These methods differ in exploiting semi-supervised learning assumptions and how they use unlabelled data. SSL methods can be divided into two main categories based on learning paradigms as proposed in (Van Engelen and Hoos, 2020), namely transductive learning and inductive learning. Transductive learning only predicts the label for available test examples at the training phase and does not generalise to unobserved examples. Inductive learning aims to learn a decision function, which can predict labels for any unseen data.

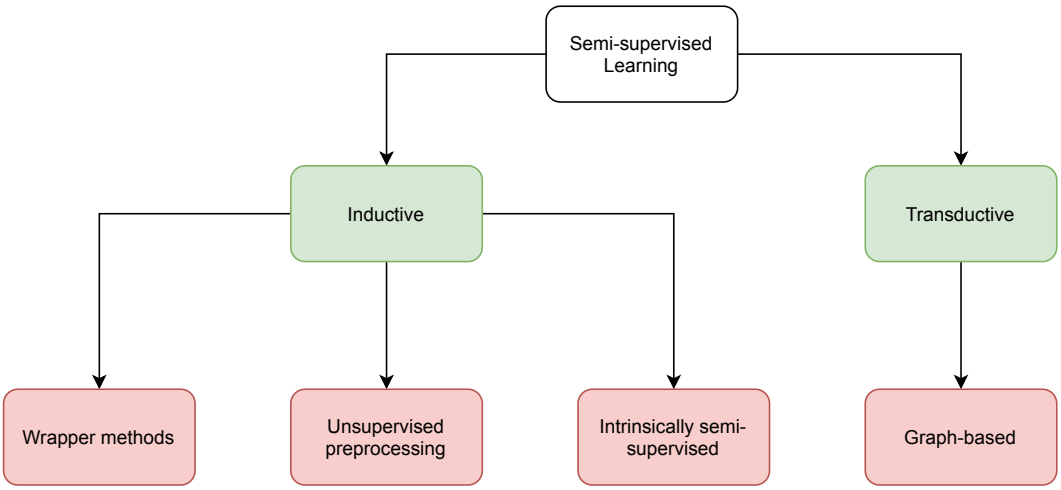


Figure 3.1: Taxonomy of Semi-supervised Learning methods (simplified) as proposed by Van Engelen and Hoos (2020).

3.2 Transductive Learning

The predictive capability of the transductive method is limited to the examples encountered during the training phase. There are no separate training and testing phases. This phenomenon intuitively gives rise to graph-based approaches. These approaches rely on the manifold assumption as well as the smoothness assumption. Transductive learning involves three steps: a) graph construction, b) graph weighting, and c) inference (Jebara et al., 2009). Each instance is represented as a node for graph construction, and an edge connects similar pairwise instances. Weighted edges represent the similarity between instances. Finally, the graph is used to infer labels of the unlabelled instances. The labels are transitively propagated to unlabeled instances from labelled ones. A well-known graph-based semi-supervised learning method is Local Learning with Global Consistency (LLGC) (Zhou et al., 2004); it is designed to promote the global consistency of labels on manifolds and local consistency in the input space. LLGC is described in detail in Section 4.2.

3.3 Inductive Learning

In contrast to transductive methods, inductive learning algorithms are designed to learn a predictive model that can generate predictions for any instance in the input space. The majority of the algorithms used in this thesis are based on inductive learning. Based on the usage of unlabeled examples, inductive learning methods can be further divided into different categories.

3.3.1 Wrapper Methods

The most widely used and oldest semi-supervised learning algorithms are based on wrapper methods (Zhu, 2005). They employ one or more base learners and iteratively use their confident predictions for retraining. In practice, the base learner is first trained on the small set of available labelled data and employed to predict labels for unlabelled data–commonly referred to as pseudo-labels. One can use single or multiple base learners on the same or different subset of the features. Self-training (also known as self-learning) is the most basic approach based on the wrapper idea. A single classifier is trained iteratively on initially labelled instances and employed to predict labels for unlabelled instances. Self-training has been successfully applied to object detection problems in the era preceding deep learning (Rosenberg et al., 2005) and achieved state-of-the-art. Considering deep learning, Pseudo-Label (Lee, 2013) is a simple self-training approach based on neural networks. A classifier is trained on the initially labelled and pseudo-labelled data, starting with a small weight of pseudo-labelled data. Pseudo-labels are less reliable at the start of training; therefore, the weights of examples with pseudo-labels are increased as the training progresses.

There are different design decisions offered by self-training. This includes the selection of pseudo-labels, reusing pseudo-labels for later iterations, and stopping criteria (Rosenberg et al., 2005; Triguero et al., 2015). The selection of pseudo-labelled data has a significant impact on the performance of

the model. Typically, prediction confidence is used as a selection criterion for the self-training paradigm. When the base learner classifier can produce probabilistic predictions, the respective probabilities can be employed as selection criteria for pseudo-labelled data. This is similar to Expectation-Maximization (EM) (Dempster et al., 1977), which has also been considered for semi-supervised learning (Nigam et al., 2006).

The introduction of multiple base learners in self-training gives rise to co-training. In co-training, two or more base learners are trained on available labelled data, and confident predictions from one base learner are used as labelled data for the other model. Co-training can be applied either using the same set of features across all base learners or a different subset of features (also known as multi-views) for each base learner. Blum and Mitchell (1998) proposed Multi-view Co-training, where two models are trained on two separate subsets of the data features. Confident predictions from one model are then used as labelled data for the other model. Co-EM (Brefeld and Scheffer, 2004) combined co-training with EM and achieved better results than either of them.

3.3.2 Unsupervised Preprocessing

Performing semi-supervised learning using unsupervised preprocessing employs labelled and unlabelled instances in two separate stages. In this type of method, unlabeled instances extract features and learn the underlying distribution of data, which helps achieve improved supervised performance when fine-tuned on labelled data.

Autoencoders can be used for this approach. An autoencoder is a neural network that is trained to reconstruct its input. Usually, the first part of the network is required to learn a compressed latent representation of the input, referred to as the encoder. The second part of the network tries to reconstruct the original input from the latent representation and is known as the decoder. In autoencoders, learning of a latent representation is closely related to the

notion of the input space containing low-dimensional manifolds. Sometimes, in autoencoders, noise is added to corrupt the input, and the reconstruction of non-corrupted input is used as an objective function known as denoising autoencoder (Vincent et al., 2008).

Erhan et al. (2010) suggested that unsupervised pretraining helps neural networks achieve better generalisation than those without pretraining. After unsupervised training on unlabelled data, the encoder is extracted and used to initialise the feature extraction layers of a model that is fine-tuned on labelled data.

3.3.3 Intrinsically Semi-supervised Methods

Unlike unsupervised preprocessing methods, intrinsically semi-supervised methods for deep learning simultaneously employ labelled and unlabelled data to optimise the network parameters.

Ladder networks (Rasmus et al., 2015) are based on denoising autoencoders. The encoder is employed as a classifier for the evaluation of the loss for labelled data. Layer-wise reconstruction is used for the loss of unlabelled data. The network parameters are optimised using the sum of both the losses. An empirical study (Pezeshki et al., 2016) of ladder networks reveals that the introduction of noise and reconstruction of the first layer has a substantial impact on the overall performance of the ladder networks.

One type of intrinsically semi-supervised method is based on introducing small perturbations either on the input sample or the classifier itself. Perturbation-based methods rely on the smoothness assumption of semi-supervised learning. Consequently, small perturbations should not change the model's prediction on the original and perturbed input. This enforces the consistency of the predictions.

Laine and Aila (2017) introduced perturbation in neural networks directly. The dropout (Srivastava et al., 2014) method is used as a perturbation method. Additionally, Gaussian noise and augmentations are also applied to inputs.

Training examples are passed to the neural networks twice, resulting in two outputs. The squared difference between the activations of the final layer of both outputs is used as a loss for unlabelled data. The weight of the unlabelled loss starts at zero and gradually increases as training progresses. This model is named the Π-model. To overcome the problem of performing network evaluation twice and the issue of noisy labels, Laine and Aila (2017) propose Temporal ensembling as an improvement to the Π-model. Temporal ensembling maintains an exponential moving average (EMA) of the predictions of unlabelled examples and updates once after each epoch. The mean square difference between these EMA predictions and the current model's predictions of unlabelled examples is used as an unsupervised loss. As each EMA prediction is updated only once per epoch, the learned information is incorporated into the training process at a slow pace. The larger the dataset, the longer the span of the updates. To overcome the limitation of temporal ensembling, mean teacher (Tarvainen and Valpola, 2017) uses averaging of the model weights instead of averaging the predictions. The teacher model uses EMA weights of the student model and provides predictions for unlabelled data for the student. The student model uses the usual softmax cross-entropy loss for labelled data and mean squared error between the current model's predictions and predictions provided by the teacher model.

All methods described above are discriminative in nature. There are other types of methods that can learn to generate data. They can be used for classification purposes when conditioned on a given label. Generative adversarial networks (GANs) (Goodfellow et al., 2014) are unsupervised neural networks trained to generate new samples based on a distribution learned from training examples. Training of GANs is based on an adversarial game, where parameters of two networks, the generator and the discriminator, are updated in a turn-wise fashion. The generator is responsible for generating new samples. On the other hand, the discriminator tries to distinguish between the true samples and the generator's synthetic samples. The generator's learning objective

is to generate nearly realistic images to fool the discriminator. The discriminator's objective is to reflect the probability that the sample comes from the true data distribution. GANs can be used for learning features from unlabeled data in the SSL scenario. One can reuse parameters learned from the discriminator for a classifier. A possible method to build a semi-supervised GANs (Salimans et al., 2016) is to employ a discriminator objective function based $(c+1)$ classes instead of binary classification as in standard GAN, where true samples are classified into c real classes, and generated samples are classified into the $(c+1)$-th class.

Fixmatch (Sohn et al., 2020) is a combination of consistency regularisation and pseudo-labelling. The loss in this method is the weighted sum of supervised and unsupervised loss computed using cross-entropy loss. For supervised loss, provided labels are used for the loss calculation. For unsupervised loss, pseudo-labels are predicted using weak augmentations of the unlabeled examples if the highest probability is greater than a threshold. Then, the loss is computed on K strongly augmented examples using predicted pseudo-labels. The weak augmentations consist of flip-and-shift. RandAugment (Cubuk et al., 2020) and CTAugment (Berthelot et al., 2019) are used for the strong augmentations where a given transformation (e.g., translation, rotation, colour inversion, colour adjustment, etc.) is randomly selected. The magnitude of the transformation is a hyperparameter that is optimised during training. Other important factors in FixMatch are the weight decay regularisation and the learning rate decay schedule.

Chapter 4

Self-training using Deep Metric Learning

Self-training methods are the most basic approach for exploiting unlabelled examples in semi-supervised settings based on wrapper methods (for more, see Section 3.3.1). They consist of a single supervised classifier that is iteratively retrained on both labelled examples and pseudo-labelled examples obtained from previous iterations of the classifier. Self-training is also known as self-learning. It was first proposed by Yarowsky (1995) for word sense disambiguation in text documents. Since then, various applications of self-training have been proposed in the literature. For instance, (Rosenberg et al., 2005; Lee, 2013) developed a self-training approach for image-based object detection problems.

Self-training offers various design choices.

- Selection of pseudo-labels: Typically, the prediction score is used as a confidence measure for the selection of unlabelled examples and selected unlabelled examples are called pseudo-labelled examples. The selection of pseudo-labels has a significant effect on the overall performance of the method.

- Usage of pseudo-labels: The pseudo-labelled examples can be reused as hard labels along with labelled examples, or a weighted sum of the loss

of pseudo-labels can be utilised for training of the model for subsequent iterations.

- Stopping criteria: Self-training can be stopped either if all unlabelled examples are labelled or if adding more pseudo-labels does not provide any performance improvement. The latter approach presupposes the availability of a separate set of labelled data as a validation set that can be used to monitor performance.

This chapter derives a new training method for deep neural networks in the self-training paradigm. Instead of applying the usual approach of learning a direct classification model based on cross-entropy loss, a similarity function using labelled examples is learned. Examples of the same class are considered similar, and those belonging to different classes are considered dissimilar. A loss function designed for metric learning is used for this. The deep features produced by the neural network are called embeddings. The similarity is calculated by applying a standard distance metric such as Euclidean distance on the embedded data. The similarity function parameterised by a neural network attempts to make groups of embeddings in Euclidean space according to the class labels. Consequently, a classifier is used on these learned embeddings to assign class labels to unlabelled examples. After that, confident predictions for unlabelled examples (pseudo-labels) are added to the labelled examples to retrain the neural network iteratively. The motivation for merging pseudo-labels with labelled data is to improve the performance of the model.

The learning of the similarity function can be tackled using metric learning. The task of metric learning is to learn a distance function over data points where similar data points are closer and dissimilar points are far apart. Metric learning produces embeddings and is not capable of producing a predictive score for given examples. Therefore, for selecting pseudo-labels, the distance to the first-nearest neighbour (1-NN) found amongst the labelled example is considered as a confidence score. However, in addition to this simple nearest-neighbour-based approach, another graph-based transductive semi-supervised

method, local learning with global consistency (Zhou et al., 2004) is also considered for estimating the confidence score for the selection of pseudo-labels.

The main objective of this chapter is to experimentally evaluate the training of neural networks using metric learning losses—triplet loss, contrastive loss, ArcFace loss—in self-training settings. Although there have been some attempts at learning semi-supervised metric embeddings (Weston et al., 2012; Hoffer and Ailon, 2017) this is the first attempt to combine metric learning with self-training to our knowledge.

4.1 Siamese Networks

Siamese networks (Bromley et al., 1993) were introduced for signature verification as an image matching problem. Two images are fed to the network for checking similarity. These networks are particularly efficient when a large number of classes with a few labelled instances per class are available (Koch et al., 2015). Siamese networks can be thought of as multiple networks with identical copies of the same function with the same weights. They are employed for training a similarity function given a pair of labelled examples in the simplest case or triplets for advanced cases. A pair can have both examples either from the same class or from different classes. Two input examples are fed to the network for computing the embeddings. The objective of the training is to bring the embeddings of similar examples closer and push embeddings of dissimilar examples farther away. Figure 4.1 depicts[1] an example of Siamese nets where two examples are passed to the network for embeddings which can be employed for loss evaluation.

Different losses are used for training Siamese networks, such as contrastive loss, triplet loss, and ArcFace loss (for details, see Section 2.2). The network parameters are updated according to the loss calculated on embeddings.

[1] `https://github.com/HarisIqbal88/PlotNeuralNet` Library used for generation of figure.

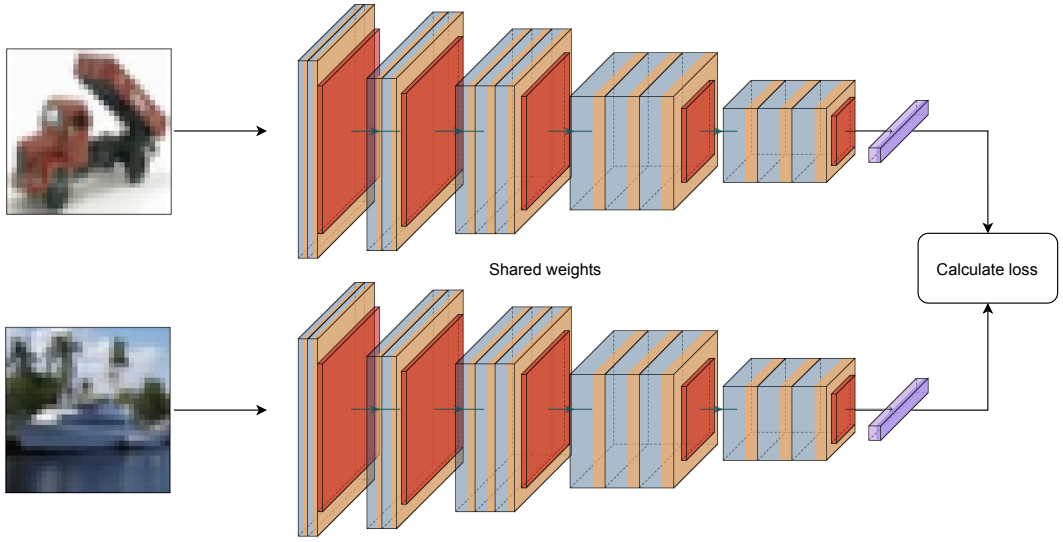


Figure 4.1: Schematic depiction of Siamese networks.

4.1.1 Self-training using Siamese Networks

The approach evaluated in this chapter builds on self-training, where the model is iteratively trained on labelled and pseudo-labelled examples from previous iterations of the model. Algorithm 1 provides an overview of the proposed approach. In the first iteration, to label some of the unlabelled examples, the Siamese network is trained on labelled examples only, using one of the three metric learning loss functions. Then a simple classifier, e.g., 1-nearest-neighbour classifier, is used to predict the labels for the unlabelled examples because learned embeddings are not capable of providing predictions. Following that, a fixed proportion of the top $p\%$ most confidently labelled examples from unlabelled examples are chosen based on a confidence measure and merged into the set of labelled examples for the next iteration. The simplest confidence measure for the selection of pseudo-labelled examples can be based on the distance of unlabelled example's embeddings from labelled examples' embeddings.

Algorithm 1 Proposed approach based on Siamese self-training

Input: Labeled examples (x_L, y_L), unlabelled examples x_U, number of meta-iterations i and selection percentage p
for 1 to i **do**
  $train\_siamesenetwork(x_L, y_L)$
  $embed_U = siamesenetwork(x_U)$
  $embed_L = siamesenetwork(x_L)$
  $labels_U, dist_U = KNN(embed_U, embed_L, y_L)$
  $sorted\_dist_U, sorted\_labels_U = sort(dist_U, labels_U)$
  $x_{new}, y_{new} = select\_top(sorted\_dist_U, sorted\_labels_U, p)$
  $x_L, y_L = concat((x_L, y_L), (x_{new}, y_{new}))$
  $x_U = delete\_from(x_U, x_{new})$
end for

4.2 Local Learning with Global Consistency

An additional confidence measure for the use in the selection of pseudo-labelled examples that we consider in this chapter is based on the graph-based SSL algorithm local learning with global consistency (LLGC) (Zhou et al., 2004). LLGC is a transductive semi-supervised approach (for more, see Section 3.3), which works by propagating label information to the neighbours of an example. The goal of LLGC is to predict labels for unlabelled instances. The algorithm initialises a matrix $Y_{n\times C}$ to represent label information of size n having C number of classes, where $Y_{ij} = 1$ if example i is labelled as j, and otherwise $Y_{ij} = 0$. We introduce a little variation for the unlabelled examples: instead of using $Y_{ij} = 0$ for all j when i is unlabeled, the predicted labels obtained with the nearest-neighbour classifier after training the Siamese network are used.

LLGC is based on calculating an adjacency matrix. This adjacency matrix is then used to establish a matrix S that is applied to update the label probabilities for the unlabelled examples. The adjacency matrix is calculated using Equation 4.1 by employing the embeddings $f(x_i;\theta)$ and $f(x_j;\theta)$ for each

pair of two examples x_i and x_j, obtained from the Siamese network having parameters θ. The parameter σ is a hyperparameter.

$$W_{ij} = \begin{cases} e^{-\sigma \times |f(x_i) - f(x_j)|^2}, & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases} \tag{4.1}$$

The matrix S is computed as:

$$S = D^{-1/2} \times W \times D^{-1/2} \tag{4.2}$$

where D is a diagonal matrix: $D_i = \sum_{j=1}^{n} W_{ij}$. The initial matrix of label probabilities is set to $F(0) = Y$, and the probabilities are updated by:

$$F(t+1) = SF(t) \times \alpha + (1 - \alpha) \times Y \tag{4.3}$$

where $\alpha \in [0, 1)$ is a hyperparameter for controlling the propagation of label information. The above operation is repeated until convergence. Finally, labels for the unlabelled instances are calculated as:

$$y_i = \underset{j \leq C}{\operatorname{argmax}} F_{ij} \tag{4.4}$$

Algorithm 2 provides an overview of the self-training approach based on LLGC. In the first iteration, the Siamese network is trained on labelled examples using triplet loss. Then a simple classifier, e.g., k-nearest-neighbour classifier, is used to predict labels for unlabelled examples because the embeddings provided by the neural network cannot provide predictions. Then, following that, labelled and unlabelled embeddings along with labels are passed to LLGC. After a certain number of iterations of LLGC, a fixed top $p\%$ of unlabelled examples are chosen based on their LLGC score and added to the labelled examples for the next iteration.

Algorithm 2 Proposed approach based on LLGC self-training

Input: Labelled examples (x_L, y_L), unlabelled examples x_U, number of meta-iterations i, selection percentage p, α and σ parameters for LLGC.
for 1 to i **do**
  $train\_siamesenetwork(x_L, y_L)$
  $embed_U = siamesenetwork(x_U)$
  $embed_L = siamesenetwork(x_L)$
  $labels_U = KNN(embed_U, embed_L, y_L)$
  $labels, score = LLGC(embed_L, embed_U, [y_L, labels_U], \sigma, \alpha)$
  $x_{new}, y_{new} = select\_top(score, p, x_U, labels)$
  $x_L, y_L = concat((x_L, y_L), (x_{new}, y_{new}))$
  $x_U = delete\_from(x_U, x_{new})$
end for

4.3 Experiments

For the evaluation of the proposed approach, the five standard image classification datasets discussed in Section 2.4 are considered. A small subset of labelled examples is chosen for all experiments according to standard semi-supervised learning practice, with a balanced number of examples from each class, and the rest are considered unlabelled. Final accuracy is calculated on the predefined test set for each dataset. No data augmentation is applied to either MNIST or Fashion-MNIST. In contrast, small vertical and horizontal translations are applied to SVHN, CIFAR-10, and PlantVillage, and the image intensities are rescaled to fall into the [0, 1] range. The CIFAR-10 and PlantVillage datasets also utilise horizontal flips for data augmentation. Siamese networks are trained using the three metric learning loss functions discussed in Section 2.2 for all five datasets. For a comparison with supervised settings, results based on cross-entropy loss for all five datasets are also reported. The results shown are averaged over three random runs, using a different random initialisation of the Siamese network parameters for each run

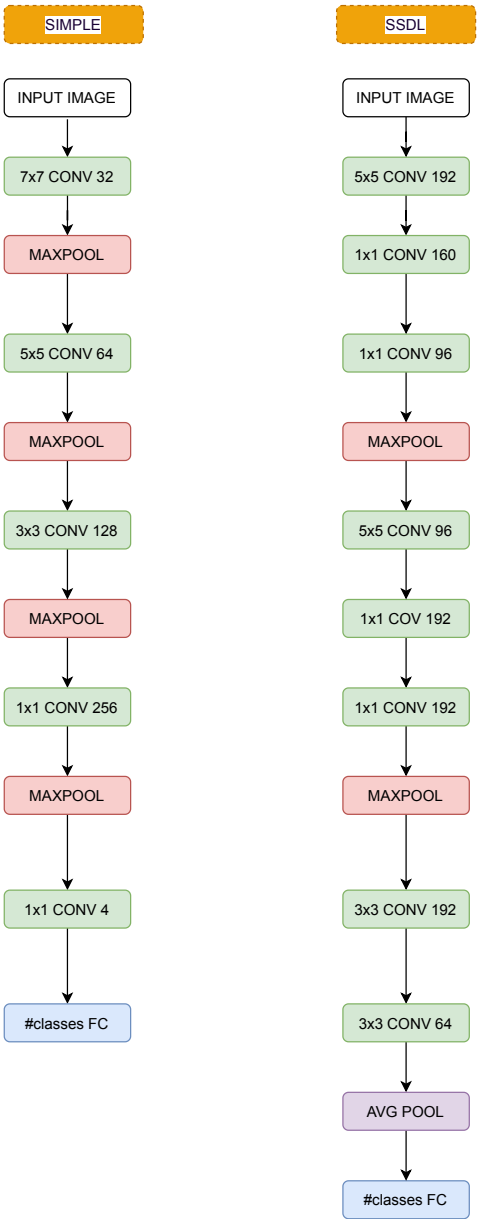


Figure 4.2: Simple custom and SSDL networks architectural diagram.

and a random selection of initially labelled examples. A baseline is provided by (a) training the network with a small number of labelled examples and (b) using all labelled examples. These two baselines should provide good empirical lower and upper bounds for semi-supervised accuracy.

A simple convolutional network architecture is used for each dataset to ensure the performance achieved is due to the proposed method and not the network architecture. For more details about the network architectures, see Figure 4.2. The simple custom model shown in Figure 4.2 is used for MNIST,

Fashion-MNIST, and SVHN and produces 16-dimensional embeddings. In contrast, the SSDL (Hoffer and Ailon, 2017) model is employed for the CIFAR-10 and PlantVillage dataset, which produces 64-dimensional embeddings. The total number of parameters is 163,908 and 693,792 for the custom and the SSDL model, respectively. For the estimation of triplet loss, only semi-hard triplets are considered. The triplet loss and contrastive loss employ $l2$-normalised embeddings for estimation of the loss, and the margin m is set 1.0. The ArcFace loss utilises a margin penalty $m = 0.5$ as per suggestion from (Deng et al., 2019). For the radius of the hypersphere s, we tried different values and found that $s = 30$ produced better results for all datasets. The networks were initially trained using mini-batch sizes 50, 100, and 200, and it was found that batch size 50 was insufficient, and 200 did not yield significant improvements compared to batch size 100. Hence, the batch size of 100 is used for all experiments, with Adam (Kingma and Ba, 2015) (for more information, see 2.3) as the optimiser for updating network parameters in most cases, while RMSProp is used for contrastive loss. Adam optimiser did not provide better test accuracy compared to RMSProp. The learning rate is set to 10^{-4} except for ArcFace loss which uses 10^{-3}. For calculation of test accuracy on all-labelled examples (the full set of labelled examples available for each problem, which provides an upper bound of performance possible using semi-supervised learning), the network is trained for 200 epochs. The proposed approaches, Siamese self-training (Algorithm 1) and LLGC self-training (Algorithm 2), respectively, are run for 25 meta-iterations. For each meta-iteration, the network is trained for 200 epochs. For LLGC, $\alpha = 0.99$ is used in all experiments, while σ is optimised for each dataset, and the network is trained using triplet loss. The summary of hyperparameters is presented in Tables A.1 and A.2. When using LLGC, the confidence score for the selection of pseudo-labels is provided by LLGC. The triplet loss, contrastive loss, and ArcFace loss utilise 1-nearest-neighbour distance as a measure of confidence for the selection of pseudo-labels. At the same time, cross-entropy-based experiments employ the softmax probability

score. The final test accuracy is computed also using a k-NN classifier with $k = 1$ for simplicity[2].

4.3.1 MNIST

For training the network with semi-supervised learning on the MNIST data, only 100 labelled examples (10 from each class) are initially selected. Then the proposed approaches are applied with a selection percentage $p = 10\%$. The LLGC-based method is applied with $\sigma = 2.0$. The results are given in Table 4.1 for test accuracy of the simple custom model trained with cross-entropy, triplet, contrastive, and ArcFace loss as well as LLGC on MNIST after 100-label training, after 25-meta-iterations of self-training, and all-labelled examples. The values in bold represent the highest test accuracy after training on 100-labelled, self-training, and all labelled examples among all losses. Self-training yields noticeable improvements for all losses when compared to labelled training with the same number of labelled examples. The LLGC-baed self-training approach obtains the highest accuracy among all loss functions for the self-training scenario.

Figure 4.3 presents test accuracy after each iteration of self-training based on three different runs of all three metric loss functions on MNIST using 1-NN as a confidence measure and triplet loss with LLGC as a confidence measure for selection of pseudo-labels. The accuracy curves for all losses show a definite improvement in the course of the 25 iterations of self-training.

4.3.2 Fashion-MNIST

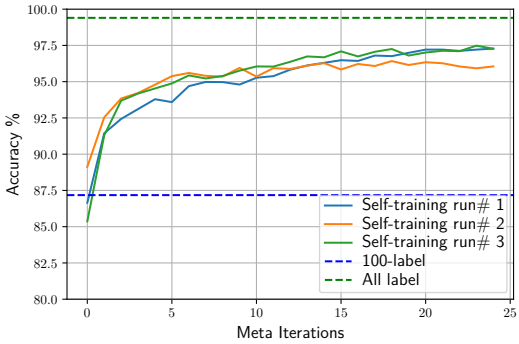
Likewise, for Fashion-MNIST, 100 labelled instances are considered for the training of the neural network. Again, the selection percentage $p = 10\%$ and $\sigma = 3.2$ is used. Table 4.2 depicts the test accuracy of the simple model trained with cross-entropy, triplet, contrastive, and ArcFace loss on Fashion-MNIST

[2]Source code is available at `https://github.com/attaullah/Self-training/blob/master/Metric_learning.md`

Table 4.1: Test accuracy % of the simple custom model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on MNIST after 100-label training, 25-meta iterations of self-training, and all-labelled examples.

Method	**100-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	82.75 ± 1.11	90.19 ± 0.30	98.44 ± 0.22
Triplet (1-NN)	86.96 ± 1.41	94.94 ± 0.63	99.40 ± 0.03
Triplet (LLGC)	86.96 ± 1.41	$\mathbf{96.87 \pm 0.58}$	99.40 ± 0.03
Contrastive	$\mathbf{87.16 \pm 1.30}$	94.55 ± 0.76	98.90 ± 0.24
ArcFace	82.05 ± 2.43	91.21 ± 1.13	$\mathbf{99.41 \pm 0.02}$

(a) Triplet (1-NN)



(b) Triplet (LLGC)

(c) Contrastive

(d) ArcFace

Figure 4.3: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on MNIST.

Table 4.2: Test accuracy % of the simple custom model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on Fashion-MNIST after 100-label training, 25-meta iterations of self-training, and all-labelled examples.

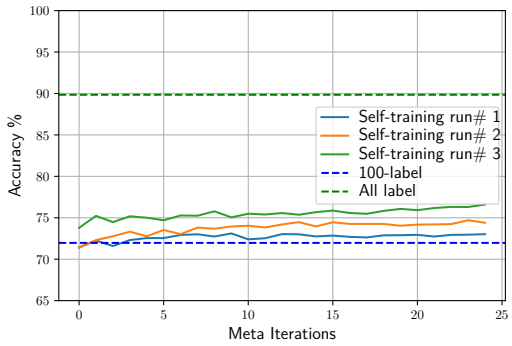
Method	**100-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	69.56 ± 1.64	73.09 ± 0.89	$\mathbf{91.54 \pm 0.38}$
Triplet (1-NN)	$\mathbf{71.97 \pm 1.60}$	$\mathbf{74.67 \pm 1.47}$	89.84 ± 0.21
Triplet (LLGC)	$\mathbf{71.97 \pm 1.60}$	72.58 ± 1.85	89.84 ± 0.21
Contrastive	71.81 ± 0.55	72.96 ± 0.44	89.63 ± 0.71
ArcFace	69.44 ± 1.10	71.11 ± 1.06	90.97 ± 0.31

after 100-label training, 25 meta-iterations of self-training, and all-labelled examples. Self-training achieved a small improvement for all losses compared to labelled training with the same number of labelled examples. Triplet loss based self-training achieved the highest test accuracy among all loss functions.

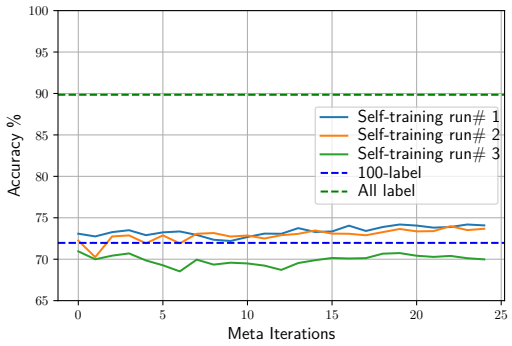
Figure 4.4 exhibits details about the test accuracy after each iteration of self-training across three different runs of all three metric loss functions on Fashion-MNIST using triplet, contrastive, and arcFace loss with 1-NN as a confidence measure and triplet loss with LLGC as a confidence measure for selection of pseudo-labels. At the start, accuracy curves show a slight improvement, and after that, the test accuracy is not improving.

4.3.3 SVHN

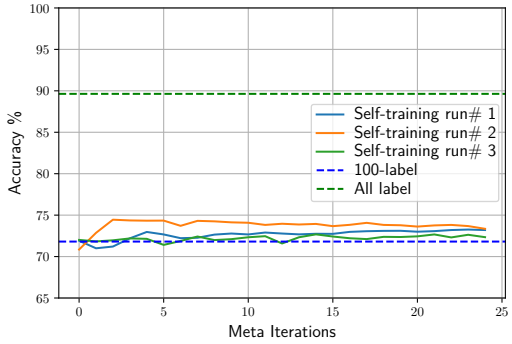
For SVHN, 1000 instances are considered as labelled initially. The selection percentage p is set to 5%, and σ to 2.4. Table 4.3 shows test accuracy of the simple custom model trained with cross-entropy, triplet, contrastive, and ArcFace loss on SVHN after 1000-label training, 25 meta-iterations of self-training, and all-labelled examples. It can be seen that self-training achieves significant improvement over the 1000-labelled examples training for all loss



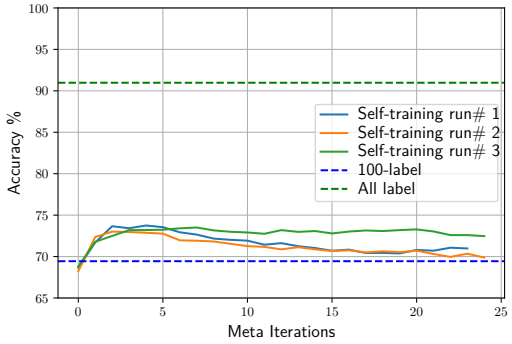
(a) Triplet (1-NN)



(b) Triplet (LLGC)



(c) Contrastive



(d) ArcFace

Figure 4.4: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on Fashion-MNIST.

Table 4.3: Test accuracy % of the simple custom model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on SVHN after 1000-label training, 25-meta iterations of self-training, and all-labelled examples.

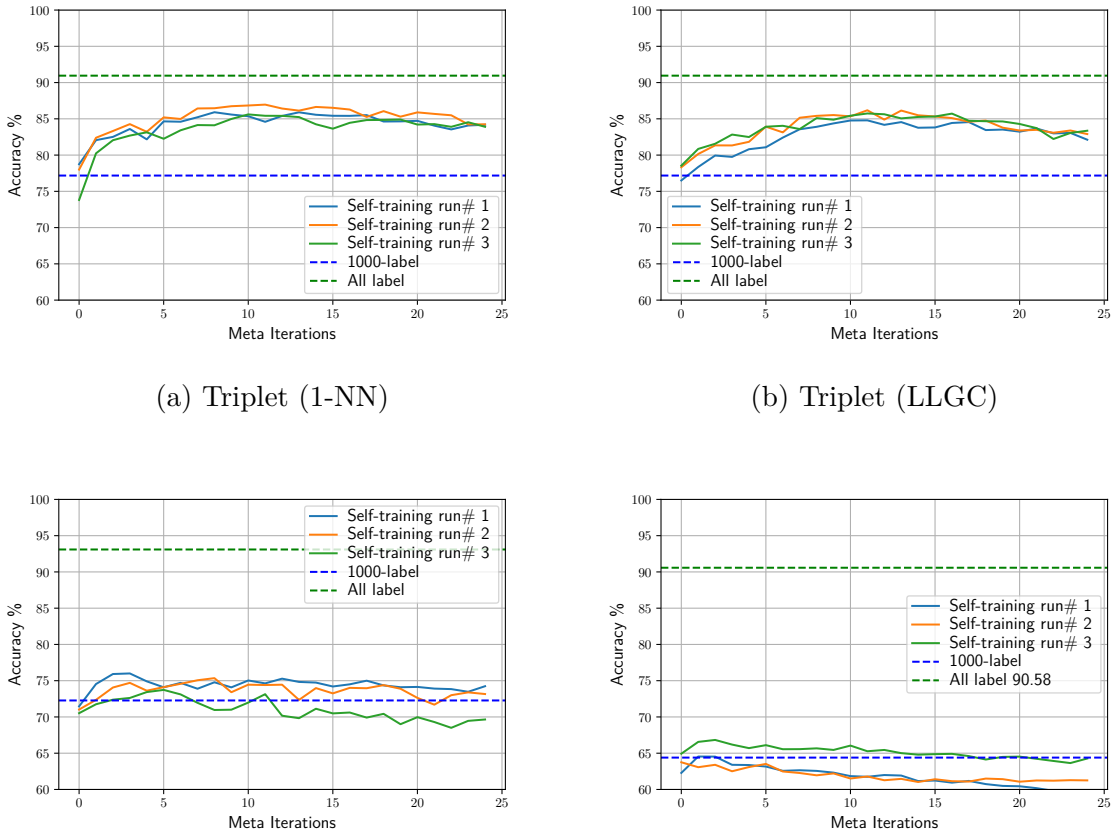
Method	**1000-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	74.66 ± 2.13	$\mathbf{85.60 \pm 2.58}$	92.15 ± 1.25
Triplet (1-NN)	$\mathbf{77.18 \pm 2.58}$	84.11 ± 0.15	90.95 ± 0.76
Triplet (LLGC)	$\mathbf{77.18 \pm 2.58}$	82.79 ± 0.51	90.95 ± 0.76
Contrastive	72.27 ± 2.52	72.35 ± 1.96	$\mathbf{93.09 \pm 0.54}$
ArcFace	64.39 ± 1.96	61.76 ± 1.88	90.59 ± 0.03

functions. The self-training approach based on cross-entropy loss achieves the highest test accuracy. Among metric loss functions, self-training utilising triplet loss achieves the highest test accuracy. The contrastive loss achieves minor improvement while ArcFace yields performance degradation.

After each iteration of self-training, the test accuracy across three different runs of all three metric loss functions are shown in Figure 4.5. It is evident that the test accuracy curves show significant improvement over the iterations for triplet loss and LLGC, while contrastive loss achieves negligible improvement and ArcFace shows performance degradation.

4.3.4 CIFAR-10

The proposed semi-supervised approaches are evaluated using 4000 labelled instances initially, with a selection percentage $p = 5\%$, and $\sigma = 2.4$. Table 4.4 shows test accuracy of the SSDL model trained with cross-entropy, triplet, contrastive, and ArcFace loss on CIFAR-10 after 4000-label training, 25 meta-iterations of self-training, and all-labelled examples. It is evident that self-training achieves significant improvement regardless of loss functions, while triplet loss achieves the highest test accuracy.



(a) Triplet (1-NN)

(b) Triplet (LLGC)

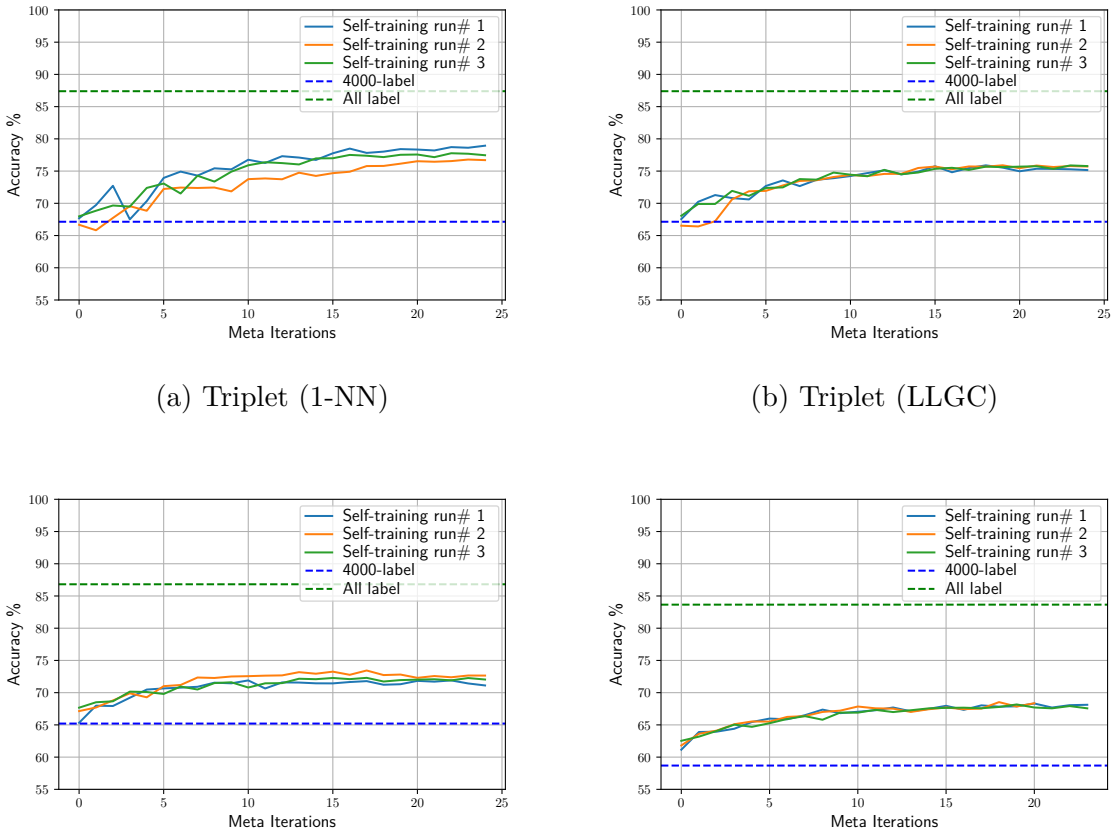
(c) Contrastive

(d) ArcFace

Figure 4.5: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on SVHN.

Table 4.4: Test accuracy % of SSDL model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on CIFAR-10 after 4000-label training, 25-meta iterations of self-training, and all-labelled examples.

Method	**4000-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	65.94 ± 1.89	76.02 ± 1.82	85.64 ± 2.79
Triplet (1-NN)	$\mathbf{67.14 \pm 1.49}$	$\mathbf{77.70 \pm 0.93}$	$\mathbf{87.39 \pm 0.59}$
Triplet (LLGC)	$\mathbf{67.14 \pm 1.49}$	75.55 ± 0.28	$\mathbf{87.39 \pm 0.59}$
Contrastive	65.21 ± 0.64	71.94 ± 0.63	86.82 ± 0.41
ArcFace	58.69 ± 0.74	67.96 ± 0.40	83.66 ± 0.12



(a) Triplet (1-NN)

(b) Triplet (LLGC)

(c) Contrastive

(d) ArcFace

Figure 4.6: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on CIFAR-10.

Figure 4.6 shows a test accuracy after each iteration of self-training across three different runs for all three metric loss functions on CIFAR-10. The test accuracy curves show consistent improvement for a few initial iterations and remain unchanged for the rest of the iterations.

4.3.5 PlantVillage

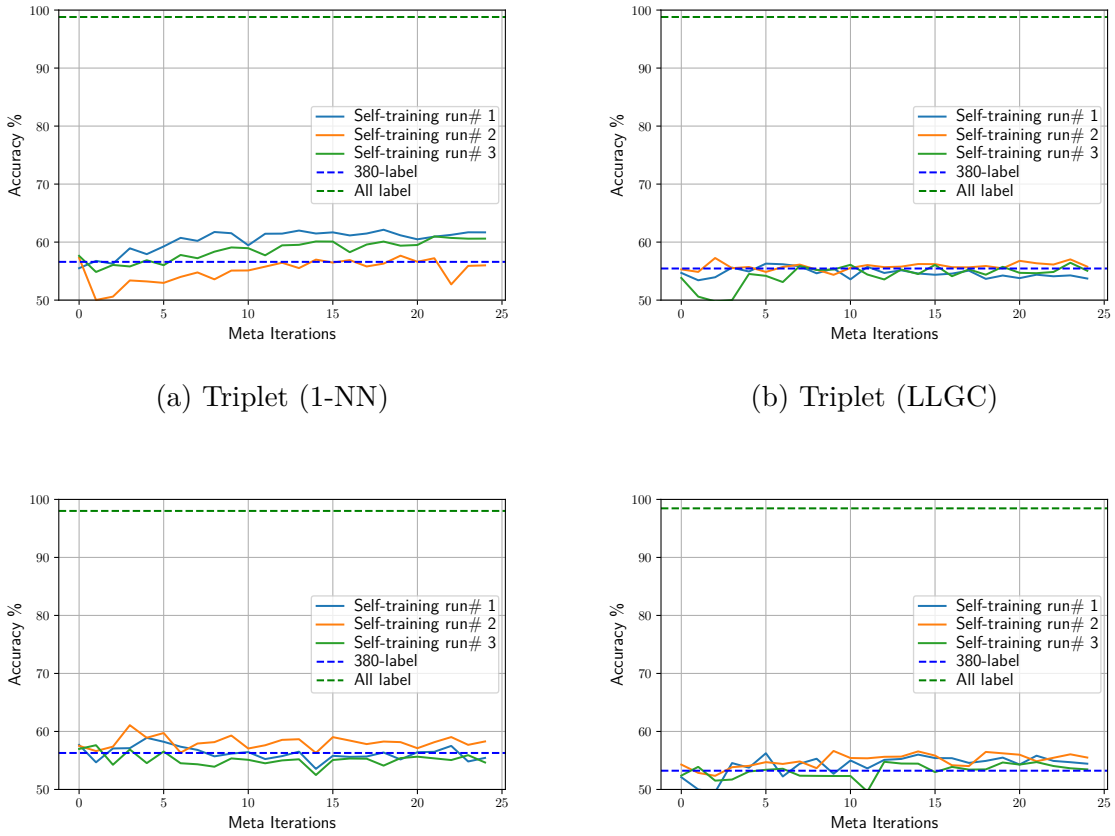
For PlantVillage, all three image sizes, i.e., 32 by 32, 64 by 64, and 96 by 96, are considered for evaluation purpose. Initially, 380 images (10 images per class) are considered as labelled instances, with a selection percentage $p = 2\%$, and $\sigma = 2.4$. Tables 4.5, 4.6, and 4.7 show test accuracy of the SSDL model trained with cross-entropy, triplet, contrastive, and ArcFace loss on PlantVillage after

Table 4.5: Test accuracy % of SSDL model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on PlantVillage32 after 380-label training, 25-meta iterations of self-training, and all-labelled examples.

Method	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	$\mathbf{60.98 \pm 1.43}$	58.12 ± 1.40	97.46 ± 1.66
Triplet (1-NN)	55.45 ± 1.47	$\mathbf{60.87 \pm 0.59}$	$\mathbf{98.80 \pm 0.02}$
Triplet (LLGC)	55.45 ± 1.44	54.85 ± 0.85	$\mathbf{98.80 \pm 0.02}$
Contrastive	56.29 ± 0.88	56.12 ± 1.56	98.02 ± 0.06
ArcFace	53.23 ± 1.05	54.47 ± 0.83	98.46 ± 0.12

380-label training, 25 meta-iterations of self-training, and all labelled examples with sizes 32 by 32, 64 by 64 and 96 by 96, respectively. Interestingly, self-training based on cross-entropy loss suffers performance degradation on all image size variations, i.e., 32, 64, and 96. The PlantVillage dataset has 38 classes, and the class distribution is skewed (for details, see Section 2.10), which makes it hard to achieve better test accuracy than the baseline with 380 labeled examples. Also, the network is probably overfitting to training examples as only a small number of labelled examples are available initially. All metric learning losses generally achieve a small improvement for self-training over the 380-labelled baseline. The self-training achieves the highest accuracy using triplet loss for PlantVillage 32 by 32 and 96 by 96, while contrastive loss achieves the highest accuracy on 64 by 64.

Figures 4.7, 4.8, and 4.9 shows the test accuracy after each iteration of self-training across three different runs of all loss functions on PlantVillage 32, 64, and 96, respectively. The accuracy curves show a small improvement on all metric loss functions.



(a) Triplet (1-NN)

(b) Triplet (LLGC)

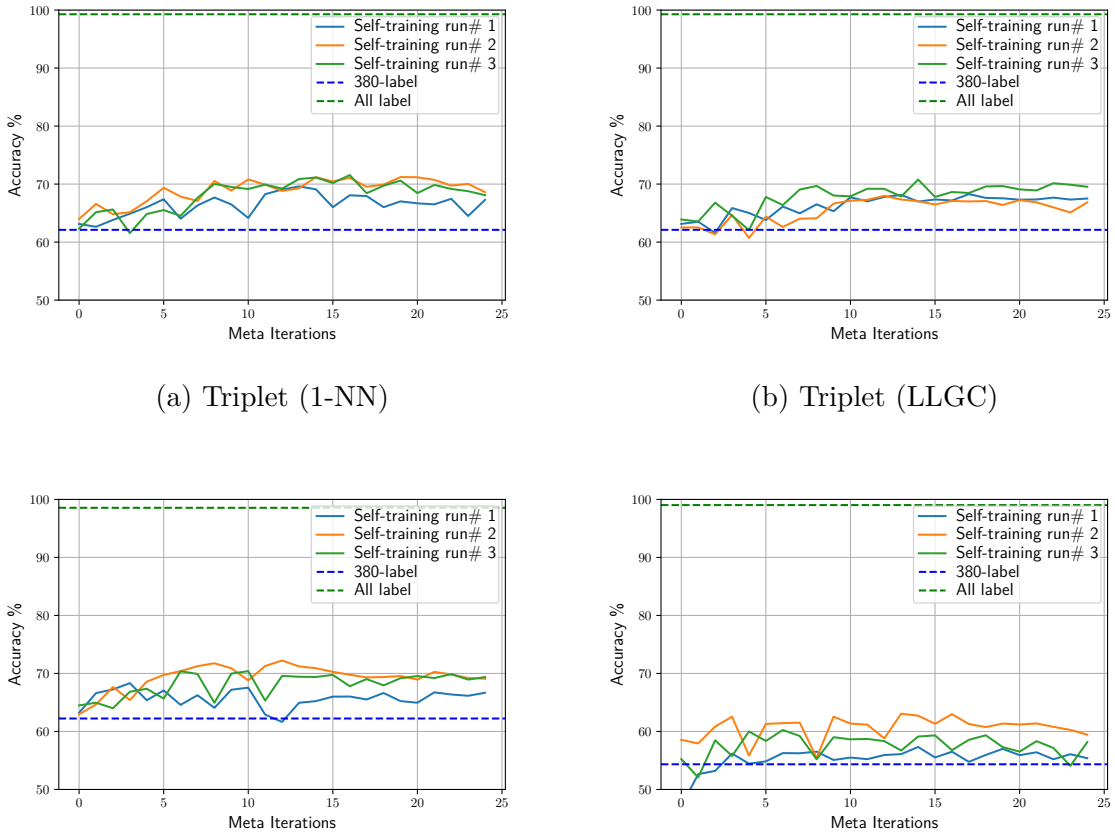
(c) Contrastive

(d) ArcFace

Figure 4.7: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on PlantVillage32.

Table 4.6: Test accuracy % of SSDL model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on PlantVillage64 after 380-label training, 25-meta iterations of self-training, and all-labelled examples.

Method	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	**67.34 ± 2.44**	63.47 ± 0.55	**99.36 ± 0.12**
Triplet (1-NN)	63.78 ± 2.03	67.99 ± 0.54	99.28 ± 0.01
Triplet (LLGC)	63.78 ± 2.03	67.96 ± 1.15	99.28 ± 0.01
Contrastive	62.23 ± 1.79	**68.38 ± 1.21**	98.56 ± 0.02
ArcFace	54.34 ± 3.19	57.66 ± 1.69	99.02 ± 0.37



(a) Triplet (1-NN)

(b) Triplet (LLGC)

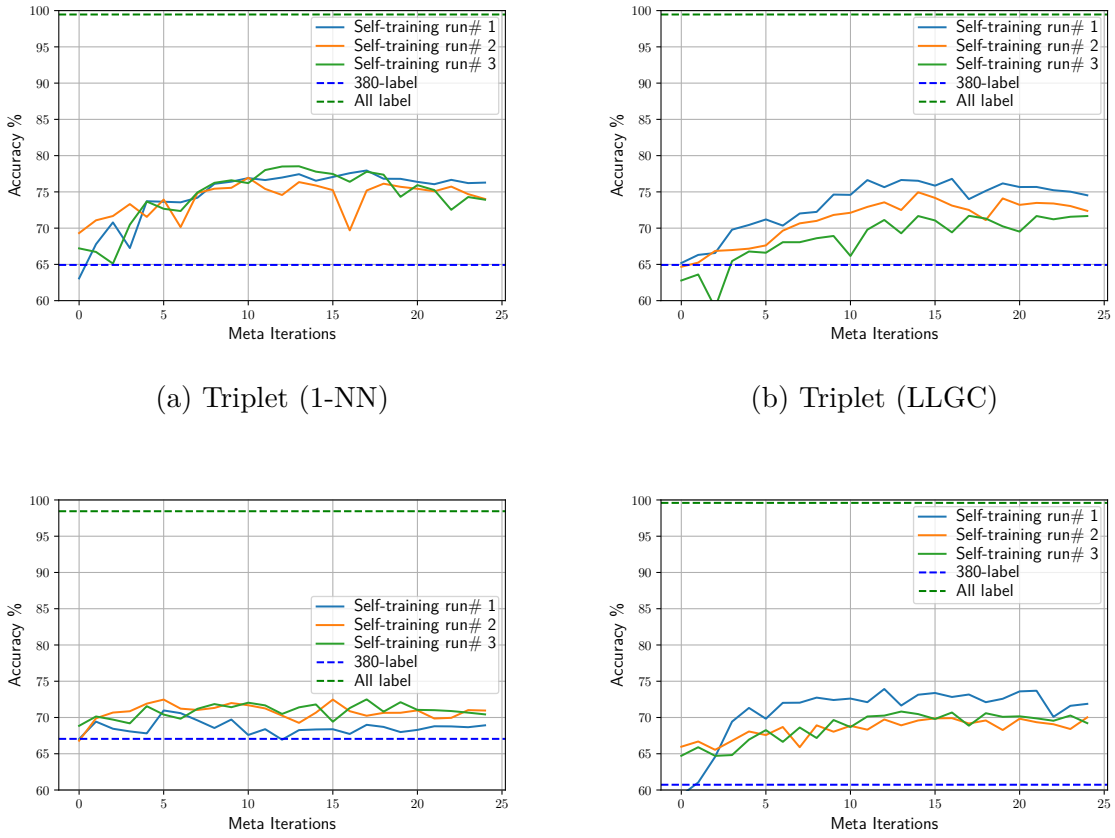
(c) Contrastive

(d) ArcFace

Figure 4.8: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on PlantVillage64.

Table 4.7: Test accuracy % of SSDL model trained with Cross-entropy, Triplet, Contrastive, and ArcFace loss on PlantVillage96 after 380-label training, 25 meta-iterations of self-training, and all-labelled examples.

Method	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	**67.66 ± 1.34**	64.34 ± 1.65	98.02 ± 0.57
Triplet (1-NN)	64.93 ± 2.51	**73.80 ± 0.24**	99.46 ± 0.14
Triplet (LLGC)	64.93 ± 2.51	72.86 ± 1.22	99.46 ± 0.14
Contrastive	67.05 ± 0.95	70.10 ± 0.86	98.44 ± 0.02
ArcFace	60.73 ± 5.49	70.37 ± 1.11	**99.60 ± 0.04**



(a) Triplet (1-NN)

(b) Triplet (LLGC)

(c) Contrastive

(d) ArcFace

Figure 4.9: Comparison of self-training using Triplet, Triplet with LLGC, Contrastive, and ArcFace loss on PlantVillage96.

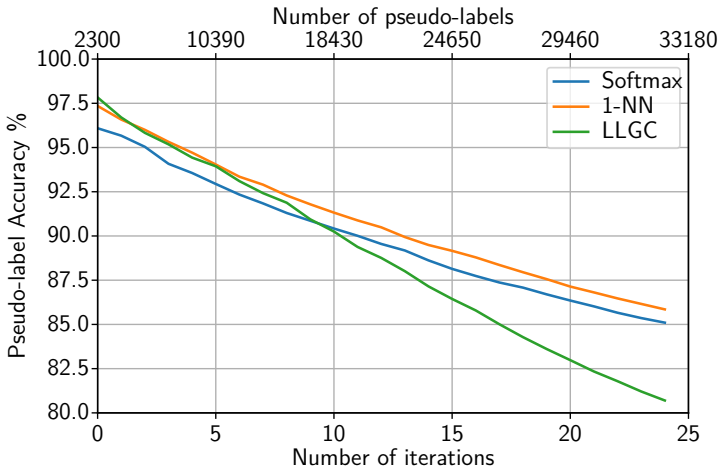


Figure 4.10: Comparison of accuracy of selected pseudo-labelled examples based on softmax score, 1-NN, and LLGC on CIFAR-10.

4.3.6 Comparison of Confidence Measures

To compare confidence measures' performance on the selection of pseudo-labelled examples, self-training is applied to CIFAR-10 using the SSDL network. Initially, 4000 examples are considered labelled and 46000 unlabelled. After each iteration of self-training, $p = 5\%$ pseudo-labelled examples are selected from the unlabelled set and merged into the labelled set for retraining.

For cross-entropy loss, the softmax score is considered as a confidence measure. For triplet loss, the distance from the unlabelled examples to the nearest labelled example (1-NN) and the LLGC prediction score are considered as confidence measures. Figure 4.10 compares the accuracy of pseudo-labelled examples across 25 iterations of self-training and the number of pseudo-labels. The selection accuracy of pseudo-labels decreases over iterations of self-training as easy pseudo-labelled examples are added and remaining unlabelled examples become hard. Softmax and 1-NN consistently and gradually decrease accuracy, while the LLGC-based confidence measure shows a rapid decrease in accuracy of pseudo-labels after ten iterations, which is also reflected in test accuracy (see Table 4.4).

Table 4.8: Test accuracy after self-training using various loss functions.

	Cross-entropy	**Triplet(1-NN)**	**Triplet(LLGC)**	**Contrastive**	**ArcFace**
MNIST	90.19 ± 0.30	94.94 ± 0.63	$\mathbf{96.87 \pm 0.58}$	94.55 ± 0.76	91.21 ± 1.13
Fashion-MNIST	73.09 ± 0.89	$\mathbf{74.67 \pm 1.47}$	72.58 ± 1.85	72.96 ± 0.44	71.11 ± 1.06
SVHN	$\mathbf{85.60 \pm 2.58}$	84.11 ± 0.15	82.79 ± 0.51	72.35 ± 1.96	61.76 ± 1.88
CIFAR-10	76.02 ± 1.82	$\mathbf{77.70 \pm 0.93}$	75.55 ± 0.28	71.94 ± 0.63	67.96 ± 0.40
PLANT32	58.12 ± 1.40	$\mathbf{60.87 \pm 0.59}$	54.85 ± 0.85	56.12 ± 1.56	54.47 ± 0.83
PLANT64	63.47 ± 0.55	67.99 ± 0.54	67.96 ± 1.15	$\mathbf{68.38 \pm 1.21}$	57.66 ± 1.69
PLANT96	64.34 ± 1.65	$\mathbf{73.80 \pm 0.24}$	72.86 ± 1.22	70.10 ± 0.86	70.37 ± 1.11

4.3.7 Comparison of Loss Functions

Table 4.8 summarises the test accuracy of proposed self-training approaches for all datasets using various loss functions. In the majority of cases, metric learning using triplet loss, applied with the 1-NN classifier, performs substantially better than cross-entropy loss with soft-max. Among metric learning losses, triplet loss using 1-NN as a confidence measure emerges as the winner.

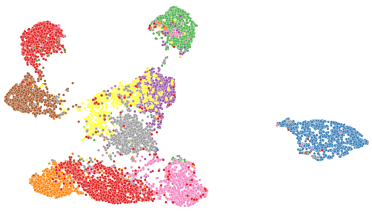
4.3.8 Visualisations of Embeddings

We can also attempt to visualise the quality of embeddings learned after training the network. For dimension reduction, UMAP (Uniform Manifold Approximation and Projection) (McInnes et al., 2018) is employed. UMAP is a dimension reduction technique that can be used for visualisation and general non-linear dimension reduction. For visualisation, the simple custom model 4.2 is trained on MNIST and produces 16-dimensional embeddings. Then UMAP is applied in an unsupervised fashion on the 16-dimensional embeddings to produce 2-dimensional output. Figure 4.11 depicts the UMAP output for test instances marked in colour according to their true class after 100-label training the network on the left; the right side depicts the UMAP output for test instances after self-training with cross-entropy, triplet, contrastive, and ArcFace loss. It can be seen that the 10000 test instances' embeddings are scattered mainly randomly throughout the 2D space before the network is trained. In

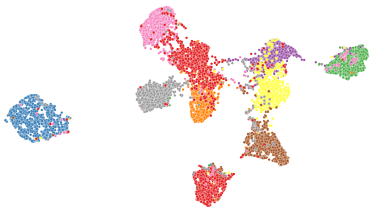
contrast, after training of the network, the test instances' embeddings form clusters in Euclidean space according to the class labels.

4.4 Discussion

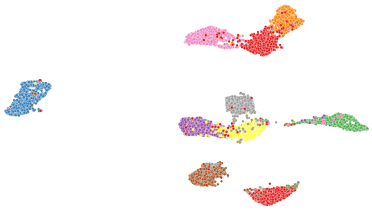
This chapter shows that neural networks can be trained in a self-training setting using a small number of labelled examples by replacing the classification objective with a loss function designed for metric learning. This metric learning based training objective is compatible with standard techniques of training the deep neural network and requires no modification of the network model. The proposed approach is conceptually simple and easy to implement. It achieves significant improvement compared to training on a small subset of labelled examples on five different datasets using several metric learning loss functions. For self-training, triplet loss with 1-NN outperforms the widely used cross-entropy loss on four out of seven datasets. Among metric learning losses, triplet loss emerges as the winner. For the selection of pseudo-labelled examples, we considered 1-nearest-neighbour distance and LLGC score as a confidence measure. In most cases, using 1-nearest-neighbour distance achieved slightly higher accuracy than the LLGC score when used as a confidence measure. For improving the results further, the next chapter will explore the effect of using bigger network models with pretrained weights.



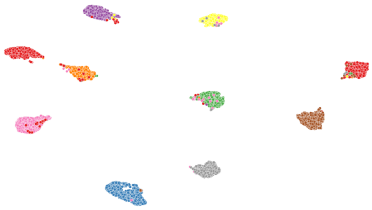
(a) CE: after 100-label training



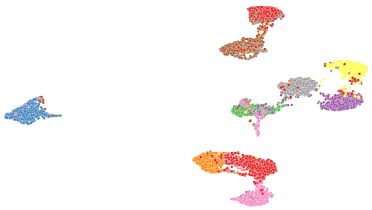
(b) CE: after self-training



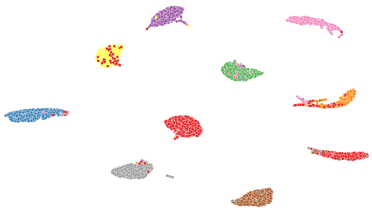
(c) Triplet: after 100-label training



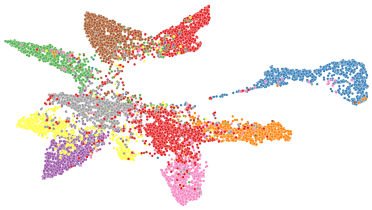
(d) Triplet: after self-training



(e) Contrastive: after 100-label training



(f) Contrastive: after self-training



(g) ArcFace: after 100-label training

(h) ArcFace: after self-training

Figure 4.11: UMAP visualisation of MNIST test set embeddings for all losses before and after the training on 100-labelled examples.

Chapter 5

Self-training using Deep Transfer Learning

In the previous chapter, the primary motivation was to investigate the performance of metric learning in a self-training setting using simple convolutional neural networks. This chapter considers more sophisticated and deeper convolutional models with pretrained weights from ImageNet.

Deep neural networks require many labelled examples for training compared to traditional machine learning methods. The collection of labelled examples can be costly and extremely difficult to obtain in some cases, e.g., clinical trials, etc. In these scenarios, transfer learning can be very beneficial. It aims to transfer the knowledge from the source domain to the target domain. The survey in (Tan et al., 2018) divides the transfer learning methods into four categories.

1. Instances-based methods: some instances from the source domain are utilised for training in the target domain with appropriate weights.

2. Mapping-based methods: source and target domains are mapped into a new data space. The resulting data spaces are learned by minimising the distance between data distributions of source and target domains.

3. Network-based methods: aims to reuse some layers of the network pre-

trained in the source domain, including its network structure and parameters.

4. Adversarial-based methods: an adversarial layer tries to discriminate between features of the source and the target domain—this layer serves as a domain adaptation regularisation term in the loss function.

This chapter explores the network-based transfer learning method, which has been applied successfully in many practical applications of standard supervised learning of deep neural networks.

Since the successful ImageNet challenge (Russakovsky et al., 2015), the network-based transfer learning method has been used widely in visual recognition tasks such as object detection (Girshick et al., 2014). Commonly, network-based transfer learning uses the network weights learned by training on the large and labelled ImageNet dataset and fine-tunes the weights for the respective target domain. The common practice is to reuse the network front layers before the fully-connected layers. This layer is called the bottleneck layer, which contains few nodes compared to the previous layers. The bottleneck layer features retain more generality compared to the final layer. When the target domain is sufficiently closely related to the source domain of ImageNet, then transfer learning generalises much better than training from scratch on the smaller target domain alone(see, for example, (Oquab et al., 2014)).

Typically, one or more fully-connected layers are added after the bottleneck layer. One can use pretrained features as fixed and update parameters of the network for newly added layers only or fine-tune the whole network for the target domain. It has been found that fine-tuning usually provides higher gains than fixed ImageNet features (Kornblith et al., 2019). For transfer learning, all the results reported in this chapter are based on fine-tuning.

The main contribution of this chapter is that it provides an extensive empirical investigation of transfer learning in the context of self-training. By studying different combinations of similarity metric learning losses (e.g., triplet

loss, contrastive loss, and ArcFace loss), we find that transfer learning always substantially improves the classification accuracy of the model when few labelled examples are available, regardless of which loss function is used for training the neural network. More specifically, for semi-supervised learning using self-training on the SVHN, CIFAR-10, and PlantVillage image classification datasets, we obtain a substantial improvement using pretrained weights when few labelled examples are available for training. Thus, our results indicate that the well-established method of performing transfer learning by reusing pretrained weights—commonly applied when performing a purely supervised training of a neural network—is particularly useful in the context of semi-supervised learning. We also pretrain a network from scratch on different image sizes of ImageNet using the usual cross-entropy loss and triplet loss and investigate the effect on accuracy for various datasets after transfer learning.

5.1 Self-training using Transfer Learning

Self-training using transfer learning builds on the technique introduced in the previous chapter, where the model is initially trained on limited data. However, in this chapter, instead of applying random initialisation of network parameters when training starts, we investigate using pretrained weights from another domain. We show that this provides a much better generalisation ability. Using pretrained model weights is a standard approach for transfer learning in supervised settings. Still, it appears to have received little attention in the context of semi-supervised learning, particularly when applying self-training to metric learning.

A schematic overview of the proposed approach is shown in Figure 5.1. We use a pretrained neural network model trained on ImageNet. Fine-tuning on data from the target domain is performed on the very small initial set of labelled examples. Following that, confident predictions for unlabelled examples are added to labelled examples for iterative retraining of the neural

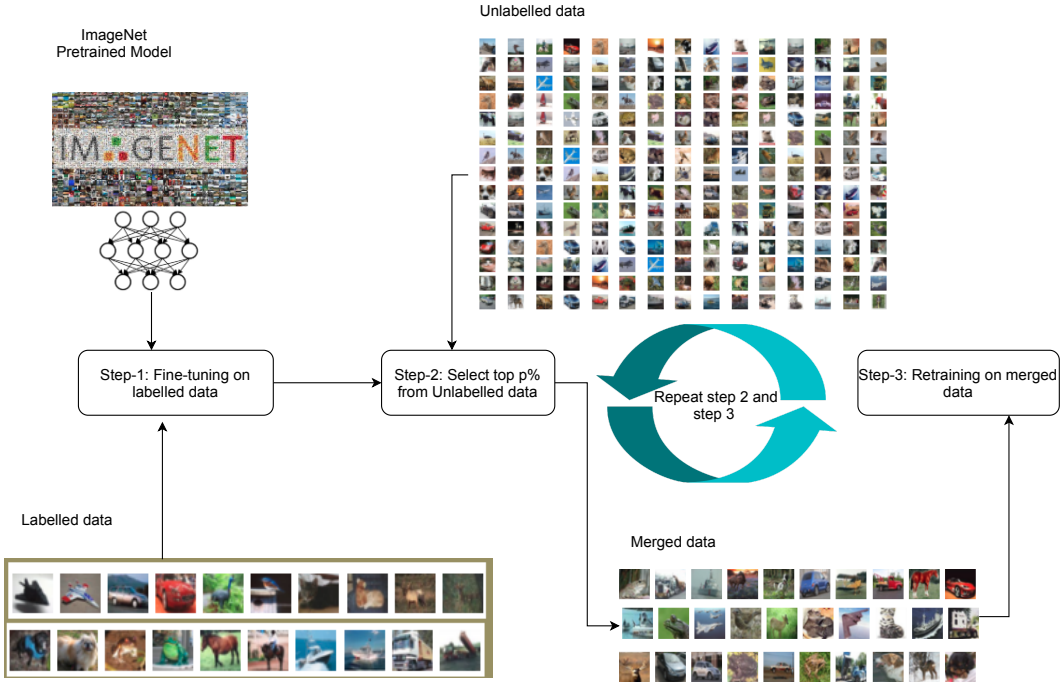


Figure 5.1: An overview of the proposed Transfer Learning approach.

network—this is the standard self-training method for semi-supervised learning. It enables us to obtain more training data, and the assumption is that this eventually helps achieve significant performance improvements.

5.2 Experiments

For evaluating the effect of transfer learning, we follow the same protocol as in the previous chapter. A small subset of labelled examples is chosen for all datasets according to standard semi-supervised learning practice, with a balanced number of examples from each class. All remaining examples are used as unlabelled training examples.

For transfer learning, we consider the VGG16 network with the publicly available checkpoint (Abadi et al., 2016) having top-1 ImageNet (image size 224) validation accuracy of 71.30%. We use VGG16 without fully-connected layers (for details, see Section 2.1.3), which is a common approach when using a pre-trained network as a feature extractor. We always include two network versions in the comparison: one using randomly initialised weights and using

pretrained weights from ImageNet.

A fully connected layer is added at the end of the VGG16 model for generating a 64-dimensional embedding space for fine-tuning the network. A mini-batch size of 100 is used for all experiments. Adam is used as the optimiser for updating the network parameters, except for the contrastive loss, which uses RMSProp. For the triplet, contrastive and ArcFace loss, the 1-nearest neighbour is used for label prediction. In self-training, for triplet, contrastive, and ArcFace loss, the distance to the nearest labelled example is used as the confidence measure when selecting unlabelled examples for labelling. For softmax cross-entropy loss, the softmax probability score is used as the confidence score. The self-training approach is run for 25 meta-iterations, and results were averaged over three runs with a random selection of initially labelled examples.

As in the previous chapter, models are evaluated on the predefined test split for each dataset in three different ways: after training only on the initially labelled examples, after training for several meta-iterations of the proposed self-training approach, and after training on all labelled training examples[1]. The two sets of extreme results computed from a) only the initial labelled training examples and b) all labelled training examples act as a form of empirical lower and upper bounds for the semi-supervised approach.

5.2.1 SVHN

On SVHN, the proposed approach is evaluated using 1000 labelled instances initially with a selection percentage of $p = 5\%$ (i.e., in each meta-iteration of self-training, 5% of the remaining unlabelled examples are selected for labelling). Table 5.1 shows test accuracy for SVHN using all four losses after training on 1000-labelled, self-training, and all-labelled examples with random and pretrained weights for the network. As we can see from the results for

[1]Source code is available at `https://github.com/attaullah/Self-training/blob/master/Transfer_learning.md`

Table 5.1: Test accuracy % of VGG16 network trained on SVHN using random and ImageNet pretrained weights.

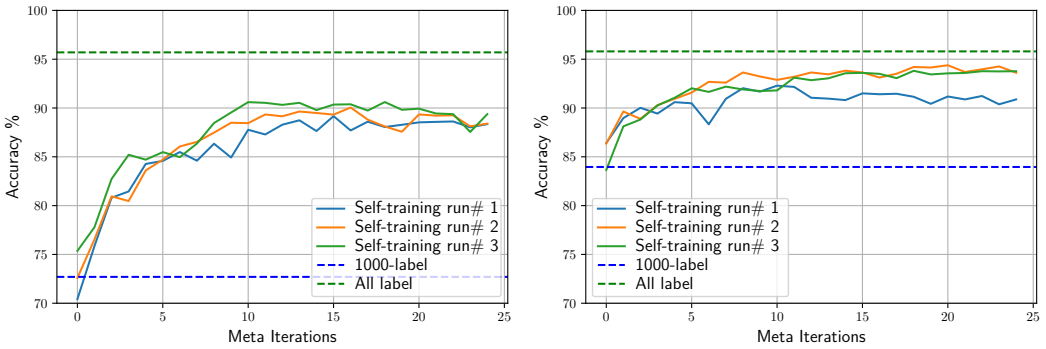
Loss	**Weights**	**1000-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	Random	72.70 ± 2.74	88.71 ± 0.47	95.70 ± 0.14
	ImageNet	$\mathbf{83.96 \pm 2.68}$	$\mathbf{92.75 \pm 1.33}$	$\mathbf{95.80 \pm 0.18}$
Triplet	Random	74.14 ± 6.15	85.63 ± 0.43	95.28 ± 0.03
	ImageNet	81.52 ± 0.91	88.39 ± 0.07	95.62 ± 0.02
Contrastive	Random	62.16 ± 1.45	72.73 ± 2.66	92.69 ± 0.25
	ImageNet	82.88 ± 0.82	79.31 ± 1.19	94.69 ± 0.86
ArcFace	Random	71.23 ± 0.80	71.61 ± 0.40	93.77 ± 0.03
	ImageNet	80.57 ± 0.31	79.71 ± 1.49	95.75 ± 0.04

all four losses, using pretrained weights from ImageNet generally results in substantial improvements over the random initialisation of the weights. When comparing all four loss functions, cross-entropy emerges as the clear winner. For the All-labelled examples setting, using pretrained weights only provides for small increases in test accuracy over fully training from scratch.

Figure 5.2 shows details about the test accuracy after each iteration of self-training across three different runs of cross-entropy loss using a) randomly initialised weights b) ImageNet pretrained weights on VGG16 for SVHN. Initially, 1000 labelled examples are selected randomly for self-training. The accuracy curves show improvements for both scenarios, with the pretrained version starting from a higher initial accuracy level and retaining this advantage over the 25 meta-iterations of self-training.

5.2.2 CIFAR-10

On CIFAR-10, the proposed self-training approach is evaluated using 4000 labelled instances initially, with a selection percentage of 5%. Table 5.2 shows



(a) Randomly initialised weights (b) ImageNet pretrained weights

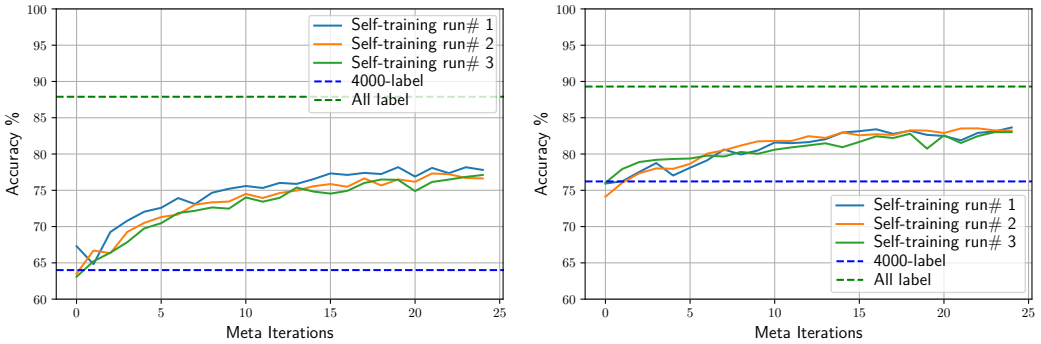
Figure 5.2: Comparison of self-training for SVHN using cross-entropy loss with random and pretrained ImageNet weights of VGG16.

accuracy on the test set for all losses after training on 4000-labelled examples, self-training, and all labelled examples employing random initialisation of the weights of the neural network as well as pretrained weights from ImageNet. Training using pretrained weights from ImageNet always results in substantial improvements over the random initialisation of the network weights. Again, cross-entropy achieves the highest test accuracy after training on a few labelled examples, self-training, and all labelled examples using random as well as pretrained weights. For pretrained weights, the triplet loss seems competitive with cross-entropy for all three cases, i.e., after training on 4000-labelled examples, self-training, and all labelled examples. On the other hand, contrastive and ArcFace loss suffer from performance degradation on self-training compared to the baseline.

Figure 5.3 shows details about the test accuracy after each iteration of self-training across three different runs of cross-entropy loss using a) randomly initialised weights b) ImageNet pretrained weights on VGG16 for CIFAR-10. Initially, 4000 labelled examples are chosen randomly for each run of self-training. The accuracy curves show improvements for both scenarios, with the pretrained version starting from a higher initial accuracy level and retaining this advantage over the 25 meta-iterations of self-training.

Table 5.2: Test accuracy % of VGG16 network trained on CIFAR-10 using random and ImageNet pretrained weights.

Loss	**Weights**	**4000-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	Random	64.00 ± 0.93	77.18 ± 0.48	87.89 ± 0.18
	ImageNet	$\mathbf{76.22 \pm 0.69}$	$\mathbf{83.30 \pm 0.27}$	$\mathbf{89.30 \pm 0.25}$
Triplet	Random	59.87 ± 0.84	68.31 ± 0.43	85.75 ± 0.32
	ImageNet	75.77 ± 0.90	82.59 ± 0.49	88.87 ± 0.26
Contrastive	Random	60.43 ± 1.03	57.49 ± 2.59	82.12 ± 0.89
	ImageNet	75.54 ± 0.40	74.34 ± 1.43	88.01 ± 0.23
Arcface	Random	55.38 ± 1.24	54.99 ± 1.16	75.50 ± 0.96
	ImageNet	73.90 ± 1.20	75.77 ± 0.61	85.73 ± 0.01



(a) Random weights

(b) Pretrained ImageNet weights

Figure 5.3: Comparison of self-training for CIFAR-10 using cross-entropy loss with random and pretrained ImageNet weights of VGG16.

5.2.3 PlantVillage

For PlantVillage, all three image sizes, i.e., 32 by 32, 64 by 64, and 96 by 96, are considered for evaluation. Initially, 380 images (10 images per class) are considered labelled instances, with a selection of $p = 2\%$ for self-training. Tables 5.3, 5.4 and 5.5 show test accuracy for all four losses using 380-labelled, self-training, and all-labelled examples with randomly initialised network weights and pretrained weights from ImageNet on PlantVillage 32, 64, and 96, respectively. Transfer learning usually provides improvement from the random initialisation of network weights. Again, for self-training, cross-entropy achieves the highest accuracy for all image size variations using random and pretrained weights of the network. For a small number of labelled examples, the triplet loss achieves the highest accuracy using random weights of the network, while the contrastive loss achieves the highest accuracy using ImageNet pretrained weights. On the other hand, all metric loss functions exhibit a significant reduction in accuracy for self-training compared to the cross-entropy baseline for random as well pretrained weights of the network for all image size variations. The reduction in test accuracy is likely due to a large number of classes and the skewed class distribution (for more information, see Section 2.10) of the PlantVillage dataset.

Figure 5.4 shows details about the test accuracy after each iteration of self-training across three different runs of cross-entropy loss using a) randomly initialised weights, and b) ImageNet pretrained weights on VGG16 for PlantVillage32. For PlantVillage32, initially, 380 labelled examples were chosen randomly for each run of self-training. The accuracy curves show improvements for both scenarios, with the pretrained version starting from a higher initial accuracy level and retaining this advantage over the 25 meta-iterations of self-training.

Table 5.3: Test accuracy % of VGG16 network trained on PlantVillage32 using random and ImageNet pretrained weights.

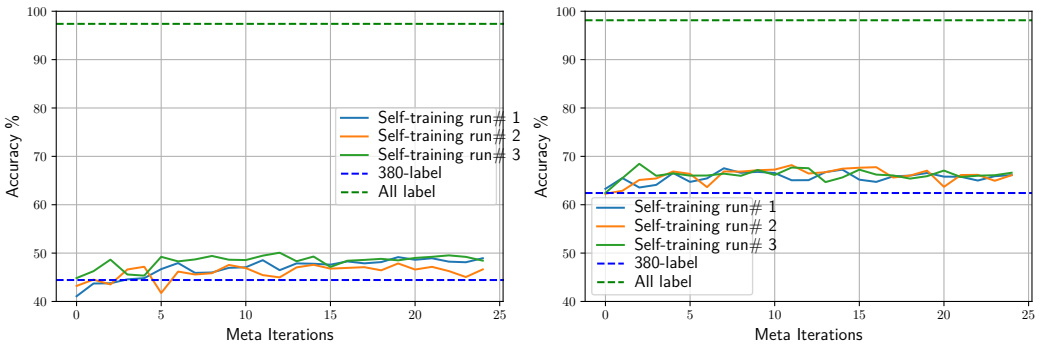
Loss	**Weights**	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	Random	42.81 ± 1.51	57.78 ± 2.74	97.39 ± 0.67
	ImageNet	62.23 ± 1.11	$\mathbf{73.72 \pm 0.58}$	$\mathbf{98.14 \pm 0.27}$
Triplet	Random	55.24 ± 1.28	55.90 ± 0.48	97.18 ± 0.41
	ImageNet	62.80 ± 2.40	59.96 ± 1.88	97.96 ± 0.10
Contrastive	Random	49.53 ± 1.46	48.24 ± 2.14	93.27 ± 0.95
	ImageNet	$\mathbf{64.67 \pm 0.36}$	55.66 ± 0.54	97.34 ± 0.07
ArcFace	Random	49.60 ± 0.98	40.70 ± 0.02	94.92 ± 0.42
	ImageNet	50.66 ± 1.22	52.85 ± 0.49	96.94 ± 0.07

Table 5.4: Test accuracy % of VGG16 network trained on PlantVillage64 using random and ImageNet pretrained weights.

Loss	**Weights**	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	Random	48.65 ± 0.34	65.57 ± 3.88	97.88 ± 1.24
	ImageNet	70.17 ± 1.80	$\mathbf{79.87 \pm 3.94}$	$\mathbf{99.16 \pm 0.16}$
Triplet	Random	59.81 ± 1.69	54.93 ± 1.73	97.68 ± 0.20
	ImageNet	72.97 ± 0.56	66.96 ± 2.04	98.42 ± 0.52
Contrastive	Random	56.59 ± 5.83	53.51 ± 0.79	98.35 ± 0.08
	ImageNet	$\mathbf{77.09 \pm 1.45}$	54.63 ± 1.69	99.13 ± 0.18
ArcFace	Random	45.44 ± 0.92	37.75 ± 5.13	96.98 ± 0.14
	ImageNet	54.63 ± 0.79	63.90 ± 1.59	98.72 ± 0.27

Table 5.5: Test accuracy % of VGG16 network trained on PlantVillage96 using random and ImageNet pretrained weights.

Loss	**Weights**	**380-Labelled**	**Self-training**	**All-Labelled**
Cross-entropy	Random	42.15 ± 2.16	48.37 ± 3.32	98.68 ± 0.25
	ImageNet	73.03 ± 0.77	$\mathbf{78.72 \pm 0.08}$	$\mathbf{99.34 \pm 0.16}$
Triplet	Random	62.36 ± 1.50	61.97 ± 1.88	98.80 ± 0.20
	ImageNet	78.42 ± 1.25	77.10 ± 1.79	99.18 ± 0.33
Contrastive	Random	61.73 ± 1.94	52.04 ± 1.32	98.28 ± 0.77
	ImageNet	$\mathbf{79.67 \pm 2.27}$	62.50 ± 1.99	99.02 ± 0.23
ArcFace	Random	52.48 ± 0.17	47.75 ± 0.96	97.98 ± 0.19
	ImageNet	62.27 ± 3.92	68.70 ± 1.25	99.12 ± 0.08



(a) Random weights (b) Pretrained ImageNet weights

Figure 5.4: Comparison of self-training for PlantVillage32 using cross-entropy loss with random and pretrained ImageNet weights of VGG16.

5.2.4 Impact of Embedding Size and Shallow Classifiers

For investigating the effect of embeddings size and shallow classifiers on test accuracy, various sizes of embeddings and different types of shallow classifiers are compared. The pretrained VGG16 network is employed for training on CIFAR-10 with triplet loss, producing embeddings of sizes 64, 128, and 256. The impact of different embeddings sizes on test accuracy is evaluated by using a k-nearest-neighbour classifier, Linear Discriminant Analysis (LDA), Random Forest (RF), and logistic regression (LR). Logistic regression applies liblinear (Fan et al., 2008) and LBFGS optimisation (Zhu et al., 1997) method. All the classifiers use the sci-kit (Pedregosa et al., 2011) library implementation with default parameters. The k-NN uses $k = 1$ neighbour and Euclidean distance as a metric. Random Forest uses 100 as the default number of trees. LDA uses singular value decomposition (SVD) as a solver. Table 5.6 presents test accuracy and time (millisecond ms, second s) for various embeddings sizes after training on 4000-labelled examples and after training on all labelled examples. LDA is the fastest, and Random Forest is the slowest in terms of running time. All shallow classifiers produced higher accuracy on a small embeddings size, i.e., 64, when trained on only a small number of labelled examples. However, when trained on the fully labelled datasets, most shallow classifiers achieve their highest accuracy for the largest embeddings size of 256.

In self-training using metric losses, shallow classifiers are used for calculation of test accuracy, labelling the unlabelled examples, and prediction score or probability is used as a confidence measure for selecting pseudo-labelled examples from unlabelled examples. All shallow classifiers achieved a similar accuracy for 64 embeddings size when trained on 4000-labelled examples. These shallow classifiers yield high prediction scores for unlabelled examples. The high scores do not provide a good measure for the selection of pseudo-labels. Therefore, we devised a confidence measure based on the 1-nearest-neighbour using the distance from the unlabelled example to the nearest labelled example.

Table 5.6: Test accuracy and time spent on different shallow classifiers for various sizes of embeddings.

	Embedding	**4000-Labelled**		**All-Labelled**	
Classifier	**size**	**Accuracy**	**Time (ms)**	**Accuracy**	**Time (s)**
KNN	64	75.80 ± 0.36	469 ± 4.91	85.24 ± 0.46	4.40 ± 0.01
	128	74.66 ± 0.81	488 ± 1.67	85.50 ± 0.49	4.71 ± 0.01
	256	74.54 ± 0.40	543 ± 2.76	85.13 ± 0.42	5.93 ± 0.01
LDA	64	76.16 ± 0.37	23 ± 0.01	87.29 ± 0.45	0.30 ± 0.01
	128	75.26 ± 0.65	48 ± 0.01	87.52 ± 0.40	0.63 ± 0.01
	256	74.91 ± 0.45	102 ± 0.01	87.66 ± 0.26	1.39 ± 0.04
RF	64	76.13 ± 0.36	1190 ± 1.84	87.28 ± 0.25	40.60 ± 0.02
	128	75.26 ± 0.58	1970 ± 1.53	87.52 ± 0.42	58.90 ± 0.04
	256	75.02 ± 0.36	2530 ± 7.06	87.61 ± 0.32	86.00 ± 0.09
LR (LIBLINEAR)	64	$\mathbf{76.32 \pm 0.25}$	304 ± 0.13	87.47 ± 0.47	14.40 ± 0.01
	128	75.34 ± 0.54	945 ± 6.46	87.61 ± 0.31	31.30 ± 0.03
	256	75.19 ± 0.31	2460 ± 1.99	87.62 ± 0.07	56.90 ± 0.02
LR (LBFGS)	64	76.31 ± 0.18	203 ± 0.39	87.51 ± 0.49	3.91 ± 0.02
	128	75.25 ± 0.46	342 ± 4.19	87.58 ± 0.28	4.38 ± 0.02
	256	75.01 ± 0.23	528 ± 5.18	$\mathbf{87.65 \pm 0.13}$	5.57 ± 0.08

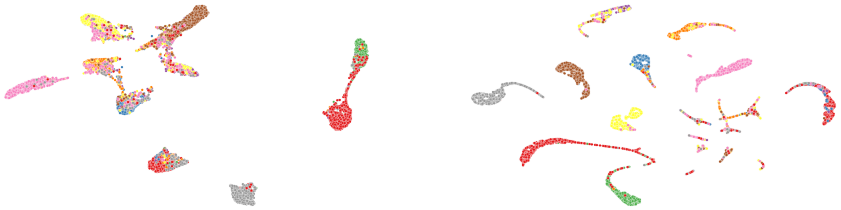
5.2.5 Visualisations of Embeddings

To investigate the effect of self-training on the embeddings, we visualise the embeddings learned using all four loss functions. Figure 5.5 shows the output of UMAP (McInnes et al., 2018) on embeddings of CIFAR-10 test examples obtained after training on 4000 labelled examples and after 25 meta-iterations of self-training using all four losses using pretrained VGG16. Self-training improves class separation, with cross-entropy (CE) loss showing the most significant improvement, consistent with its high final accuracy (see Table 5.2). All the visualisations are generated using the default parameters of UMAP for a fair comparison, the stringy structures in UMAP output are due to the interaction of different parameters of the UMAP algorithm itself.

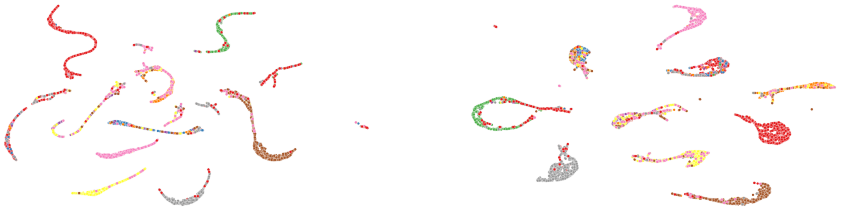
5.3 Pretraining Wide Residual Network on ImageNet

The VGG16 network used in the previous Section 5.2 is trained on ImageNet with size 224 by 224 images using softmax cross-entropy loss having 14 Million parameters (excluding fully-connected layers). Here, we explore pretraining a relatively smaller network, Wide Residual Network (WRN-28-2), with approximately 1.5 Million parameters (see Section 2.1.4). The smaller size of the WRN-28-2 network makes it easy to pretrain on various downsampled smaller sizes of ImageNet. Therefore, various smaller sizes of ImageNet are used for pretraining WRN-28-2 for investigating the effect on accuracy. Moreover, it has been observed that cross-entropy loss usually benefits more from VGG16 pretrained network weights with cross-entropy (see Section 5.2) loss winning for most of the different datasets. It is thus worthwhile to investigate pretraining using different loss functions.

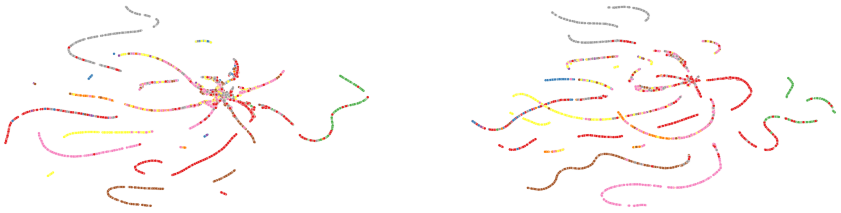
(a) CE: training on 4K-labelled (b) CE: after 25 meta-iterations



(c) Triplet: training on 4K-labelled (d) Triplet: after 25 meta-iterations



(e) Contrastive: training on 4K-labelled (f) Contrastive: after 25 meta-iterations



(g) ArcFace: training on 4K-labelled (h) ArcFace: after 25 meta-iterations

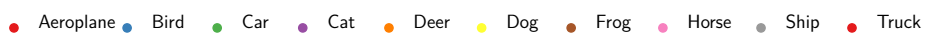


Figure 5.5: UMAP visualisation of CIFAR10 test set embeddings after training on 4000-labelled examples and after 25-meta iterations of self-training using various losses with ImageNet pretrained network weights.

5.3.1 Pretraining using Different Image sizes

The datasets considered in the thesis for evaluation have image sizes 32, 64, and 96. For investigating the effect of ImageNet networks pretrained on smaller image size, the WRN-28-2 network is pretrained on downsampled (Chrabaszcz et al., 2017) ImageNet with sizes 32, 64, and 224. For pretraining, WRN-28-2, the cross-entropy loss is used on these various image sizes.

- pretrained32: The network is trained on downsampled 32x32 ImageNet for 100 epochs and achieves 40.96% top-1 validation accuracy.
- pretrained64: The network is trained on downsampled 64x64 ImageNet for 100 epochs and achieves 41.72% top-1 validation accuracy.
- pretrained224: The network is trained on the widely used 224x224 image size of ImageNet for 100 epochs and achieves 50.09% top-1 validation accuracy.

Validation accuracy is measured on the official validation set of ImageNet[2]. For detection of duplicate images between CIFAR-10 and ImageNet32, we used WRN-28-2 pretrained32. There are approximately 144 matching images present in the ImageNet32 (Chrabaszcz et al., 2017) training set and CIFAR-10 test set (for more, see Appendix B). It is standard practice to fine-tune an ImageNet pretrained network without taking any special consideration of duplicate matching images. So, for CIFAR-10, we calculate test accuracy on all test examples.

Table 5.7 shows the test accuracy for CIFAR-10, SVHN, and PlantVillage32, 64, 96 using randomly initialised, pretrained32, pretrained64, and pretrained224 weights for WRN-28-2 network employing cross-entropy and triplet loss. The reported test accuracies are obtained after training with a small number of labelled examples (1000 for SVHN, 4000 for CIFAR-10, and 380 for

[2]Pretrained weights are available at `https://github.com/attaullah/Pretraining-WideResNet`

PlantVillage), after self-training, and all labelled examples. The test accuracy for contrastive and ArcFace loss is omitted as previous results show that these losses do not perform better in accuracy than cross-entropy and triplet loss for VGG16 random and pretrained.

For SVHN, pretrained32 yields higher test accuracy than randomly initialised, pretrained64, or pretrained224 when few labelled examples are available for training. At the same time, random initialisation of the network achieves the highest accuracy for self-training and all-labelled examples. The impact of pretrained weights on SVHN for all-labelled and self-training is negligible due to the fact that SVHN is rather different from the source domain of pretraining, i.e., ImageNet. For CIFAR-10, pretrained32 outperforms randomly initialised, pretrained64, and pretrained224 for both loss functions on all configurations, with pretrained64 being the second best. For all PlantVillage sizes, pretrained32 achieves the highest accuracy compared to random, pretrained64 and pretrained224. Among pretrained weights, pretrained32 achieves the highest accuracy most of the time for all datasets. Therefore, for WRN-28-2 pretrained weights, pretrained32 weights are used for the remainder of the thesis.

5.3.2 Pretraining using Triplet Loss

As it has been observed that VGG16 pretrained network weights provided more performance improvement on cross-entropy loss compared to metric learning losses, e.g., triplet, contrastive and ArcFace loss, we now compare to WRN-28-2 using triplet loss for pretraining on ImageNet32 (Chrabaszcz et al., 2017). The network is trained for 500 epochs and achieves 13.00% validation accuracy using Linear Discriminant Analysis (LDA). All five datasets' test accuracy using a small number of labelled examples, and all labelled examples employing cross-entropy and triplet loss is evaluated on WRN-28-2. Table 5.8 compares the test accuracy using random weights, pretrained32 weights using cross-entropy loss (CE-pretrained) and pretrained32 weights using triplet

Table 5.7: Test accuracy on WRN-28-2 using random, pretrained32, pretrained64, and pretrained224 weights employing cross-entropy and triplet loss.

		N-Labelled		Self-training		All-Labelled	
	Weights	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**
SVHN	Random	87.04 ± 1.07	77.53 ± 2.29	$\mathbf{94.28 \pm 0.45}$	$\mathbf{86.02 \pm 1.60}$	$\mathbf{96.28 \pm 0.13}$	96.01 ± 0.05
	pretrained32	$\mathbf{88.94 \pm 0.35}$	$\mathbf{84.53 \pm 1.24}$	90.84 ± 0.48	85.03 ± 1.05	95.92 ± 0.40	$\mathbf{96.06 \pm 0.20}$
	pretrained64	79.29 ± 2.55	83.01 ± 2.09	88.37 ± 2.59	85.06 ± 1.18	95.63 ± 0.25	96.14 ± 0.05
	pretrained224	78.60 ± 1.34	81.84 ± 2.40	91.23 ± 0.11	83.62 ± 0.70	95.97 ± 0.15	95.66 ± 0.06
CIFAR-10	Random	75.33 ± 0.82	74.75 ± 1.60	83.05 ± 0.99	82.83 ± 0.78	90.95 ± 0.70	88.80 ± 0.41
	pretrained32	$\mathbf{87.90 \pm 0.94}$	$\mathbf{89.01 \pm 1.29}$	$\mathbf{91.62 \pm 0.58}$	$\mathbf{91.41 \pm 0.23}$	$\mathbf{94.21 \pm 0.55}$	$\mathbf{94.32 \pm 0.05}$
	pretrained64	86.88 ± 0.97	86.64 ± 0.63	90.67 ± 0.16	89.70 ± 0.13	92.98 ± 0.27	93.18 ± 0.31
	pretrained224	80.78 ± 0.55	80.99 ± 0.53	85.12 ± 0.08	86.04 ± 0.02	91.96 ± 0.03	91.88 ± 0.09
PLANT32	Random	56.59 ± 3.25	63.71 ± 0.06	68.24 ± 2.44	67.62 ± 1.29	98.02 ± 1.15	98.53 ± 0.18
	pretrained32	$\mathbf{77.34 \pm 0.97}$	$\mathbf{83.28 \pm 0.58}$	87.48 ± 0.84	$\mathbf{81.31 \pm 0.58}$	$\mathbf{99.03 \pm 0.03}$	$\mathbf{99.30 \pm 0.06}$
	pretrained64	74.70 ± 0.78	77.35 ± 2.67	$\mathbf{88.70 \pm 0.89}$	74.86 ± 3.67	98.25 ± 0.12	99.04 ± 0.12
	pretrained224	65.22 ± 2.94	63.14 ± 7.40	79.32 ± 1.84	66.84 ± 0.33	98.50 ± 0.02	98.44 ± 0.35
PLANT64	Random	68.65 ± 1.89	62.54 ± 3.33	69.36 ± 1.80	71.03 ± 1.72	98.92 ± 0.31	99.57 ± 0.02
	pretrained32	$\mathbf{79.83 \pm 0.55}$	$\mathbf{84.36 \pm 1.41}$	$\mathbf{92.60 \pm 1.03}$	$\mathbf{82.20 \pm 1.23}$	$\mathbf{99.47 \pm 0.19}$	$\mathbf{99.76 \pm 0.02}$
	pretrained64	78.84 ± 2.21	81.82 ± 1.35	88.63 ± 0.47	78.31 ± 2.30	98.28 ± 0.68	99.61 ± 0.08
	pretrained224	72.54 ± 2.08	75.08 ± 2.74	81.31 ± 2.77	74.98 ± 0.54	98.73 ± 0.05	99.46 ± 0.03
PLANT96	Random	71.13 ± 4.31	66.55 ± 1.80	76.09 ± 2.35	74.87 ± 0.60	99.00 ± 0.26	99.43 ± 0.13
	pretrained32	82.25 ± 0.98	$\mathbf{84.72 \pm 0.44}$	$\mathbf{89.65 \pm 0.88}$	$\mathbf{82.52 \pm 1.85}$	$\mathbf{99.28 \pm 0.05}$	$\mathbf{99.70 \pm 0.03}$
	pretrained64	$\mathbf{82.88 \pm 1.60}$	83.29 ± 1.00	85.30 ± 0.86	78.30 ± 0.29	97.90 ± 0.29	99.55 ± 0.10
	pretrained224	76.28 ± 0.49	78.58 ± 3.09	88.19 ± 0.33	79.18 ± 0.22	98.73 ± 0.02	99.25 ± 0.17

Note: For each dataset, bold highlights the highest accuracy from each loss across random, pretrained32, pretrained64, and pretrained224 weights for N-labelled, self-training, and all-labelled.

Table 5.8: Test accuracy of WRN-28-2 using Random weights, CE-weights, and Triplet-weights.

		N-Labelled		All-Labelled	
	Weights	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**
SVHN	Random	87.04 ± 1.07	77.53 ± 2.29	$\mathbf{96.28 \pm 0.13}$	96.01 ± 0.05
	CE-pretrained	$\mathbf{88.94 \pm 0.35}$	$\mathbf{84.53 \pm 1.24}$	95.92 ± 0.40	$\mathbf{96.06 \pm 0.20}$
	Triplet-pretrained	77.65 ± 2.25	76.54 ± 0.85	95.38 ± 0.39	95.55 ± 0.47
CIFAR-10	Random	75.33 ± 0.82	74.75 ± 1.60	90.95 ± 0.70	88.80 ± 0.41
	CE-pretrained	$\mathbf{87.90 \pm 0.94}$	$\mathbf{89.01 \pm 1.29}$	$\mathbf{94.21 \pm 0.55}$	$\mathbf{94.32 \pm 0.05}$
	Triplet-pretrained	81.40 ± 3.72	84.24 ± 0.44	91.71 ± 0.19	91.43 ± 0.13
Plant32	Random	56.59 ± 3.25	63.71 ± 0.06	98.02 ± 1.15	98.53 ± 0.18
	CE-pretrained	$\mathbf{77.34 \pm 0.97}$	$\mathbf{83.28 \pm 0.58}$	$\mathbf{99.03 \pm 0.03}$	$\mathbf{99.30 \pm 0.06}$
	Triplet-pretrained	69.10 ± 0.79	72.08 ± 0.09	97.88 ± 0.82	99.01 ± 0.03
Plant64	Random	68.65 ± 1.89	62.54 ± 3.33	98.92 ± 0.31	99.57 ± 0.02
	CE-pretrained	$\mathbf{79.83 \pm 0.55}$	$\mathbf{84.36 \pm 1.41}$	$\mathbf{99.47 \pm 0.19}$	$\mathbf{99.76 \pm 0.02}$
	Triplet-pretrained	73.30 ± 0.91	76.89 ± 0.82	98.92 ± 0.10	99.25 ± 0.25
Plant96	Random	71.13 ± 4.31	66.55 ± 1.80	99.00 ± 0.26	99.43 ± 0.13
	CE-pretrained	$\mathbf{82.25 \pm 0.98}$	$\mathbf{84.72 \pm 0.44}$	$\mathbf{99.28 \pm 0.05}$	$\mathbf{99.70 \pm 0.03}$
	Triplet-pretrained	74.76 ± 0.84	78.05 ± 0.05	98.30 ± 0.64	99.54 ± 0.05

Note: For each dataset, bold highlights the highest accuracy from each loss across random, CE-pretrained, and Triplet-pretrained weights for N-labelled and all-labelled.

loss (Triplet-pretrained). It is evident that CE-pretrained weights outperform random and triplet-pretrained regardless of the number of labelled examples and loss functions used. Since training on a few labelled and all-labelled examples using triplet-pretrained weights does not yield any improvement over CE-pretrained, self-training results are not computed.

5.3.3 Fixed K Training

In self-training, after each iteration, the model predicts labels for unlabelled examples and merges pseudo-labelled examples with confident predictions into the set of labelled examples for the training of subsequent iterations. In this way, noisy labels can be introduced by adding incorrect labels into labelled examples. To overcome this, instead of using completely random sampling, a fixed k initially labelled examples are added into each mini-batch (Arazo et al., 2020) during training of the model. We set the value of k the same as the number of classes, i.e., 10 for CIFAR-10 and SVHN and 38 for PlantVillage. Table 5.9 shows the test accuracy after 25 iterations of self-training for all datasets using cross-entropy and triplet loss on WRN-28-2 with random and ImageNet32 pretrained weights for randomly sampled mini-batches and k-fixed initially labelled examples in each mini-batch. It is evident that the randomly sampled mini-batches perform better than k-fixed mini-batch construction for all datasets on both losses with random and pretrained weights.

5.3.4 Visualisations of Embeddings

Figure 5.6 shows the output of 2-dimensional UMAP (McInnes et al., 2018) obtained after 64-dimensional embedding of WRN-28-2 using random and pretrained weights. The embeddings are obtained at the start of training, after training with 4000 labelled examples, and after 25 meta-iterations of self-training on CIFAR-10. It can be seen that pretrained weights result in better separation at all three stages: after init, after the first 4k examples, and finally after all 25 meta-iterations.

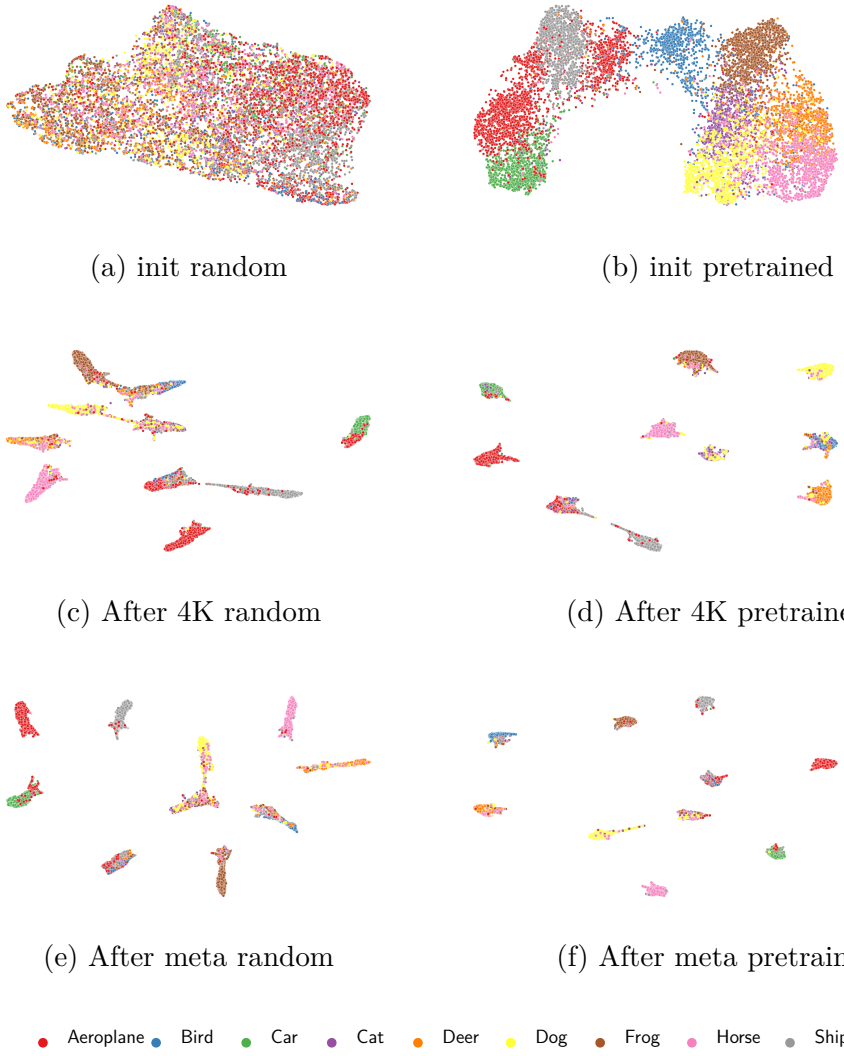
5.4 Comparison of Networks

Table 5.10 summarises the test accuracy for the proposed self-training approach on VGG16 and WRN-28-2 for all datasets with cross-entropy loss. Results are shown with randomly initialised weights and ImageNet pretrained

Table 5.9: Test accuracy after self-training on WRN-28-2 using random and ImageNet pretrained weights for randomly sampled and k-fixed initially labelled examples in each mini-batch.

		Cross-entropy		**Triplet**	
	Weights	**Random**	**k-fixed**	**Random**	**k-fixed**
SVHN	Random	$\mathbf{94.28 \pm 0.45}$	90.02 ± 0.77	86.02 ± 1.60	85.36 ± 0.91
	ImageNet	90.84 ± 0.48	$\mathbf{91.76 \pm 0.44}$	85.03 ± 1.05	88.08 ± 0.02
CIFAR-10	Random	$\mathbf{83.05 \pm 0.99}$	81.72 ± 0.12	82.83 ± 0.78	79.66 ± 0.71
	ImageNet	$\mathbf{91.62 \pm 0.58}$	90.30 ± 0.43	91.41 ± 0.23	89.26 ± 0.63
PLANT32	Random	$\mathbf{68.24 \pm 2.44}$	63.00 ± 2.81	67.62 ± 1.29	62.28 ± 2.66
	ImageNet	$\mathbf{87.48 \pm 0.84}$	82.55 ± 0.16	81.31 ± 0.58	81.63 ± 0.18
PLANT64	Random	69.36 ± 1.80	69.42 ± 0.77	71.03 ± 1.72	$\mathbf{71.09 \pm 2.99}$
	ImageNet	$\mathbf{92.60 \pm 1.03}$	84.18 ± 0.68	82.20 ± 1.23	83.66 ± 0.82
PLANT96	Random	$\mathbf{76.09 \pm 2.35}$	67.19 ± 4.53	74.87 ± 0.60	74.991 ± 0.82
	ImageNet	$\mathbf{89.65 \pm 0.88}$	83.11 ± 1.08	82.52 ± 1.85	81.50 ± 0.81

Note: For each dataset, bold highlights the highest accuracy for both losses with randomly sampled and k-fixed initially labelled examples in each mini-batch using random and ImageNet pretrained weights for self-training.



(a) init random

(b) init pretrained

(c) After 4K random

(d) After 4K pretrained

(e) After meta random

(f) After meta pretrained

Figure 5.6: UMAP visualisation of CIFAR-10 test set embeddings using triplet loss with random and ImageNet pretrained network weights on WRN-28-2.

Table 5.10: Test accuracy after self-training on VGG16 and WRN-28-2 using random and ImageNet pretrained weights.

	SVHN	**CIFAR-10**	**PLANT32**	**PLANT64**	**PLANT96**
Random weights					
VGG16	88.71 ± 0.47	77.18 ± 0.48	57.78 ± 2.74	65.57 ± 3.88	48.37 ± 3.32
WRN-28-2	$\mathbf{94.28 \pm 0.45}$	$\mathbf{83.05 \pm 0.99}$	$\mathbf{68.24 \pm 2.44}$	$\mathbf{69.36 \pm 1.80}$	$\mathbf{76.09 \pm 2.35}$
ImageNet pretrained weights					
VGG16	$\mathbf{92.75 \pm 1.33}$	83.30 ± 0.27	73.72 ± 0.58	79.87 ± 3.94	78.72 ± 0.08
WRN-28-2/224	91.23 ± 0.11	85.12 ± 0.08	79.32 ± 1.84	81.31 ± 2.77	88.19 ± 0.33
WRN-28-2/64	88.37 ± 2.59	90.67 ± 0.16	$\mathbf{88.70 \pm 0.89}$	88.63 ± 0.47	85.30 ± 0.86
WRN-28-2/32	90.84 ± 0.48	$\mathbf{91.62 \pm 0.58}$	87.48 ± 0.84	$\mathbf{92.60 \pm 1.03}$	$\mathbf{89.65 \pm 0.88}$

weights of the network. For VGG16, publically available ImageNet 224 by 224 pretrained weights are used, and for WRN-28-2, the network was trained on ImageNet 224 by 224, 64 by 64, and 32 by 32 (see Section 5.3.1). For randomly initialised weights, WRN-28-2 outperforms VGG16. For ImageNet pretrained weights, WRN-28-2 pretrained on ImageNet 32 by 32 achieves the highest test accuracy for the majority of the datasets.

5.5 Discussion

In this chapter, we have shown that transfer learning can be highly beneficial for image classification. More specifically, when there are a few labelled examples available for training, transfer learning also improves the performance of self-training, even using different types of losses for the training of neural networks. We also pretrained the WRN-28-2 network on various smaller sizes of ImageNet using cross-entropy and triplet loss and found that cross-entropy loss on ImageNet32 provides higher accuracy on all datasets. Moreover, WRN-28-2 outperforms VGG16 on all datasets using random as well as pretrained weights on ImageNet32. In terms of loss functions, overall, cross-entropy outperforms more specialised losses like triplet loss, contrastive loss, or ArcFace loss. Still,

for a small number of labels, triplet loss is very competitive. SVHN seems to get little or no improvement from transfer learning for a few settings of loss functions and networks. One of the possible explanations is that SVHN is not much related to the source domain of ImageNet. For PlantVillage, metric learning losses suffer from performance reduction for self-training with random and pretrained weights on VGG16.

Self-training utilises the unlabelled examples after each iteration by labelling them and selecting them based on the confidence score in subsequent iterations. In the next chapter, we explore the avenues for exploiting the unlabelled examples at the start of the training.

Chapter 6

Self-training using Self-supervised Learning

In self-training, a single supervised classifier is trained on initially labelled examples and then employed to predict labels for unlabelled examples. Afterwards, the model is trained iteratively on initially labelled examples and confident predictions of unlabelled examples. As we can see, self-training uses unlabelled examples only after they have been labelled and selected for training based on confidence. This chapter explores how to exploit unlabelled examples from the beginning of the model training in an unsupervised way.

More specifically, we consider self-supervised learning for unlabelled examples to improve the prediction performance of the model. Self-supervised learning (Jing and Tian, 2020) is unsupervised learning, where the model is trained using a standard supervised loss, but the labels come from a pretext task. Pretext tasks, also called self-supervised tasks, provide labels based on context and other properties of images from the data. No human labelling is needed; therefore, self-supervised learning is considered unsupervised learning. The motivation of training is not to maximise performance on a pretext task but to learn features that will eventually help achieve high predictive performance on the target task.

Self-supervised learning can be used as:

1. Pretraining on unlabelled examples followed by fine-tuning on labelled examples as demonstrated in (Chen et al., 2020a,b)

2. Simultaneously training with supervised training on labelled examples and self-supervised training on unlabelled examples as demonstrated in (Zhai et al., 2019; Tran, 2019).

This chapter introduces self-supervised pretraining followed by fine-tuning and joint self-supervised and supervised training in the context of self-training. In joint self-supervised and supervised training, we also introduce triplet loss instead of the commonly used cross-entropy loss for supervised training. Moreover, the effect of replacing cross-entropy loss with triplet loss in self-supervised training is investigated. We also explore the effect of using randomly initialised weights and ImageNet pretrained weights for the neural network in self-training scenarios.

6.1 Self-supervised Pretext Tasks

There are a variety of pretext tasks proposed for self-supervised training. Here we briefly describe some pretext tasks for computer vision tasks.

- ExemplarCNN (Dosovitskiy et al., 2015): N different classes are generated by applying transformations on random image patches, and the network is trained to predict the correct class of the given patch.

- RotNet (Gidaris et al., 2018): Geometric transformations such as rotations by 0, 90, 180, and 270 degrees are applied to an image, and the neural network is trained to predict the rotations applied to the image.

- Jigsaw Puzzles (Noroozi and Favaro, 2016): For a given image, nine patches are generated, and the network is trained to predict the correct permutation order of the patches for that image.

- Contrastive learning: The network is trained to differentiate between positive and negative samples; the image and its augmented version are considered positives. The rest of the images are considered negative for a given batch of images. An example of such a technique is presented in (Chen et al., 2020a,b).

6.2 Self-training using Self-supervised Learning

For self-supervised learning, we consider a pretext task based on rotations as introduced in (Gidaris et al., 2018) and also horizontal (left-right) and vertical (up-down) flips introduced in (Tran, 2019). These geometric transformations give rise to six auxiliary classes based on rotations by 0, 90, 180, and 270 degrees, horizontal left-right flip, and vertical up-down flip. For a given image, six transformed versions are used for self-supervised training, and the objective is to predict the true transformation applied to the image. Here we discuss three different settings of applying self-supervised learning.

6.2.1 Combined Training

In combined training (CT), we jointly train the model using self-supervised loss on unlabelled examples and supervised loss on labelled examples. CT is an intrinsically semi-supervised approach (for more, see Section 3.3.3) based on self-training. Figure 6.1 shows the semi-supervised loss calculation using unlabelled examples and labelled examples. The supervised and self-supervised branches of the model share the backbone convolutional neural network (CNN), while only the final layer is different. Only the supervised branch of the model is used for the inference purpose, and the self-supervised branch is discarded. As shown in Equation 6.1, the final loss is the weighted sum of the supervised and self-supervised loss.

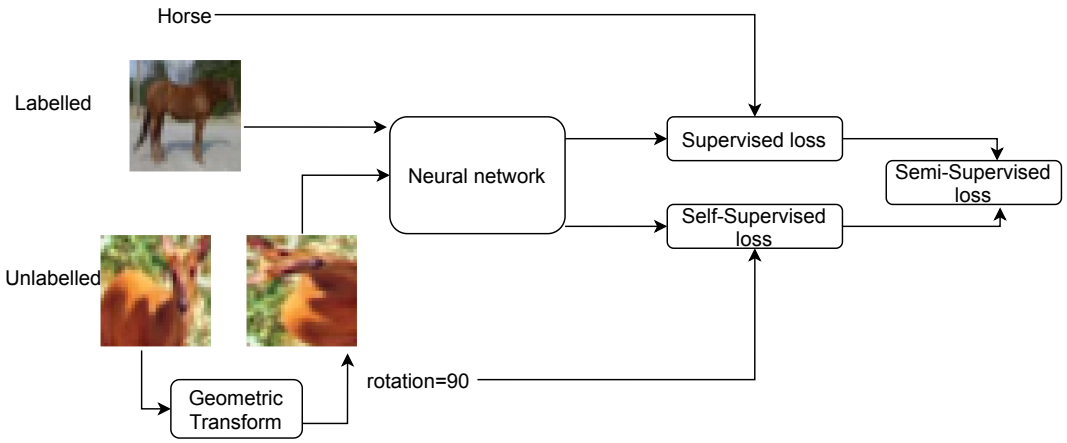


Figure 6.1: Semi-supervised loss calculation using self-supervised and supervised loss.

$$\mathcal{L} = \mathcal{L}_{SUPER}(x_L, y_L) + \lambda_u \mathcal{L}_{SELF}(x_U, y_U) \tag{6.1}$$

Supervised loss is calculated on labelled examples x_L having ground truth labels y_L, while self-supervised loss is calculated on unlabelled examples x_U where labels y_U are provided by a pretext task, i.e., geometric transformations. λ_u is a hyper-parameter for controlling the impact of self-supervised loss. For our experiments, we set $\lambda_u = 1$ to avoid having to tune it separately for every single dataset.

Algorithm 3 shows the training procedure for a mini-batch of supervised and self-supervised losses. For each training step, we sample two mini-batches having the same number of labelled and unlabelled examples. Small vertical and horizontal translations are applied to the mini-batches as an augmentation. All six geometric transformations (four rotations and two flips) are applied to each image in a mini-batch of unlabelled examples resulting in images six times the batch size. Then, both labelled and transformed unlabelled mini-batches are passed to the network f_θ having parameters θ, and the losses are computed using original ground truth labels for the labelled mini-batch, while for unlabelled mini-batch, the labels are provided by the pretext task. The gradients are back-propagated to update the network parameters θ. For

estimation of self-supervised loss, all training examples, i.e., unlabelled and labelled examples, are considered for training.

Algorithm 3 Mini-batch training using supervised and self-supervised loss.

Input: Labelled examples (x_L, y_L), unlabelled examples x_U, Neural network f_θ with parameters θ and $\lambda_u = 1$.
for each mini-batch **do**
  $b_L = \text{sample}(x_L, y_L)$
  $b_U = \text{sample}(x_U \cup x_L)$
  $b_U = \text{Geometric-transform}(b_U)$
  $z_L = f_\theta(b_L)$
  $z_U = f_\theta(b_U)$
  $\mathcal{L} = \mathcal{L}_{SUPER}(z_L) + \lambda_u \mathcal{L}_{SELF}(z_U)$
  $\theta = \theta - \nabla_\theta \mathcal{L}$
end for

Figure 6.2 shows the overall training procedure for self-training using joint training of self-supervised and supervised training based on three steps.

- Step-1:The network model is jointly trained using self-supervised and supervised loss. Self-supervised loss is evaluated on unlabelled examples with geometric transformation prediction, while supervised loss is calculated on initially available labelled examples.
- Step-2: The model is employed to predict labels for unlabelled examples, and top $p\%$ confident predictions are merged with labelled examples.
- Step-3: The model is jointly trained using unlabelled data for self-supervised loss and merged data for supervised loss. Step 2 and step 3 are repeated for several iterations.

In step three, joint training of supervised and self-supervised training, all unlabelled examples are used for self-supervised in each meta-iteration, which makes CT computationally expensive. For supervised training, we also investigate the effect of applying the triplet loss instead of cross-entropy loss.

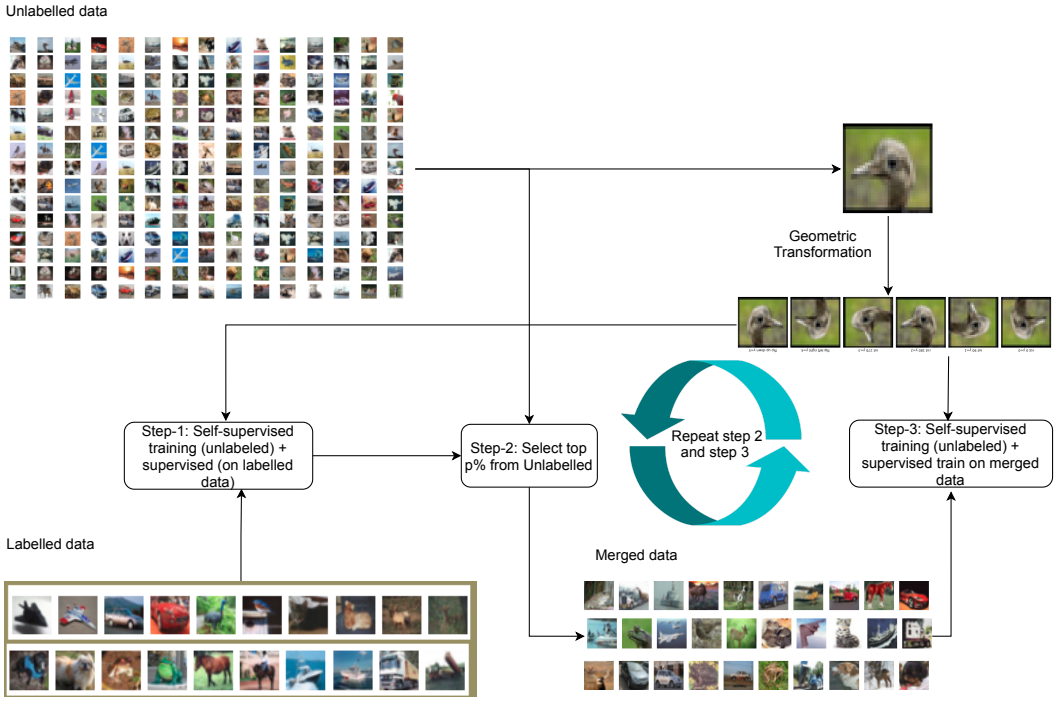


Figure 6.2: An overview of Combined Training.

6.2.2 Self-supervised Pretraining

In unsupervised pretraining, the model is trained on an auxiliary task without human-provided labels before fine-tuning the target task. The motivation for unsupervised pretraining is that the model will learn useful features about the data distribution. This may be particularly suitable, especially in semi-supervised learning, where we have a few labelled and a lot of unlabelled examples available (for more, see Section 3.3.2). Self-supervised pretraining aims to learn the features of the data without using human-annotated labels. Typically, self-supervised pretraining is applied on unlabelled examples followed by fine-tuning on labelled examples (Gidaris et al., 2018). Figure 6.3 shows a schematic overview of the proposed self-training technique using self-supervised pretraining (SS-Pretrain) based on four steps.

- Step-1: Self-supervised pretraining is performed using six geometric transformations based on four rotations and two flips as a pretext task.
- Step-2: Fine-tuning using supervised loss is performed on labelled examples.

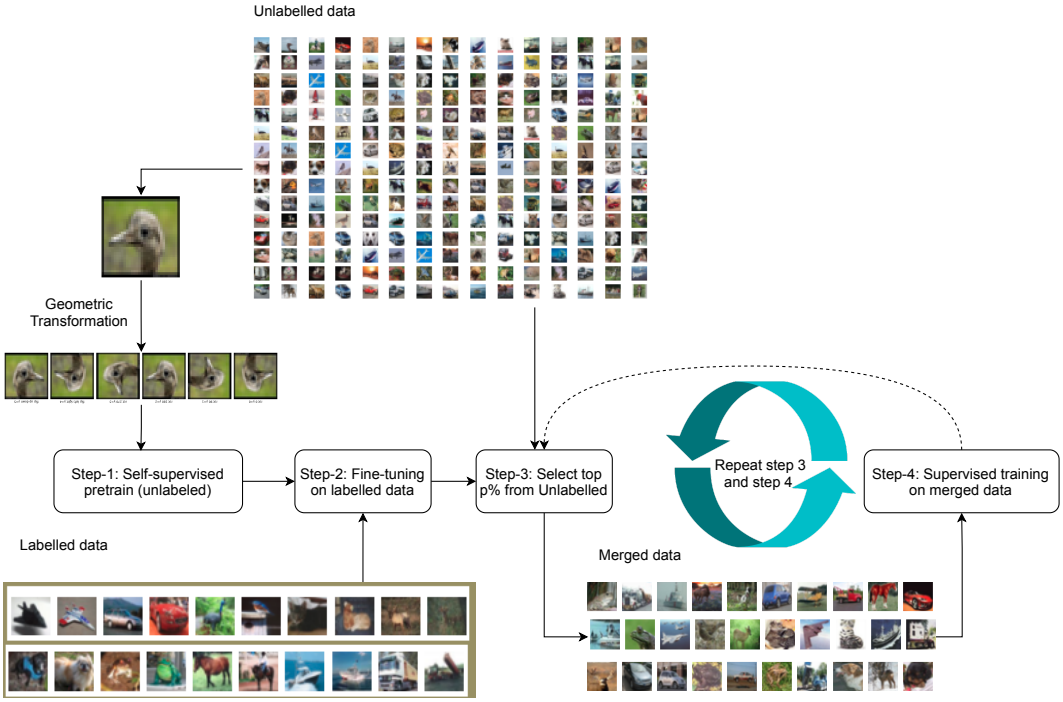


Figure 6.3: Schematic diagram of SS-Pretrain

- Step-3: The model is employed to predict the labels for unlabelled examples, and top $p\%$ most confidently labelled unlabeled examples are merged with labelled examples.

- Step-4: Then, the model is retrained on the merged data using supervised loss only. Step 3 and step 4 are repeated for several iterations.

For self-supervised pretraining, the prediction of six transformations is used as a pretext task.

6.2.3 Self-training using Single-Step Combined Training

In combined training (see Section 6.2.1), self-supervised training is applied to unlabelled examples in each iteration of self-training along with supervised training. Since self-supervised training is computationally expensive and unlabelled examples do not change during training, we develop an alternative to full combined training. More specifically, combined training of self-supervised and supervised training is applied for one iteration of self-training, and subsequent iterations only apply supervised training on initially labelled and pseudo-

labelled examples. Figure 6.4 shows the overall training procedure for self-training using a single joint self-supervised and supervised training iteration followed by supervised training in subsequent iterations based on three steps.

- Step-1: The network model is jointly trained using self-supervised and supervised loss. Self-supervised loss is evaluated on unlabelled examples with geometric transformation prediction, while supervised loss is calculated on initially available labelled examples.
- Step-2: The model is employed to predict labels for unlabelled examples, and top $p\%$ most confidently labelled unlabelled examples are merged with labelled examples.
- Step-3: The model is trained on merged data using supervised loss only. Step 2 and step 3 are repeated for several iterations.

For brevity, we call this technique STSSC. For self-supervised training, we apply cross-entropy loss. STSSC represents self-training using a single iteration of combined training using cross-entropy loss for self-supervision. For supervised training, we also investigate the effect of applying the triplet loss along with cross-entropy loss.

6.3 Experiments

For evaluating the effect of self-supervised learning, we follow the same protocol as in the previous chapters. A small subset of labelled examples is chosen randomly for all five datasets, i.e., SVHN, CIFAR-10, PlantVillage32, PlantVillage64, and PlantVillage96 according to standard semi-supervised learning practice, with a balanced number of examples from each class. All remaining examples are used as unlabelled training examples. Initially labelled examples for SVHN are 1000, 4000 for CIFAR-10 and 380 for PlantVillage32, 64, and 96. We always report three results for test accuracy: (a) After training on a small labelled examples, (b) after 25 self-training meta-iterations, and (c)

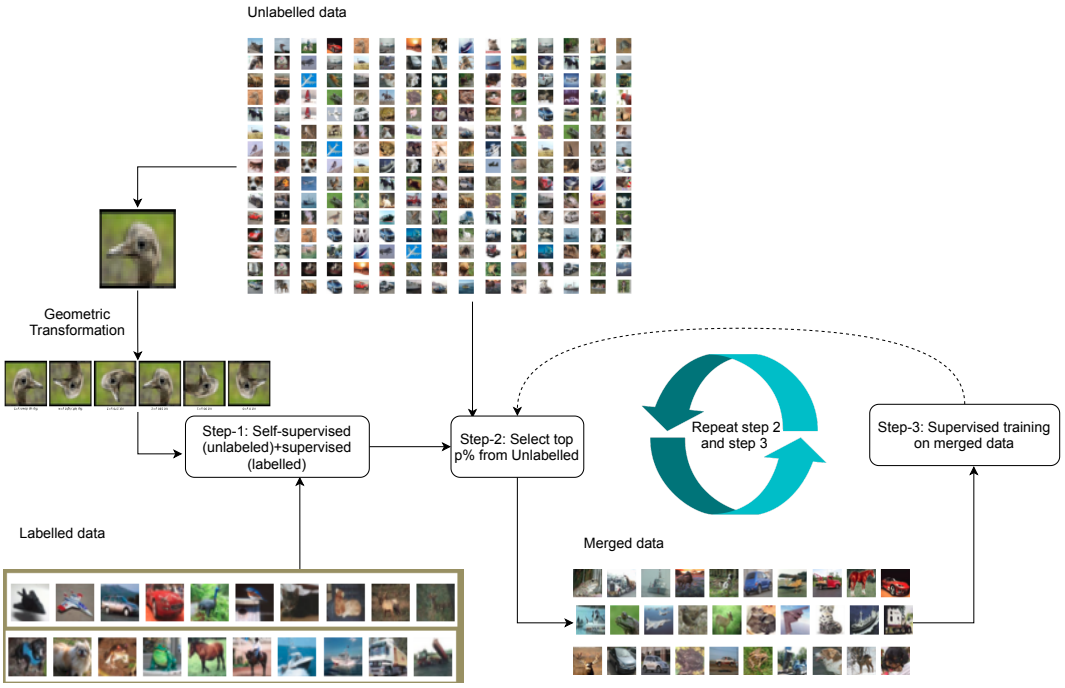


Figure 6.4: A schematic overview of STSSC.

after training on all training examples. All the results are averaged over three runs with a random selection of initially labelled examples.

For self-supervised training, the pretext task based on six geometric transformations is used. WRN-28-2 is used as a shared CNN backbone for self-supervised and supervised training, having: a) randomly initialised weights and b) ImageNet pretrained weights. A fully connected layer is added at the end of the WRN-28-2 model producing 64-dimensional embeddings. A mini-batch size of 100 is used for all experiments for SVHN, CIFAR-10, while Plant32, 64, and 96 use a batch size of 64. Adam is used as an optimiser for updating the network parameters with a learning rate of 10^{-3} for randomly initialised weights and 10^{-4} for ImageNet pretrained weights. After the first iteration of self-training, the learning rate is further reduced by a factor of 0.1. For the triplet loss, $l2$-normalised embeddings are used with the margin m set to 1 to estimate the loss and a 1-nearest-neighbour classifier is employed to evaluate test accuracy with $k = 1$. In self-training, the distance to the nearest labelled example is used as the confidence measure for selecting the pseudo-labelled examples for triplet loss and the softmax probability score for

cross-entropy loss. Self-training is applied for 25 meta-iterations[1].

6.3.1 Combined Training

Table 6.1 shows test accuracy for CIFAR-10, SVHN, and PlantVillage32, 64, 96 on WRN-28-2 using random and pretrained weights based on combined training (see Section 6.2.1). The cross-entropy loss is used for self-supervised training, while supervised training uses cross-entropy loss and triplet loss. The values in bold highlight the best test accuracy after training on few labelled examples, after applying self-training, and after training on all training examples using random and ImageNet pretrained weights, employing cross-entropy and triplet loss for supervised training for each dataset. Self-training consistently achieves significant performance improvement compared to N-labelled training for all datasets when the cross-entropy loss is applied for supervised training. Conversely, when triplet loss is applied for supervised training, self-training always exhibits performance degradation compared to N-labelled training for all datasets. Most of the time, the network initialised with ImageNet pretrained weights performs better than the randomly initialised weights. In terms of comparison of loss functions, cross-entropy outperforms triplet loss for N-labelled, self-training, and all-labelled training.

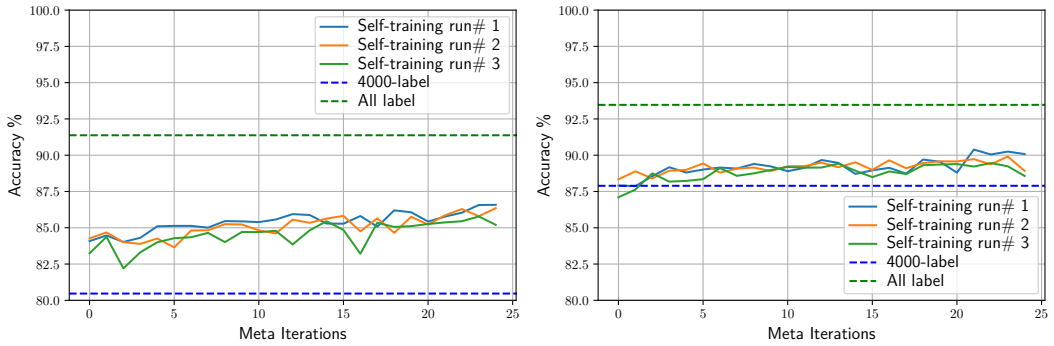
Figure 6.5 shows details about the test accuracy after each iteration of self-training using combined training (CT) across three different runs using a) random weights and b) ImageNet pretrained weights for CIFAR-10 using 4000 initially labelled examples. For supervised training, cross-entropy loss is used. The accuracy curves show improvements for both scenarios, with the pretrained version starting from a higher initial accuracy level and retaining this advantage over the 25 meta-iterations of self-training

[1]Source code is available at `https://github.com/attaullah/Self-training/blob/master/Self_supervised.md`

Table 6.1: Test Accuracy on WRN-28-2 using random and ImageNet pre-trained weights for Combined Training (CT).

		N-Labelled		Self-training		All-Labelled	
	Weights	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**
SVHN	Random	86.37 ± 3.57	89.38 ± 1.69	88.10 ± 3.02	79.47 ± 0.82	$\mathbf{95.74 \pm 0.02}$	95.07 ± 0.15
	ImageNet	$\mathbf{89.62 \pm 1.22}$	88.98 ± 2.14	$\mathbf{90.83 \pm 1.67}$	81.96 ± 3.24	95.16 ± 0.16	94.65 ± 0.35
CIFAR-10	Random	80.67 ± 2.95	81.25 ± 1.77	86.04 ± 0.60	81.51 ± 0.73	91.38 ± 0.39	86.97 ± 3.47
	ImageNet	87.00 ± 1.97	$\mathbf{90.04 \pm 1.22}$	$\mathbf{89.19 \pm 0.65}$	89.11 ± 2.33	93.47 ± 0.08	$\mathbf{93.92 \pm 0.09}$
PLANT32	Random	67.46 ± 3.39	55.27 ± 3.77	72.91 ± 3.68	50.99 ± 1.21	97.97 ± 0.64	91.23 ± 0.86
	ImageNet	$\mathbf{77.96 \pm 0.58}$	73.38 ± 1.28	$\mathbf{78.18 \pm 0.09}$	66.67 ± 0.60	$\mathbf{98.38 \pm 0.08}$	96.58 ± 0.48
PLANT64	Random	72.52 ± 3.86	63.86 ± 0.94	83.52 ± 1.84	59.84 ± 0.38	98.52 ± 0.34	91.99 ± 0.30
	ImageNet	$\mathbf{78.39 \pm 0.87}$	76.65 ± 1.06	$\mathbf{84.98 \pm 0.68}$	69.26 ± 6.11	$\mathbf{99.08 \pm 0.10}$	94.67 ± 0.52
PLANT96	Random	75.12 ± 3.91	65.94 ± 1.43	79.94 ± 1.75	44.55 ± 1.70	98.08 ± 0.65	89.47 ± 0.99
	ImageNet	$\mathbf{79.07 \pm 0.68}$	79.04 ± 0.67	$\mathbf{84.48 \pm 1.30}$	76.26 ± 1.40	$\mathbf{99.15 \pm 0.12}$	95.60 ± 1.27

Note: For each dataset, bold highlights the highest accuracy for both losses using random and ImageNet pretrained weights for N-labelled, self-training, and all-labelled.



(a) Randomly initialised weights (b) Pretrained ImageNet weights

Figure 6.5: Comparison of self-training on WRN-28-2 using CT for CIFAR-10 using cross-entropy loss.

6.3.2 Self-supervised Pretraining

Table 6.2 shows test accuracy for CIFAR-10, SVHN, and PlantVillage32, 64, 96 on WRN-28-2 using random and pretrained weights after applying SS-Pretrain (see Section 6.2.2). For self-supervised pretraining, the cross-entropy loss is applied for 120 epochs on all training examples, followed by fine-tuning on labelled examples for 200 epochs. After that, self-training is applied for 25 meta-iterations using supervised loss. For supervised loss, cross-entropy and triplet loss are used. The values in bold highlight the best test accuracy after training on few labelled examples, after self-training, and after training on all training examples using random and ImageNet pretrained weights employing cross-entropy and triplet loss for each dataset. Triplet loss achieves better accuracy than cross-entropy loss when few labelled examples are available for training, while cross-entropy achieves higher test accuracy than triplet loss for self-training and when a large number of labelled examples are available for training. For self-training, cross-entropy achieves performance improvement most of the time, while triplet loss shows performance degradation most of the time. The network initialised with ImageNet pretrained weights always performs better than the randomly initialised weights for all settings.

Figure 6.6 shows details about the test accuracy after each iteration of self-training using SS-Pretrain across three different runs using a) random weights and b) pretrained ImageNet weights with CIFAR-10 using 4000 initially labelled examples. Cross-entropy loss is employed for self-supervised pretraining and supervised training. Again, the accuracy curves show improvements for both scenarios, with the pretrained version starting from a higher initial accuracy level and retaining this advantage over the 25 meta-iterations of self-training

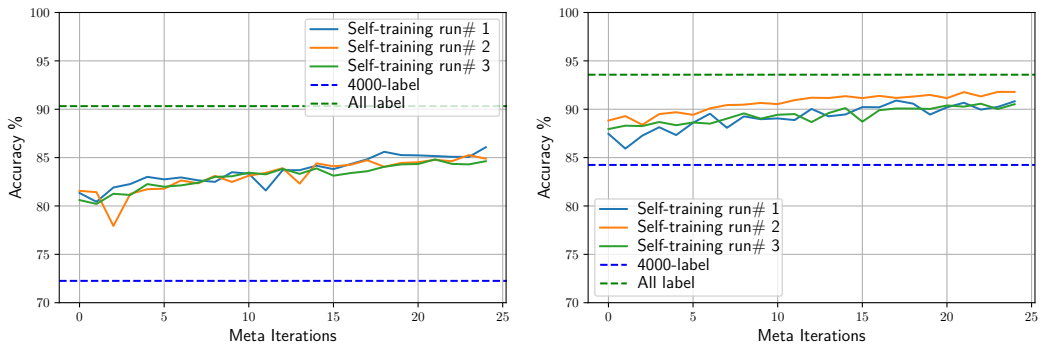
6.3.3 Self-training using Single Step Combined Training

Table 6.3 shows test accuracy of self-training for CIFAR-10, SVHN, and PlantVillage32, 64, 96 on WRN-28-2 using random and ImageNet pretrained weights

Table 6.2: Test accuracy on WRN-28-2 using random and ImageNet pretrained weights for SS-Pretrain.

		N-Labelled		Self-training		All-Labelled	
	Weights	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**	**Cross-entropy**	**Triplet**
SVHN	Random	89.51 ± 0.74	86.48 ± 2.00	$\mathbf{89.72 \pm 0.57}$	83.73 ± 2.46	95.63 ± 0.46	95.05 ± 0.06
	ImageNet	89.90 ± 0.40	$\mathbf{90.08 \pm 0.60}$	89.67 ± 0.10	86.05 ± 0.44	$\mathbf{95.90 \pm 0.05}$	95.30 ± 0.16
CIFAR-10	Random	72.02 ± 0.50	75.76 ± 2.11	85.50 ± 0.41	75.51 ± 1.88	90.32 ± 0.94	89.11 ± 0.68
	ImageNet	84.83 ± 1.99	$\mathbf{86.98 \pm 0.84}$	$\mathbf{91.06 \pm 0.47}$	90.18 ± 0.87	$\mathbf{93.48 \pm 0.09}$	92.64 ± 0.70
PLANT32	Random	69.80 ± 1.07	72.03 ± 1.51	68.72 ± 0.30	65.04 ± 1.19	97.14 ± 0.68	96.64 ± 0.07
	ImageNet	72.38 ± 1.22	$\mathbf{72.46 \pm 1.07}$	$\mathbf{75.21 \pm 1.34}$	69.96 ± 0.05	$\mathbf{99.24 \pm 0.12}$	97.77 ± 0.77
PLANT64	Random	73.35 ± 1.76	71.10 ± 1.16	73.45 ± 3.32	67.06 ± 1.42	97.94 ± 0.48	87.76 ± 0.96
	ImageNet	75.64 ± 0.54	$\mathbf{77.51 \pm 1.05}$	$\mathbf{75.50 \pm 0.93}$	68.37 ± 1.43	$\mathbf{99.06 \pm 0.27}$	89.48 ± 0.56
PLANT96	Random	73.29 ± 0.72	70.61 ± 0.83	75.19 ± 0.88	67.18 ± 1.81	98.43 ± 0.05	90.54 ± 1.18
	ImageNet	78.37 ± 1.19	$\mathbf{78.74 \pm 0.22}$	$\mathbf{79.11 \pm 1.24}$	74.97 ± 0.57	$\mathbf{99.47 \pm 0.12}$	93.32 ± 1.35

Note: For each dataset, bold highlights the highest accuracy for both losses using random and ImageNet pretrained weights for N-labelled, self-training, and all-labelled.



(a) Random weights (b) Pretrained ImageNet weights

Figure 6.6: Comparison of self-training on WRN-28-2 using SS-Pretrain for CIFAR-10 using cross-entropy loss.

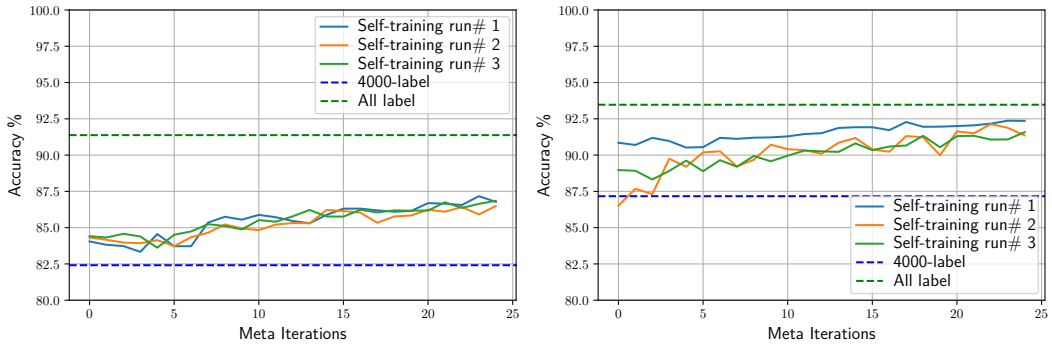
Table 6.3: Self-training test accuracy on WRN-28-2 using random and ImageNet pretrained weights for STSSC.

	Weights	**Cross-entropy**	**Triplet**
SVHN	Random	**95.27 ± 0.20**	83.62 ± 0.03
	ImageNet	95.12 ± 0.32	83.47 ± 0.98
CIFAR-10	Random	87.24 ± 0.67	79.36 ± 1.02
	ImageNet	**91.64 ± 0.38**	90.75 ± 1.48
PLANT32	Random	77.39 ± 1.32	54.44 ± 3.72
	ImageNet	**86.72 ± 1.10**	65.14 ± 1.85
PLANT64	Random	86.13 ± 0.46	56.23 ± 2.01
	ImageNet	**89.48 ± 0.37**	74.29 ± 1.27
PLANT96	Random	84.71 ± 1.89	58.14 ± 0.82
	ImageNet	**89.95 ± 0.37**	74.41 ± 3.75

Note: For each dataset, bold highlights the highest accuracy for both losses using random and ImageNet pretrained weights.

for STSSC (see Section 6.2.3). The reported results are obtained after 25 iterations of self-training using cross-entropy and triplet loss for supervised training. The values in bold highlight the best test accuracy for a dataset using randomly initialised and Imagenet pretrained weights for cross-entropy and triplet loss. Self-training using cross-entropy loss outperforms the triplet loss for all five datasets for both randomly initialised and ImageNet pretrained weights. In the majority of the cases, cross-entropy loss with ImageNet pretrained weights achieves the high test accuracy for STSSC.

Figure 6.7 shows details about the test accuracy after each iteration of STSSC across three different runs using a) randomly initialised weights and b) ImageNet pretrained weights on CIFAR-10 using 4000-labelled examples ini-



(a) Random weights (b) Pretrained ImageNet weights

Figure 6.7: Comparison of self-training using STSSC on WRN-28-2 for CIFAR-10 using cross-entropy loss.

tially. Cross-entropy loss is used for supervised training. Again, the accuracy curves show definite improvements for both scenarios. The pretrained version starts from a higher initial accuracy level and retains this advantage over the 25 meta-iterations of self-training.

6.3.4 Comparison of Self-training Approaches with Self-supervision

Table 6.4 summarises the test accuracy after applying the proposed self-training approaches, i.e., combined training (CT), self-supervised pretraining (SS-Pretrain), and single-step self-supervised combined (STSSC) training with WRN-28-2 using randomly initialised and ImageNet pretrained weights for all five datasets, i.e., SVHN, CIFAR-10, PLANT32, PLANT64, and PLANT96. Cross-entropy loss is used for both supervised and self-supervised loss estimation. The results show that STSSC emerges as the winner using random as well pretrained ImageNet weights.

STSSC applies combined self-supervised training on unlabelled examples and supervised training on labelled examples in the first iteration, while the rest of the iterations apply only supervised training on initially labelled and

Table 6.4: Test accuracy after self-training on WRN-28-2 using various self-supervised settings.

	SVHN	**CIFAR-10**	**PLANT32**	**PLANT64**	**PLANT96**
Random weights					
CT	88.10 ± 3.02	86.04 ± 0.60	72.91 ± 3.68	83.52 ± 1.84	79.24 ± 1.45
SS-Pretrain	89.72 ± 0.57	85.50 ± 0.41	68.72 ± 0.30	73.45 ± 3.32	75.19 ± 0.88
STSSC	$\mathbf{95.27 \pm 0.20}$	$\mathbf{87.08 \pm 0.66}$	$\mathbf{77.39 \pm 1.32}$	$\mathbf{86.13 \pm 0.46}$	$\mathbf{84.71 \pm 1.89}$
ImageNet weights					
CT	90.83 ± 1.67	89.19 ± 0.65	78.18 ± 0.09	84.60 ± 0.77	84.48 ± 1.30
SS-Pretrain	89.67 ± 0.10	91.06 ± 0.47	75.21 ± 1.34	75.50 ± 0.93	79.11 ± 1.24
STSSC	$\mathbf{95.12 \pm 0.32}$	$\mathbf{91.65 \pm 0.43}$	$\mathbf{86.72 \pm 1.10}$	$\mathbf{89.48 \pm 0.37}$	$\mathbf{89.95 \pm 0.37}$

pseudo-labelled examples. A possible explanation for the result is that unlabelled training examples do not change during the training process, and because self-supervised loss approaches zero in the first iteration, STSSC performs better than CT and SS-Pretrain.

To compare the time spent by the proposed self-supervised based self-training approaches, i.e., combined training (CT), self-supervised pretraining (SS-Pretrain), and single-step self-supervised combined (STSSC), WRN-28-2 network model is trained on CIFAR-10 dataset using cross-entropy loss. Figure 6.8 shows time (seconds) spent by each epoch over 25 meta-iterations for all three approaches. For each epoch, CT performs self-supervised training on all training examples; therefore the time required is constant and higher than SS-Pretrain and STSSC. After the pretraining step of SS-Pretrain and the first combined iteration of STSSC, both approaches require a similar time for an epoch for the remaining meta-iterations.

6.3.5 Visualisations of Embeddings

We visualise the quality of embeddings learned after training WRN-28-2 after applying STSSC. For self-supervised and supervised training, cross-entropy

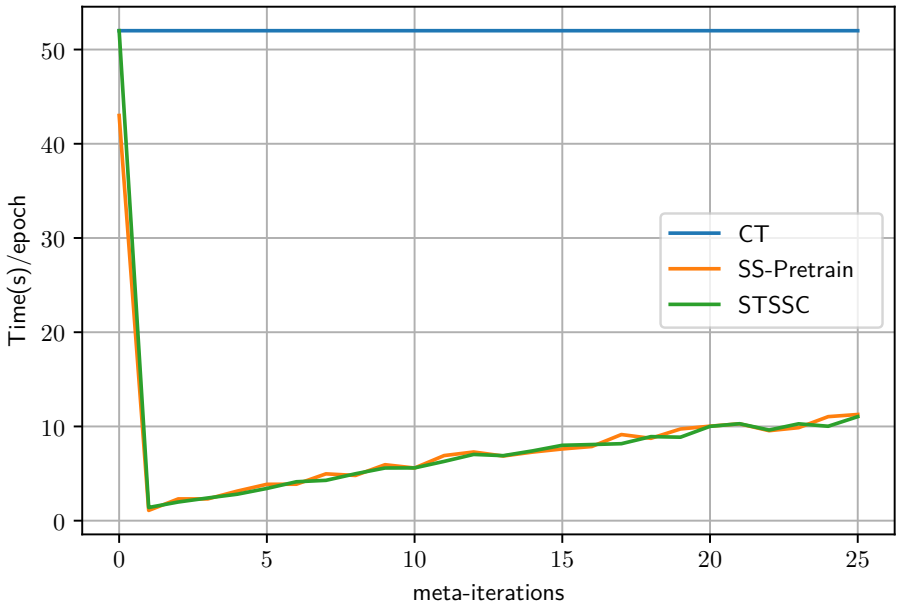
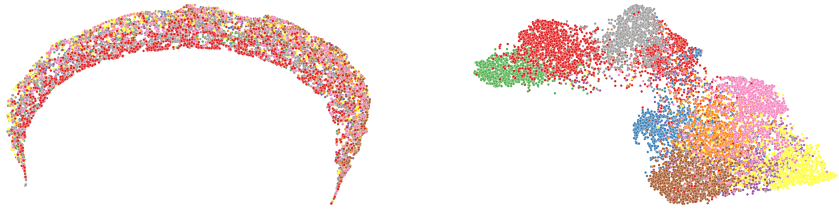


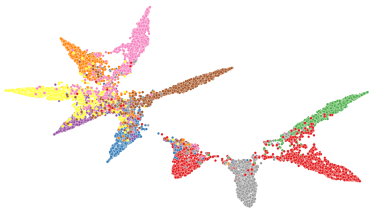
Figure 6.8: Time spent on each epoch over meta-iterations of self-training approaches using self-supervised learning.

loss is employed. Figure 6.9 shows the 2-dimensional output of UMAP (McInnes et al., 2018) obtained from embeddings of CIFAR-10 test examples: (i) at the start of training, (ii) after applying the combined training of self-supervision on all training examples and supervised training on 4000-labelled examples, and (iii) after 25 meta-iterations of self-training using supervised loss on initially labelled and pseudo-labelled examples. For all these three settings, visualisations are shown using WRN-28-2 with randomly initialised weights on the left (a),(c),(e), and for ImageNet pretrained weights of the network on the right (b),(d),(f). The colour indicates the actual class of examples. Self-training improves the separation of embeddings compared to the single iteration of CT. Network with ImageNet pretrained weights separates the embeddings better than with randomly initialised weights, which is also reflected in higher test accuracy (see Table 6.4).

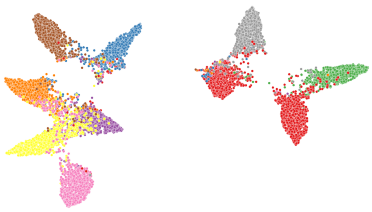


(a) Randomly initialised weights

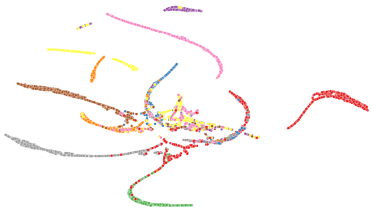
(b) ImageNet pretrained weights



(c) After one iteration of CT



(d) After one iteration of CT



(e) After Self-training

(f) After Self-training

Figure 6.9: UMAP visualisation of CIFAR10 test set embeddings obtained from WRN-28-2 using random weights [left: (a),(c),(e)] and ImageNet pretrained weights [right: (b),(d),(f)] by applying STSSC training.

6.4 Discussion

This chapter introduced methods for exploiting unlabelled examples at the beginning of self-training and in each self-training iteration by applying self-supervised learning. For self-training, self-supervised learning can be applied as pretraining (SS-Pretrain), combined training (CT), and single-step combined training (STSSC). CT is computationally expensive to run compared to SS-Pretrain and STSSC. Among these three proposed approaches, STSSC with cross-entropy loss for supervised training achieves the highest test accuracy for all five datasets using the network with randomly initialised and ImageNet pretrained weights. It is evident that self-supervision is only beneficial when applied at the first iteration of self-training. For self-supervised training, a pretext task based on the prediction of six transformations is used with cross-entropy loss. Triplet loss for self-supervision does not provide any performance benefits compared to cross-entropy loss. In terms of loss functions for supervised training, overall, cross-entropy loss outperforms triplet loss. Networks with ImageNet pretrained weights consistently achieve higher accuracy than networks with randomly initialised weights, regardless of the loss function used for self-supervised and supervised training.

Chapter 7

Conclusions

This thesis began by presenting an argument for exploring metric learning in self-training, in particular, the investigation of three contrasting lines of research with similar goals:

- Applying metric learning for self-training to learn a similarity function motivated by the hypothesis that this will yield better generalisation performance than self-training using cross-entropy.
- Exploring the effect of transferring pretrained parameters for the initialisation of deep neural networks in self-training, based on the observation that transfer of knowledge is particularly useful when small amounts of labelled data are available for training.
- Adapting self-supervised learning for self-training to exploit unlabelled examples in a manner that goes beyond what self-training does by labelling unlabeled examples, thus improving the predictive performance of the model.

Regarding the first point, Chapter 4 has presented a self-training based approach by applying metric learning losses, i.e., contrastive, triplet, and arcFace loss. The proposed approach is very general, suggesting that a spectrum of loss functions can work well in this framework. We also used the most widely used classification loss, i.e., cross-entropy, to compare with metric learning losses.

In self-training iterations, we proposed two confidence measures for selecting pseudo-labels from unlabelled examples: (1) 1-nearest-neighbour distance to the nearest labelled example, (2) LLGC-based prediction score.

The second line of enquiry has been addressed by the method presented in Chapter 5. For exploring the effect of transfer learning on self-training, a publically available VGG16 network pretrained on ImageNet 224 by 224 is compared with a much smaller WRN-28-2 network. WRN-28-2 is pretrained on ImageNet 224 by 224, 64 by 64, and 32 by 32 using cross-entropy and triplet loss.

Chapter 6 addresses the third line of research. Three self-training approaches are introduced in Chapter 6 for applying self-supervised learning. (1) Combined training (CT): a combination of self-supervised training on unlabelled examples and supervised training on labelled and pseudo-labelled examples are applied. (2) Self-supervised pretraining (SS-Pretrain): self-supervised pretraining is applied on all training examples, followed by fine-tuning on labelled examples and then usual self-training. (3) Self-training using Single-Step Combined training (STSSC): CT is applied in the first iteration of self-training followed by usual self-training. CT is computationally expensive because self-supervision is applied to all training examples in each iteration of self-training.

7.1 Summary of Results

Table 7.1 compares the test accuracy of all datasets using the self-training approach introduced in Chapter 4 and self-supervised based self-training approaches introduced in Chapter 6, i.e., Combined Training (CT), Self-supervised pretraining (SS-Pretrain), and Single-Step Combined Training (STSSC) on WRN-28-2 using randomly initialised weights and ImageNet pretrained weights of WRN-28-2 from Chapter 5, against Pseudo-Label Lee (2013); Oliver et al. (2018) (see the discussion of this method in Section 3.3.1). For all config-

Table 7.1: Self-training test accuracy for all datasets on WRN-28-2 using cross-entropy (CE) and triplet loss.

	SVHN	**CIFAR-10**	**PLANT32**	**PLANT64**	**PLANT96**
Random weights					
Pseudo-Label	93.14 ± 0.49	84.62 ± 0.36	76.56 ± 1.48	81.25 ± 1.05	$\mathbf{86.56 \pm 0.06}$
Self-training CE	94.28 ± 0.45	83.05 ± 0.99	68.24 ± 2.44	69.36 ± 1.80	76.09 ± 2.35
Self-training Triplet	86.02 ± 1.60	82.83 ± 0.78	67.62 ± 1.29	71.03 ± 1.72	74.87 ± 0.60
CT CE	88.10 ± 3.02	86.04 ± 0.60	72.91 ± 3.68	83.52 ± 1.84	79.24 ± 1.45
CT Triplet	79.47 ± 0.82	81.51 ± 0.73	50.99 ± 1.21	59.84 ± 0.38	44.55 ± 1.70
SS-Pretrain CE	89.72 ± 0.57	85.50 ± 0.41	68.72 ± 0.30	73.45 ± 3.32	75.19 ± 0.88
SS-Pretrain Triplet	83.73 ± 2.46	75.51 ± 1.88	65.04 ± 1.19	67.06 ± 1.42	67.18 ± 1.81
STSSC CE	$\mathbf{95.27 \pm 0.20}$	$\mathbf{87.08 \pm 0.66}$	$\mathbf{77.39 \pm 1.32}$	$\mathbf{86.13 \pm 0.46}$	84.71 ± 1.89
STSSC Triplet	83.62 ± 0.03	79.36 ± 1.02	54.44 ± 3.72	56.23 ± 2.01	58.14 ± 0.82
ImageNet pretrained weights					
Pseudo-Label	92.94 ± 0.04	90.85 ± 0.14	$\mathbf{91.15 \pm 1.24}$	$\mathbf{94.84 \pm 0.04}$	$\mathbf{92.38 \pm 1.38}$
Self-training CE	90.84 ± 0.48	91.62 ± 0.58	87.48 ± 0.84	92.60 ± 1.03	89.65 ± 0.88
Self-training Triplet	85.03 ± 1.05	91.41 ± 0.23	81.31 ± 0.58	82.20 ± 1.23	82.52 ± 1.85
CT CE	90.83 ± 1.67	89.19 ± 0.65	78.18 ± 0.09	84.60 ± 0.77	84.48 ± 1.30
CT Triplet	81.96 ± 3.24	89.11 ± 2.33	66.67 ± 0.60	69.26 ± 6.11	76.26 ± 1.40
SS-Pretrain CE	89.67 ± 0.10	91.06 ± 0.47	75.21 ± 1.34	75.50 ± 0.93	79.11 ± 1.24
SS-Pretrain Triplet	86.05 ± 0.44	90.18 ± 0.87	69.96 ± 0.05	68.37 ± 1.43	74.97 ± 0.57
STSSC CE	$\mathbf{95.12 \pm 0.32}$	$\mathbf{91.65 \pm 0.43}$	86.72 ± 1.10	89.48 ± 0.37	89.95 ± 0.37
STSSC Triplet	83.47 ± 0.98	90.75 ± 1.48	65.14 ± 1.85	74.29 ± 1.27	74.41 ± 3.75

urations, results are compared for cross-entropy (CE) and triplet loss. Our proposed approach STSSC with cross-entropy loss achieves higher accuracy than Pseudo-Label for four out of five datasets for randomly initialised network weights. For pretrained network weights, STSSC achieves the highest accuracy on SVHN and CIFAR-10, while Pseudo-Label achieves the highest accuracy on all three plant datasets.

The main results of this thesis can be stated as follows:

- The proposed self-training using the metric learning approach introduced in Chapter 4 always obtained higher accuracy than using the same number of labelled examples for supervised training. On the majority of the

datasets, with the small networks considered in this chapter, triplet loss loss achieves the highest test accuracy.

- The 1-nearest-neighbour confidence measure for selecting pseudo-labels performs slightly better than the LLGC-based score (Chapter 4).

- The self-training approach based on transfer learning proposed in Chapter 5 consistently achieves higher accuracy than using randomly initialised weights and different neural network architectures for all datasets but SVHN. Transfer learning provides high benefits when a few labelled examples are available for training.

- Pretrained network weights on similar or smaller image sizes yield higher test accuracy than pretrained weights from greater image sizes. Pretrained network weights obtained using cross-entropy loss obtain higher accuracy on cross-entropy and metric learning losses applied to the target task than a pretraining network using triplet loss (Chapter 5).

- Applying different shallow classifiers on embeddings produced by metric learning losses does not significantly improve accuracy. However, LDA is a very fast method to apply (Chapter 5).

- For the majority of the datasets, with the larger networks considered in this chapter, cross-entropy loss achieves higher test accuracy than triplet loss for self-training using randomly initialised and ImageNet pretrained weights (Chapter 5).

- After each iteration of self-training, pseudo-labels are merged into labelled examples for retraining. Incorrect labels are likely to be selected as pseudo-labels and used for training. This gives rise to confirmation bias (Arazo et al., 2020). To overcome this, we experimented with a fixed minimum number of initially labelled examples per minibatch during training, but this does not provide any performance improvement (Chapter 5).

- Chapter 6 presented three approaches using self-supervised training, namely CT, SS-Pretrain, and STSSC. CT and STSSC achieve higher accuracy than the transfer learning approach introduced in Chapter 5 on ImageNet pretrained weights on the majority of the datasets. However, when using ImageNet pretrained weights, SS-Pretrain exhibited a slight decrease in performance compared to self-training using the transfer learning approach introduced in Chapter 5.

- STSSC achieved the highest test accuracy among the three self-supervised learning approaches for self-training. This suggests that self-supervised learning is unnecessary for each iteration of self-training as performed in CT (Chapter 6): performing it in the first iteration is the best of the three options considered.

7.2 Future Work

The results obtained in this thesis open several directions for further research. In this section, we discuss possible extensions to self-training proposed in Chapters 4 and 5.

- For the selection of pseudo-labels, this thesis considered simple 1-nearest-neighbour and LLGC-based (Zhou et al., 2004) confidence measures. For a more robust confidence measure, one could consider using information obtained from triplets.

- The methods presented in this thesis considered pseudo-labels as hard labels, but one can investigate the weighted sum of the loss of initially labelled and pseudo-labelled examples (Sohn et al., 2020).

The approaches based on self-supervised learning introduced in Chapter 6 can be extended as follows:

- Among self-supervision based self-training approaches introduced in Chapter 6, STSSC applied self-supervised combined training for the first it-

eration, followed by the usual self-training and achieved the highest accuracy. It will be of great interest to see if combined self-supervised learning can improve accuracy when applied to more than one iteration.

- For self-supervision, Chen et al. (2020a) introduced the usage of contrastive loss, where an image and its augmented version are considered positives. The rest of the images in the batch are considered negatives. The most straightforward extension to this method would be to use triplet loss for self-supervision.

Finally, all three research directions pursued in this thesis are based on a single idea that using metric learning, transfer learning, and self-supervised learning improves the accuracy of deep neural networks for image classification learned using self-training. Metric learning using triplet loss achieved competitive accuracy compared to cross-entropy loss. Transfer learning always improves the model's predictive performance. Combined training of self-supervised and supervised learning was investigated for self-training; it improves the performance when applied in the first iteration. Investigating more robust confidence measures for the selection of pseudo-labels would be of great interest. For instance, considering the information provided by triplets. For self-supervised learning, this thesis considered a pretext task based on geometric transformations. Applying self-supervised tasks based on image distortion, for instance, ExemplarCNN (Dosovitskiy et al., 2015) based pretext tasks, is one of the logical continuations of the work presented.

Bibliography

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Florent Perronnin, Jorge Sánchez, and Yan Liu. Large-scale image categorisation with explicit data embedding. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2297–2304. IEEE, 2010.

Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalisation. *Communications of the ACM*, 64(3):107–115, 2021.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a Siamese time-delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744. Morgan Kaufmann, 1993.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. MIT Press, 2017.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, editors. *Semi-Supervised Learning*. The MIT Press, 2006. ISBN 9780262033589. doi: 10.7551/mitpress/9780262033589.001.0001.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defence of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017. URL `http://arxiv.org/abs/1703.07737`.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Sergey Ioffe and Christian Szegedy. Batch normalisation: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbour classification. *Journal of Machine Learning Research*, 10(2), 2009.

Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimisation methods from a machine learning perspective. *IEEE Transactions on Cybernetics*, 50(8):3668–3681, 2019.

Yurii Nesterov. A method for unconstrained convex minimisation problem with the rate of convergence o (1/k^ 2). In *Doklady an USSR*, volume 269, pages 543–547, 1983.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimisation. *Journal of Machine Learning Research*, 12(7), 2011.

Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimisation. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR*, 2015.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan,

Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016. URL `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, volume 1681 of *Lecture Notes in Computer Science*, page 319. Springer, 1999.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on Deep Learning and Unsupervised Feature Learning*, volume 2011, page 5, 2011.

Alex Krizhevsky and G Hinton. Convolutional deep belief networks on CIFAR-10. `https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf`, 2010.

David P. Hughes and Marcel Salathé. An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. *CoRR*, abs/1511.08060, 2015. URL `http://arxiv.org/abs/1511.08060`.

Ümit Atila, Murat Uçar, Kemal Akyol, and Emine Uçar. Plant leaf disease classification using EfficientNet deep learning model. *Ecological Informatics*, 61:101182, 2021.

Geoffrey J McLachlan. Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *Journal of the American Statistical Association*, 70(350):365–369, 1975.

Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

Tony Jebara, Jun Wang, and Shih-Fu Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 441–448, 2009.

Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with Local and Global Consistency. In *Advances in Neural Information Processing Systems*, pages 321–328, 2004.

Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.

C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05)*, volume 1, pages 29–36, 2005.

Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013.

Isaac Triguero, Salvador García, and Francisco Herrera. Self-labelled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information systems*, 42(2):245–284, 2015.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Kamal Nigam, Andrew McCallum, and Tom M. Mitchell. Semi-supervised text classification using EM. In Olivier Chapelle, Bernhard Schölkopf, and

Alexander Zien, editors, *Semi-Supervised Learning*, pages 32–55. The MIT Press, 2006. doi: 10.7551/mitpress/9780262033589.003.0003.

Avrim Blum and Tom Mitchell. Combining labelled and unlabelled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100. ACM, 1998.

Ulf Brefeld and Tobias Scheffer. Co-EM support vector learning. In *Proceedings of the 21st International Conference on Machine Learning ICML*, page 16. ACM, 2004.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, pages 1096–1103, 2008.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pretraining help deep learning? In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

Mohammad Pezeshki, Linxi Fan, Philemon Brakel, Aaron Courville, and Yoshua Bengio. Deconstructing the ladder network architecture. In *International Conference on Machine Learning*, pages 2368–2376, 2016.

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, pages 1195–1204, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. FixMatch: Simplifying semi-supervised learning with consistency and confidence. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems, December 6-12, virtual*, 2020.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. RandAugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. ReMixMatch: Semi-supervised learning with

distribution matching and augmentation anchoring. In *International Conference on Learning Representations*, 2019.

David Yarowsky. Unsupervised word sense disambiguation rivalling supervised methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, 1995.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Elad Hoffer and Nir Ailon. Semi-supervised deep learning by metric embedding. In *5th International Conference on Learning Representations, ICLR*, 2017.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29):861, 2018.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International Conference on Artificial Neural Networks*, pages 270–279. Springer, 2018.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, 2014.

Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better ImageNet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimisation. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. *arXiv preprint arXiv:1707.08819*, 2017.

Eric Arazo, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. Pseudo-labelling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. Big self-supervised models are strong semi-supervised learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems, December 6-12, 2020, virtual*, 2020b.

Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1485, 2019.

Phi Vu Tran. Exploring self-supervised regularisation for supervised and semi-supervised learning. *arXiv preprint arXiv:1906.10343*, 2019.

Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1734–1747, 2015.

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

Avital Oliver, Augustus Odena, Colin Raffel, Ekin D Cubuk, and Ian J Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3239–3250, 2018.

Appendix A

Hyperparameters

A.1 Hyperparameters for Datasets

The number of initially labelled examples, sigma σ for LLGC, selection $p\%$ for pseudo-labels, and batch size used for each dataset in different experiments are listed in Table A.1.

A.2 Hyperparameters for Loss Functions

The optimiser, learning rate and margin parameter used for all four losses are listed in Table A.2.

Table A.1: Hyperparameters for datasets

Dataset	**Initially labelled**	**sigma σ**	**Selection p%**	**Batch size**
MNIST	100	2.0	10	100
Fashion-MNIST	100	3.2	10	100
SVHN	1000	2.4	5	100
CIFAR-10	4000	2.4	5	100
PlantVillage	380	2.4	2	64

Table A.2: Hyperparameters for loss functions

Loss	**Optimiser**	**Learning rate**	**Margin**
cross-entropy	Adam	10^{-4}	–
Triplet	Adam	10^{-4}	1.0
Contrastive	RMSProp	10^{-4}	1.0
ArcFace	Adam	10^{-3}	0.5

Appendix B

Overlap between CIFAR-10 and ImageNet

We used a WRN-28-2 network pretrained on ImageNet32 to detect duplicate images present in both the CIFAR-10 test set and the ImageNet32 training set. The squared Euclidean distance was calculated on $l2$-normalised embeddings of the network. Sample images are shown in Figure B.1 with labels on the top, a) top row contains CIFAR-10 test image b) bottom row ImageNet32 training images. The images are shown based on increasing Euclidean distance. For each CIFAR-10 test image, the closest ImageNet32 training image is investigated as a potential duplicate. We visualised the first 600 pairs sorted by distance and found that there were 144 duplicate images.

Figure B.2 shows the frequency of duplicates for the first 600 image pairs in intervals of 50 sorted by distance. It is evident that the number of duplicates reduces as the distance increases.

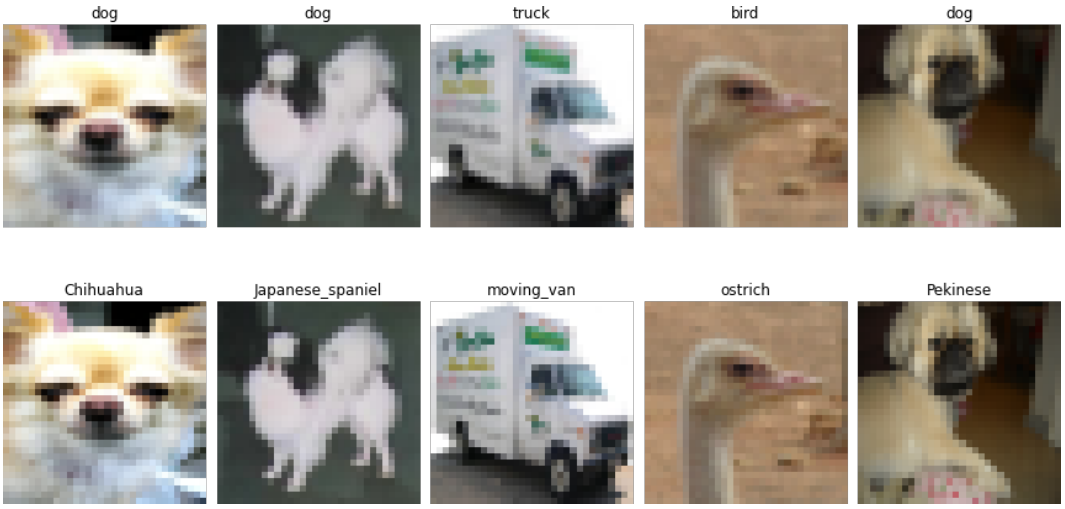


Figure B.1: CIFAR-10 test images at the top and closest matching ImageNet32 training images at the bottom.

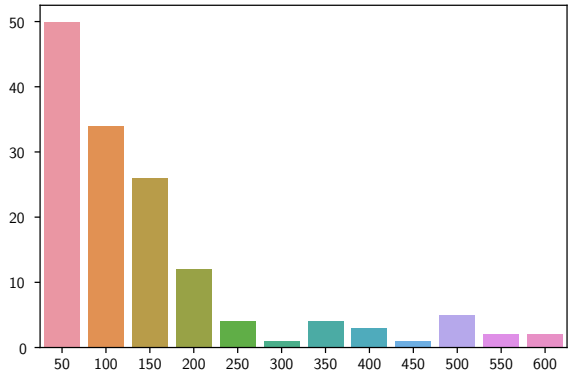


Figure B.2: Histogram showing number of duplicates in 50-interval based on sorted distance.