School of Computer Science
The University of Auckland
New Zealand

Improving the Generalisation of Neural Networks with Unlabelled Data

Alex Yuxuan Peng

October 2021

Supervisors:

Yun Sing Koh

Patricia Riddle

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science, the University of Auckland, 2021.

Abstract

Neural networks have attracted a lot of attention from both academia and industry due to their success in challenging tasks such as machine vision, speech recognition and natural language processing. However, in order to achieve good performance on supervised learning tasks, neural networks have to be trained on large amounts of labelled training data. This often requires human experts to painstakingly label every single example in the data. The labelling process can be costly and time consuming. Even though labelled data can be scarce and expensive to collect in practice, unlabelled data are usually available in abundance. The goal of this thesis is to improve the generalisation performance of neural networks on classification tasks by utilising the unlabelled data. This thesis studies three strategies to tackle this problem: *pretraining*, *semi-supervised learning* and *active learning*.

In this thesis, we propose a self-supervised pretraining method for tabular data that learns to identify real data from randomly shuffled data. Then the weights learned in the pretraining are reused as initial weights for the original task on the labelled training set. In the second piece of work, we break the common assumption in semi-supervised learning that the labelled data and unlabelled data come from the same distribution. We empirically show that novel classes in unlabelled data can lead to a degradation in generalisation performance for semi-supervised algorithms. We propose a 1-nearest-neighbour based method to assign a weight to each unlabelled example in order to reduce the negative effect of novel classes in unlabelled data. Lastly, we propose a new uncertainty-based active learning method specifically for neural networks trained using stochastic gradient descent by querying examples whose predictions change the most during the training. Experimental results show that the proposed method is more effective when a large labelled training set is already available. We also show that different types of active learning methods perform differently under different settings. It suggests that to fully evaluate the characteristics of an active learning algorithm, experiments under a wide range of settings are required.

Acknowledgements

Throughout the journey of my PhD study and the writing of this thesis, I have received a great deal of help and support. First and foremost, I would like to thank my supervisors, Dr. Yun Sing Koh and Dr. Patricia Riddle, for your invaluable advice and patient explanation. Under your guidance, not only have I learned to do research independently, I have also grown as a person. I would also like to thank Professor Bernhard Pfahringer from University of Waikato, for the discussions on my projects and helpful suggestions throughout this journey.

I would like to express my gratitude to all my labmates, colleagues and organisers at the KMG Research Group, the Machine Learning Group and the AI Reading Group, for the interesting and inspiring discussions and presentations. Thank you all for the valuable feedback and comments on my own presentations and research.

I would also like to thank the staff at Centre for eResearch for providing additional computing resources and technical support. I also want to thank the School of Computer Science for providing financial support by granting me several PhD Tuition Fee Awards.

Last but not least, I would like to thank my family, especially my parents, for your unconditional support and understanding. You are the reason why I am free to pursue my own dreams and passions.

Contents

List of Publications

Parts of Chapter 3 and Chapter 4 have been published at:

- Alex Yuxuan Peng, Yun Sing Koh, Patricia Riddle, and Bernhard Pfahringer (2019) Using supervised pretraining to improve generalisation of neural networks on binary classification problems. In: Berlingerio M., Bonchi F., Gärtner T., Hurley N., Ifrim G. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018. Lecture Notes in Computer Science, vol 11051. Springer, Cham. `https://doi.org/10.1007/978-3-030-10925-7_25`

Parts of Chapter 3 and Chapter 5 have been published at:

- Alex Yuxuan Peng, Yun Sing Koh, Patricia Riddle, and Bernhard Pfahringer (2019). Investigating the effect of novel classes in semi-supervised learning. In Asian Conference on Machine Learning (pp. 615–630). PMLR.

List of Figures

List of Tables

List of Notations

β	A hyperparameter for transforming a distance into a weight.
ϵ	The momentum hyperparameter used in stochastic gradient descent with momentum.
η	A hyperparameter that controls the decay rate of λ in exponential moving average.
γ	A hyperparameter that controls the decay of learning rate during training.
$\hat{y}$	Predicted output of a neural network for an example.
λ	The hyperparameter in exponential moving average.
θ	All the trainable parameters of a neural network.
a	A hyperparameter used to combine two terms in a loss function.
b	Bias of a hidden unit.
C	The number of classes in a multi-class classification problem.
c	Any output unit in a multi-class classification problem.
d	Distance between data examples.
f	Activation function.
h	The value of a hidden unit.
J	A loss function.

K	The number of committees in an ensemble.
k	The kth output unit.
L	Labelled dataset.
l	The number of layers in a neural network.
lr	Learning rate.
M	The time complexity of matrix multiplication.
m	The number of labelled examples.
m'	The number of unlabelled examples.
P	Probability. Also used to represent the probability output of a softmax output function.
p	The fan-out of a layer.
q	The fan-in of a layer.
s	The current training step.
t	The current training epoch.
U	Unlabelled dataset.
W	The weight matrix/tensor connecting two layers.
w	The weight value of a single connection.
x_i	An example from the labelled dataset.
x_j	An example from the unlabelled dataset.
y_i	The label of a labelled example.
z	The weighted sum of the units from the previous layer. It is also the input of the next layer.

1 Introduction

Neural networks have attracted a lot of attention from both academia and industry due to their success in different machine learning tasks. Recurrent neural networks (RNNs) have been successfully applied to speech recognition and natural language translation [39, 98, 113]. Convolutional neural networks (CNNs) have been the winning models for many machine vision challenges [40, 44, 67]. Fully-connected neural networks (FNNs) were shown to be effective at identifying exotic particles without features hand-engineered by physicists [3]. The success of these supervised-learning systems relies on an enormous amount of labelled training data. Clean and labelled training data are not always easy and cheap to obtain in many domains.

For example, collecting labelled training data for medical applications usually requires hiring domain experts to manually tag training examples. The World Health Organisation (WHO) reported that pneumonia resulted in more than 2 million deaths in children under 5 years old [95]. Developing a system that can accurately and reliably detect pneumonia can improve the speed and accuracy of diagnosis of pneumonia. Kermany et al. [55] applied neural networks in detecting pneumonia from chest X-ray images. The training data contain 5232 images (3883 positive and 1349 negative). They achieved an accuracy of 92.8% on the test set (390 positive and 234 negative). In order to improve the accuracy, a lot more labelled training examples are required for supervised learning. However, data labelling can be an expensive and time-consuming practice.

Fortunately, although labelled data can be difficult to obtain, unlabelled data are usually readily available in abundance. Therefore, it is desirable to develop machine learning systems that can take advantage of this abundance of unlabelled data. This **motivates** us to study how unlabelled data can be used to improve the generalisation performance of a classifier.

1.1 Problem Definition

In this thesis we define **generalisation** as the ability to generalise a learned classifier on future unseen data with the same distribution as the training data. Generalisation is measured using *test accuracy* (accuracy on the test set) throughout this thesis. The test set has the same distribution as the labelled training set.

In supervised learning, the goal is to train a classifier on a labelled dataset and make accurate predictions on future unseen data. The learning scenario we study in this thesis consists of both a **labelled** dataset L and an **unlabelled** dataset U. The labelled dataset is a set containing pairs of examples and labels: $L = \{(x_1, y_1), ..., (x_{|L|}, y_{|L|})\}$. We use i to denote the index of a labelled example. The unlabelled dataset is a set of examples without labels: $U = \{x_{|L|+1}, ..., x_{|L|+|U|}\}$. We use j to denote the index of an unlabelled example. Both the labelled dataset and the unlabelled data come from the same task (for example, image recognition). The **goal** of this thesis is to improve the generalisation performance of neural networks on classification tasks with unlabelled data.

1.2 Outline

In order to improve the generalisation of neural networks, we study three different strategies of utilising the unlabelled data: *pretraining*, *semi-supervised learning* and *active learning*.

One of the earliest approaches of utilising unlabelled data to improve the performance of neural networks on supervised tasks is pretraining [5, 46, 92]. However, the main goal was to improve the training stability of neural networks, instead of the generalisation performance. Figure 1.1 demonstrates a typical pretraining process.

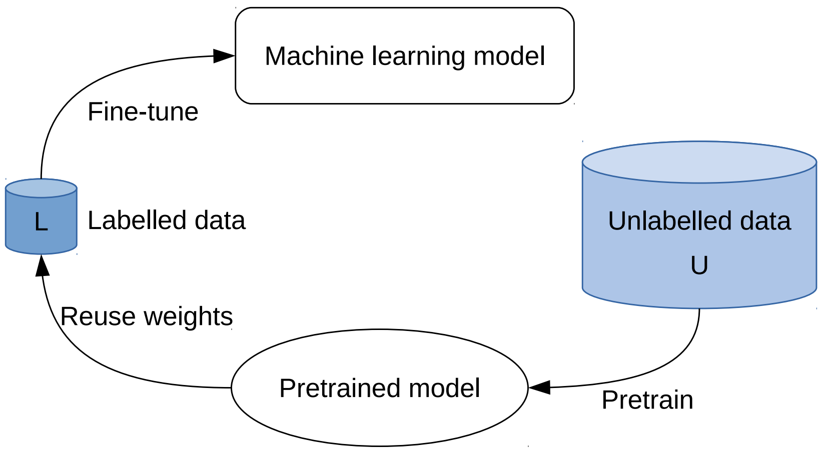


Figure 1.1: Pretraining.

During the pretraining, a model is learned from the unlabelled data. The learned weights are then reused as initial weights of the final model and fine-tuned on the labelled dataset. Pretraining assumes that the weights learned in the pretraining process are useful for the original supervised task. In Chapter 4, we propose a pretraining method that is supervised in nature by automatically creating a labelled dataset from the unlabelled data. The goal is to improve the generalisation performance on the original classification task by using the learned weights as initial weights.

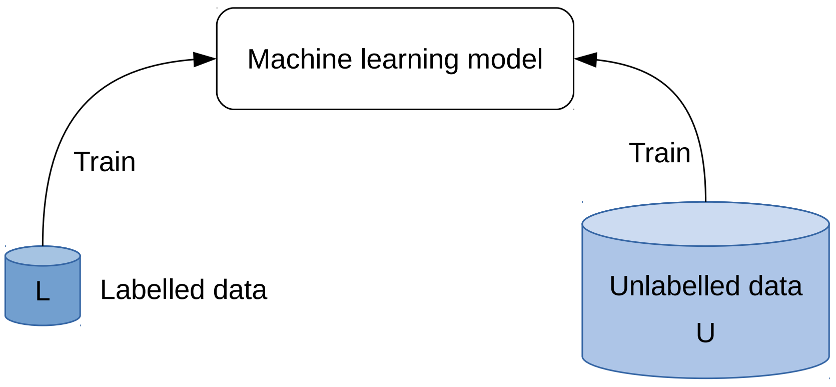


Figure 1.2: Semi-supervised learning.

Pretraining is a two-phase strategy of utilising unlabelled data. Semi-supervised learning provides a one-phase strategy for taking advantage of the unlabelled data. In semi-supervised learning, a classifier can be trained on labelled data and unlabelled data simultaneously (shown in Figure 1.2). This is achieved by optimising a loss function that utilises both labelled data and unlabelled data. Semi-supervised learning assumes that the unlabelled data comes from the same distribution as the labelled data. However, this assumption may not hold in practice. We will show in Chapter 5, the presence of novel classes in the unlabelled data can degrade the performance of semi-supervised algorithms for neural networks. We propose a general distance-based weighting framework and a 1-nearest-neighbour-based implementation that assigns weights to unlabelled data in order to reduce the impact of novel classes.

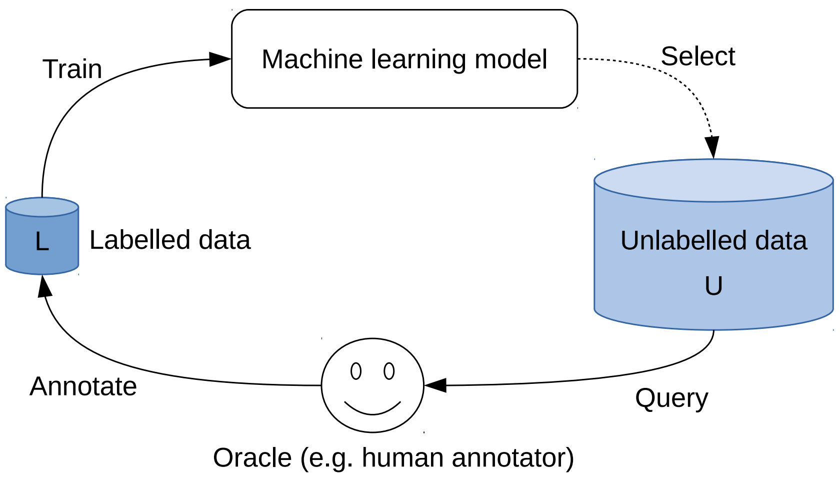


Figure 1.3: Active learning.

Another strategy of improving the generalisation performance of a classifier using unlabelled data is simply labelling more data. Strategically selecting unlabelled data to be queried and added to the labelled dataset is called active learning. A typical active learning cycle is shown in Figure 1.3. A model trained on the available labelled data is used to select examples from the unlabelled dataset (the dashed line indicates that not all active learning algorithms require a trained model). The selected examples are then sent to an oracle, typically a human annotator, for annotation. The additional labelled data are added to the labelled training set. The model is then retrained on the updated training set. Most literature focuses on the improvement of generalisation performance when the labelled dataset is small, possibly due to the common evaluation methods used. In Chapter 6, we show that different types of active learning methods perform differently under different settings. It means that an active learning algorithm optimised for small labelled dataset may not be as effective when the labelled dataset is already big. We propose a new uncertainty-based active learning framework and an implementation of it, inspired by recent research on the convergence property of neural networks [86, 110, 117, 124]. The proposed method is more effective when the initial labelled dataset is large. An active learning algorithm that is more effective when labelled dataset is large is desirable, because practitioners usually want to continuously improve the generalisation of their models even after the models are in production.

1.3 Research Questions

We answer the following questions for each of the strategies we study throughout this thesis:

Pretraining

1. How do the initial weights affect the generalisation performance of a neural network?

2. How can we create a supervised pretraining task from the unlabelled data to pretrain a model, and then reuse the learned weights as initial weights in order to improve the generalisation on the original supervised task?

Semi-supervised learning

1. How do novel classes in unlabelled data affect the generalisation performance of semi-supervised algorithms for neural networks?

2. How can we come up with a method to mitigate the negative effect of novel classes on the generalisation performance of neural networks?

Active learning

1. How do different types of active learning methods perform under different settings on the number of initial labelled examples and the number of queried

examples?

2. How can we apply recent theoretical and empirical studies on the convergence properties of neural networks to active learning?

1.4 Scope

This thesis only focuses on the generalisation performance of neural networks on classification tasks. We do not study other machine learning tasks, such as regression, in this thesis. We also do not consider *multi-label* classification, where an example can have multiple labels. An example can only have one class label in this thesis.

As the title suggests, we focus on neural networks in this thesis. Other types of machine learning classifiers are not considered in our research. The architecture of a neural network is a form of inductive bias, so it can have an impact on the generalisation performance. However, designing novel architectures is not a goal of this thesis. Additionally, we do not use state-of-the-art architectures or any data augmentation techniques in our experiments. We do not intend to achieve state-of-the-art results on any specific datasets. In order to answer our proposed research questions, we only need to make sure all the variables (including the architectures used) are consistent in all competing methods. Therefore, we used simple architectures in our experiments to reduce the computational cost.

A common approach to improving the generalisation of neural networks when the labelled training set is small is *transfer learning*. However, transfer learning requires a large labelled dataset from a different but related task, apart from the original labelled dataset. This violates the problem definition of this thesis (only one labelled dataset and one unlabelled dataset from the same task are available). Therefore, transfer learning is not considered in this thesis.

1.5 Objectives

In order to answer our proposed research questions, we define the following objectives for this thesis:

Pretraining

1. To demonstrate the effect of initialisation on the generalisation of neural networks.
2. Propose a supervised pretraining method to improve the generalisation of neural networks.
3. Evaluate the proposed pretraining method against a standard initialisation method commonly used for modern neural networks.

Semi-supervised learning

1. Conduct empirical study on the effect of novel classes on the performance of semi-supervised algorithms for neural networks.
2. Propose a method to mitigate the negative effect of novel classes in unlabelled data when applying semi-supervised learning algorithms.
3. Evaluate the effectiveness of the proposed method.

Active learning

1. Conduct experiments to test different types of active learning methods under different settings.
2. Propose a new active learning method motivated by the recent research on the convergence property of neural networks.
3. Evaluate the proposed active learning method against other benchmarks under different settings.

1.6 Contributions

This thesis makes the following contributions:

Pretraining

- We demonstrate that initialisation does affect the generalisation performance of neural networks. We show that "bad" initial weights can lower the test accuracy of a neural network. The evaluation results on our proposed pretraining method suggest that "good" initial weights can improve the generalisation performance especially when the labelled training set is small.

- We propose a supervised pretraining method for tabular data. The proposed method creates a binary classification task for the pretraining automatically from the unlabelled data. It creates a shuffled dataset by randomly permuting each feature independently. The pretraining trains a model to identify the real data from the shuffled data. The experimental results show that the pretraining leads to better test accuracy when the labelled training set is small.

Semi-supervised learning

- We empirically show that the presence of novel classes in the unlabelled data can lead to degradation in generalisation performance of semi-supervised learning algorithms.
- We propose a distance-based weighting framework that assigns weights to unlabelled data. This framework assumes that the unlabelled examples that are far away from the labelled examples are more likely to belong to the novel classes, and should be assigned lower weights in training. The proposed framework can be applied to any semi-supervised learning algorithm that includes unlabelled data in the loss function.
- We propose a 1-nearest-neighbour based implementation of the framework. The experimental results show that when the proposed method is applied to semi-supervised algorithms, the degradation in generalisation performance caused by novel classes becomes statistically insignificant.

Active learning

- We propose a new uncertainty-based active learning framework for neural networks that selects examples whose model outputs fluctuate the most during training. This framework assumes that the unlabelled examples whose model outputs fluctuate the most during training are more difficult to learn, and therefore are more useful to be added to the labelled set.
- We propose an implementation of the framework that queries unlabelled examples whose predictions change the most during training. The experimental

results show that the proposed method is more effective when the initial labelled dataset is large. We think this active learning setting where a large labelled dataset is already available has been overlooked due to the common evaluation methods used in the literature.

- We empirically show that different types of active learning algorithms perform differently under different settings. This suggests that researchers should evaluate their proposed active learning algorithms under a wide range of settings in order to reveal the characteristics of the algorithms. A more comprehensive evaluation can also motivate researchers to design active learning algorithms optimised for specific settings.

1.7 Thesis Structure

The rest of the thesis is structured as follows:

- *Chapter 2: Preliminary Overview of Neural Networks.* This chapter introduces preliminary concepts that help the reader understand later chapters. We first explain what an artificial neural network is. Then we introduce main components found in modern neural networks.

- *Chapter 3: Literature Review.* This chapter reviews relevant literature on improving the generalisation of classifiers using unlabelled data. We first review related works on using pretraining to improve the generalisation of neural networks. Then we review important semi-supervised algorithms for neural networks. Finally, we discuss relevant literature on active learning.

- *Chapter 4: Supervised Pretraining with Unlabelled Data.* In this chapter, we demonstrate that the initial weights can affect the generalisation of neural networks. We then describe the proposed pretraining method before discussing the experimental results.

- *Chapter 5: Investigating the Effect of Novel Classes in Semi-Supervised Learning.* This chapter breaks the common assumption in semi-supervised learning that the unlabelled data come from the same distribution as the labelled training data. We empirically investigate whether the presence of novel classes affect the performance of semi-supervised learning algorithms for neural networks. We then propose a method to reduce the effect of novel classes on the performance of semi-supervised learning algorithms.

- *Chapter 6: Measuring Output Fluctuation During Training for Active Learning.* In this chapter, we propose a new active learning method inspired by the recent theoretical and empirical works on the convergence property of neural networks. We evaluate the proposed method against different types of active learning methods under different settings by changing the number of initial labelled examples and the number of queried examples.
- *Chapter 7: Conclusion.* We conclude the thesis by highlighting our achievements and contributions, pointing out limitations of our research and finally discussing potential future works.

2 Preliminary Overview of Neural Networks

2.1 Overview

In this chapter, we briefly introduce what a neural network is and describe the important components of a typical neural network. Generally speaking, all neural networks have an input layer, at least one hidden layer and an output layer. The input layer connects the input (data) to the hidden layer. Hidden layers perform nonlinear transformations of their input by using nonlinear **activation functions**. Nonlinear activation function is an important reason why a neural network can represent nonlinear functions. Different activation functions have different properties and can affect both the representation capacity and the optimisation of neural networks. Activation functions are discussed in detail in Section 2.5. The output layer is used to change the size of its input to the desired number of output units. In classification tasks, the output layer is also used to transform its input into a probability distribution over the different classes. A **loss function** is then applied to the output of the output layer to evaluate how accurate the predictions are against true labels. Each layer is connected to at least another layer with a set of weights. The same activation functions, output functions and loss functions can be used across different types of neural networks. Different types of neural networks mainly differ in how the layers are connected to each other. In other words, different neural networks have different structures of weights that connect the layers together. This is called the **architecture** of neural networks.

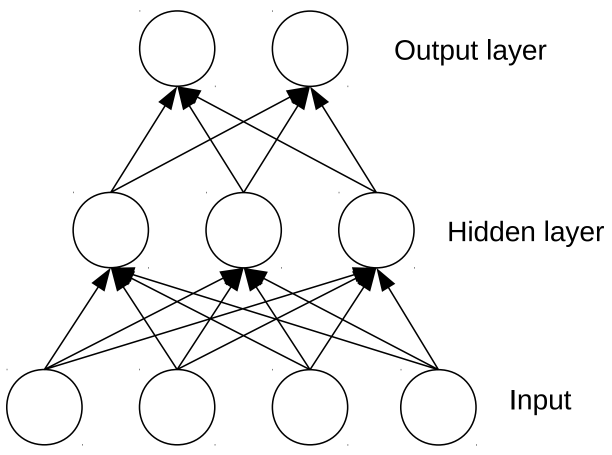


Figure 2.1: A two-layer FNN with one hidden layer. The input has four dimensions. The hidden layer and the output layer have three and two units respectively.

2.2 Architecture

The simplest architecture in neural networks is **fully-connected neural network (FNN)**. FNN is a type of **feedforward neural network**, which does not allow cyclic connections among units or layers. In FNN, each unit is connected to every unit in the next layer, but not to units in its own layer. Some literature refers to this type of neural network as **multi-layer perceptron (MLP)**. This is a misnomer, since perceptron is a linear classifier. A multi-layer perceptron is still a linear classifier. However, the name MLP is almost always used to refer to a non-linear multi-layer FNN. Figure 2.1 is an example of a simple two-layer FNN. In this example, the input of the network has four dimensions. Each of the dimensions in the input is connected to every unit in the hidden layer. Each of the three hidden units is connected to all of the output units. Note that units in the same layer are not connected to each other, and each layer is always connected to the next neighbour layer (there are no skip connections). Each connection in the network has a weight associated with it. Generally speaking, the magnitude of the weight indicates the importance of the connection between two units. The input to a unit in the next layer is a weighted sum of the values of all the units in the current layer. Each unit apart from the input units also has a bias that needs to be added to the sum. For example, let us denote the four input values as x_1, x_2, x_3 and x_4, the bias of the first hidden unit as b and the weights for the incoming connections to this hidden unit as w_1, w_2, w_3 and w_4. The input to this hidden unit (z) is calculated as below:

$$z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b.$$

Each hidden unit has an activation function f, which takes as input the weighted sum of the output values of units from the previous layer. The output of the first hidden unit in our example is therefore $f(z)$. The activation function is usually nonlinear. If a linear activation is used for every hidden unit, then the output of the entire network is also linear. This means we can only learn a linear function regardless of the size of the neural network. Activation functions are discussed in detail in Section 2.5.

Convolutional neural network (CNN) is also a type of feedforward neural network that is widely used in machine vision tasks. LeCun et al. [70] first proposed the idea of using convolution operations on image data to learn feature extractors by taking advantage of the fact that the pixels in natural images are locally related to each other. The idea was successfully applied in the handwritten digit recognition task [71]. The network architecture proposed in this work has been known as LeNet. LeNet has a series of blocks that contain a convolutional layer and a pooling layer. The convolutional layer applies convolution operations on an image or the output of the previous layer. The pooling layer downsamples the output of the convolutional layer. Convolution makes the model translation equivariant for the locations of features learned. It means that if a transformation is applied to an image and then convolution is applied to the same image, the result is the same as what we would get if we applied the convolution before the transformation. However, convolution does not necessarily make the model equivariant to other transformations like scale or rotation. Pooling introduces partial translation invariance to some small translations. It means that the output of pooling can be the same even if the input changes slightly. The basic ideas of LeNet have been applied at much larger scales in architectures used in the ImageNet Large Scale Visual Recognition Challenges (ILSVCRs) [23], such as AlexNet [67], VGG [108], GoogLeNet [114] and ResNet [44]. A layer in a CNN or feedforward network in general only connects to the next layer after it. This is true for all the architectures mentioned above except ResNet. ResNet has skip connections that connect layers that are not next to each other. These skip connections allow gradient information to propagate more easily in deep networks.

Both CNN and FNN are feedforward neural networks that do not have cyclic connections. **Recurrent neural network (RNN)** is a type of neural network in which cyclic connections are allowed. RNNs are designed to learn sequential relationships in the data. However, RNNs had been shown to be really difficult to train to learn long-term dependencies, until the invention of **long short-term memory** (LSTM) [47] and its variations [14, 33, 62, 126]. We do not explain RNNs in detail here, since they were not used in our studies. Note that architecture design is not a goal of the research reported in this thesis, hence we used simple architectures in our experiments to reduce the computational cost.

2.3 Output Function

The output function of a neural network is usually one of three forms: **linear**, **sigmoid** and **softmax**. Let W be the weight matrix between the output layer and the hidden layer before the output layer. Let $\vec{h}$ be the output vector of the hidden layer and b be the bias for the output unit. The linear output is calculated as follows:

$$\hat{y} = W^{\intercal}\vec{h} + b.$$

Linear output is used when we expect the output to be distributed according to the Gaussian distribution. We usually use this linear output function in regression tasks and the networks usually only have one output unit. If the output is a Bernoulli distribution, we usually use sigmoid as the output function. There is also only one output unit in this case. The sigmoid output is computed as follows:

$$z = W^{\intercal}\vec{h} + b, \text{ and}$$

$$\hat{y} = \frac{1}{1 + exp(-z)}.$$

Sigmoid output function is suited for binary classification problems. It squashes the input into the range of $[0, 1]$. If the output is larger than a threshold, we predict the example to be positive, otherwise we predict it to be negative. If the classification problem has more than two classes, a softmax output function can be used. In this case, the number of the output units is equal to the number of classes. Softmax output for one output unit is calculated as follows:

$$\hat{y_k} = \frac{exp(z_k)}{\sum_c exp(z_c)}$$

where k is the kth output unit, z_k is the weighted sum of the incoming signals for the kth output unit and z_c is the weighted sum of the incoming signals for any output unit c. The class with the highest output value is the predicted class. We use the softmax function as the output function in this thesis unless specified otherwise, because we mainly deal with classification problems.

2.4 Loss Function

A neural network also needs a **loss function** that quantifies how accurate its predictions are. Note that the loss function is usually not the same metric used to evaluate a neural network. This is also true for other types of machine learning models. This happens when we cannot apply an efficient optimisation algorithm to optimise the evaluation metric directly. For instance, a common metric used to evaluate a classifier is accuracy, but the most efficient optimisation algorithms for training neural networks cannot be used to directly optimise accuracy on a training set. The common optimisation algorithms for neural networks all require gradient information, however, accuracy is not a smooth and continuous function that these algorithms can operate on. Common optimisation algorithms in neural networks are reviewed in Section 2.7.

A loss function should be able to indicate how well a model is fitting the data, and it should be a function that can be optimised efficiently. Most of the commonly used loss functions in neural networks are derived using **cross-entropy** according to the **Maximum Likelihood Estimation** (MLE). The specific form of the loss function depends on the choice of the output function. If the output is a linear function, the loss function will take the form

$$J = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

where y_i is the ground truth, $\hat{y}_i$ is the predicted output, and m is the number of instances in the data set. If the output is a sigmoid or softmax, then the loss function becomes

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C}(y_{ic}log(\hat{y}_{ic}))$$

where y_{ic} is the ground truth (usually a binary value) for class c of example i, $\hat{y}_{ic}$ is the predicted probability for class c of example i and C is the total number of classes. This is the loss function used through out this thesis unless specified otherwise.

2.5 Activation Function

The choice of activation function for the hidden units is important when designing a neural network. Activation functions need to be nonlinear, otherwise the whole neural network can only represent a linear function. Before the prevalent usage of **rectified linear units** (ReLU), the **sigmoid** function

$$f(z) = \frac{1}{1 + exp(-z)}$$

and the **hyperbolic tangent** function

$$f(z) = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)}$$

were the mostly used activation functions. The output of the sigmoid function is in the range of $[0, 1]$ and $[-1, 1]$ for hyperbolic tangent function. Both of these functions saturate when the magnitude of z is large. The large saturation region is a major draw back of these activation functions when a gradient-based learning algorithm is used. The gradient for either of the functions reaches zero when z is large in magnitude; no gradient information is able to flow to the early layers, this stagnates the training. To overcome this problem, researchers started initialising the weights using layer-wise unsupervised learning [5, 46, 92].

It was later discovered that if ReLU is used as the activation function, one can simply train a neural network using purely supervised training without unsupervised pretraining to achieve high accuracy, provided that a large labelled dataset is available [36]. ReLU takes the form of $f(z) = max\{0, z\}$. ReLU has now become the default activation function to use because of its ease to optimise and fast computation. The right half of ReLU is a linear function and the left half is a constant zero. Empirically, the flat left half of ReLU is usually not a problem for training when the model depth is not too big. However, the flat region of ReLU can result in "dead neurons" when the model is big or the choice of initial weights is poor. "Dead neurons" is a phenomenon where the activation values are stuck in the flat region of the activation function so that the network stops learning. To solve this problem, different variations of piece-wise linear activation functions have been developed. Note that although the ReLU activation function is not differentiable at $z = 0$, in practice we can simply set the derivative at this point to 0 or the slope of the linear part. The same treatment of non-differentiable points can be applied to other variations of piece-wise linear activation functions.

Softplus is a smoothed version of ReLU: $f(z) = ln(1 + exp(z))$ so that anywhere on the function is differentiable [87]. However, Glorot et al. [36] found that ReLU produced better results than softplus. Maas et al. [78] introduced **leaky ReLU**: $f(z) = max(0, z) + a\ min(0, z)$ where a is a small constant value (e.g. 0.01). He et al. [43] adopted the idea of leaky ReLU but made a a learnable parameter (**parametric ReLU**). They showed that by carefully choosing the parameter in leaky ReLU, one could get better results than the regular ReLU. But the parameter tuning process can be tedious. Parametric ReLU is able to learn the parameter from data.

Clevert et al. [16] introduced the **exponential linear unit (ELU)**: $f(z) = max(0, z) + a(exp(z) - 1)$. They claimed that the negative values of ELU pushes the mean activation values to zero where the gradient is well behaved. This has the effect of batch normalisation [52] but without its expensive computation. According to their experiments, ELU gives faster training and better generalisation for networks with more than 5 layers than ReLU and leaky ReLU. Built on top of the idea of ELU, Klambauer et al. [60] proposed **self-normalising neural network (SNN)** using **scaled exponential linear units (SELU)**: $f(z) = \lambda(max(0, z) + a(exp(z) - 1))$.

They derived the values for a and λ using stable and attracting fixed point theory. They theoretically proved that as long as activation values are close to zero mean and unit variance, as it propagates to deeper layers, the activation values will converge to zero mean and unit variance.

Goodfellow et al. [38] introduced **maxout** to work with the dropout [111] regularisation method. Instead of a two-piece linear function like ReLU, maxout has k linear pieces. Maxout is defined as $f(\vec{z}) = max(z_i)$ where $i \in k$. One maxout unit can approximate any convex function up to a certain degree of error. The high representation capacity of the maxout unit means it should be used with more regularisation methods. Dropout is usually applied when maxout units are used.

Although we have seen a lot of research efforts in coming up with new activation functions in recent years, it still remains difficult to know which activation function to use in a particular application without experimenting. Currently, we do not have good theoretical guidance on the choice of what activation function to use. Empirically speaking, ReLU is a good default activation function to start with if the depth of the network is not too big. If during training, the neural network is experiencing the "dead neuron" phenomenon, activation functions such as leaky ReLU, parametric ReLU, ELU can be experimented with. Unless specified otherwise, we always use ReLU as the activation function in our experiments.

2.6 Initialisation

As described in the previous sections, neural networks have weighted connections that connect all the layers together. When we train a neural network, we essentially update these weights connecting the layers (optimisation methods used to train neural networks are discussed in Section 2.7). This means we need to set initial weights for a neural network before the training starts. This is called **initialisation**. Deep neural networks are notoriously sensitive to initialisation [35, 43, 112]. Poor initialisation can lead to poor generalisation, difficulty in training or numerical problems. We will discuss some of the research on initialising neural networks in this section.

First of all, we need to make sure the units in the same layer are initialised with different weights, otherwise these weights will always be updated in the same direction with the same magnitude. This is known as "symmetry breaking".

We usually initialise weights using either a Gaussian distribution with zero mean and small standard deviation or an uniform distribution with a limited range of values to draw from. LeCun et al. [72] recommended using a normal distribution with mean 0 and standard deviation $\frac{1}{\sqrt{p}}$, where p is the number of units in the current layer. Glorot and Bengio [35] derived a way to initialise weights using the following uniform distribution

$$W \sim Uniform\left(-\sqrt{\frac{6}{q+p}}, \sqrt{\frac{6}{q+p}}\right)$$

where W is the weights, q and p are the number of units in the previous layer and the next layer. They derived this method by assuming only linear activation functions are used, and attempting to initialise the weights in such a way that the variances of activation values and gradient across all layers are the same at the beginning of training. Despite the unrealistic assumption on the linear activation function, this initialisation works well in practice. Using the same idea, He et al. [43] derived an initialisation method specifically for the ReLU activation function

$$W \sim Normal\left(0, \sqrt{\frac{2}{q}}\right)$$

or

$$W \sim Uniform\left(-\sqrt{\frac{6}{q}}, \sqrt{\frac{6}{q}}\right)$$

where q is the number of units in the previous layer. This initialisation method has been shown to be effective even for really deep networks using ReLU where the Glorot and Bengio [35] method fails [43].

Saxe et al. [100] recommended initialising weights to random orthogonal matrices. However one needs to carefully choose the gain factor when a nonlinear activation function is used. Mishkin and Matas [82] proposed a method called layer-sequential unit-variance (LSUV) initialisation. LSUV firstly initialises the weights with orthonormal matrices, and then normalises the activation values of each layer to have a variance of 1. Krähenbühl et al. [63] also developed a data-dependent initialisation method for convolutional neural networks in an attempt to make all the weights change at roughly the same rate.

All of the works on initialisation mentioned above have been focusing on stabilising and accelerating the training process. Initialisation of neural networks is a difficult problem to study due to the complex nature of the loss (cost) surface and its optimisation dynamics. It is even more difficult to conduct research on initialisation with regard to the generalisation of neural networks. It is possible that even though the above methods can successfully improve the stability and speed of training, they may not provide the best generalisation performance in neural networks. More research efforts need to be devoted to exploring the connection between the initialisation and the generalisation of neural networks.

2.7 Optimisation

When training a neural network, we want to optimise the weights connecting the layers to minimise the value of a loss function (described in Section 2.4) given the data and true labels. If the loss surface is convex, we can solve the optimisation problem analytically. However, the loss surface of a neural network is almost always non-convex with many local minimums and saddle points, so we have to use an iterative algorithm to update the parameters. There is no guarantee that the model will converge. Even when it does converge, we cannot know if the model has converged to the global minimum or a local minimum.

The most commonly used optimisation methods in training neural networks are all different variants of **gradient descent**. The word "gradient" simply means the partial derivatives of a loss function with respect to a vector of variables. In machine learning we usually have to optimise more than one parameter especially in deep neural networks. A gradient descent algorithm makes small steps in the direction of the negative gradient. Let f be a function of variable w, we denote the derivative of f with respect to w as $\nabla_w f(w)$. Then the weight update rule of gradient descent is

$$w' = w - lr\nabla_w f(w)$$

where lr is the **learning rate**. The learning rate determines the step size we take in the weight update.

We have seen how to update a weight parameter given its derivative, but we also need an efficient algorithm to calculate these derivatives in a neural network. **Backpropagation** [97] is such an algorithm that can compute the partial derivatives of all the weight parameters in a neural network by doing only one forward pass and one backward pass. Backpropagation is basically an application of the chain rule of calculus. We use Figure 2.2 as an example to briefly illustrate how backpropagation works. Let $\vec{z_1}$ and $\vec{z_2}$ be the weighted sum of the unit values of $\vec{x}$ and $\vec{h}$: $\vec{z_1} = W_1^\intercal \vec{x}$

and $\vec{z_2} = W_2^\intercal \vec{h}$. Let $\vec{h} = f(\vec{z_1})$ and $\vec{y} = g(\vec{z_2})$.

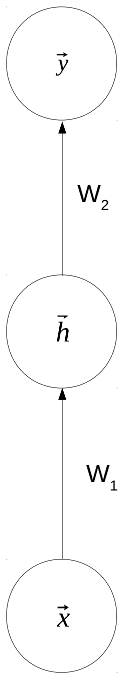


Figure 2.2: An example of a forward pass, where $\vec{y}$ is the output units, $\vec{h}$ and $\vec{x}$ are hidden units and input units respectively, and W_1 and W_2 are weight matrices of the network.

In the first step of backpropagation algorithm, we do a forward pass to calculate the values of the output units $\vec{y}$, and compute the value of the loss function J based on $\vec{y}$. Then we compute the partial derivatives of J with respect to $\vec{y}$: $\nabla_{\vec{y}} J$. The gradient of $\vec{z_2}$ is $\nabla_{\vec{z_2}} J = \nabla_{\vec{z_2}} \vec{y} \odot \nabla_{\vec{y}} J$. The gradient of the weights W_2 is therefore $\nabla_{W_2} J = \vec{h} \nabla_{\vec{z_2}} J$. The gradient of $\vec{h}$ is the weighted sum of the gradient of $\vec{z_2}$: $W_2 \nabla_{\vec{z_2}} J$. By applying the same chain rule, the gradient of W_1 is $\nabla_{W_1} J = \vec{x} \nabla_{\vec{z_1}} J$, where $\nabla_{\vec{z_1}} J = \nabla_{\vec{z_1}} \vec{h} \odot \nabla_{\vec{h}} J$. We usually do not need to compute gradients for input $\vec{x}$ because we are not trying to optimise the input values.

The most commonly used optimisation algorithm is probably **stochastic gradient descent (SGD)**. The weight update rule is

$$\theta \leftarrow \theta - lr \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} J(y_i, \hat{y_i}) \right)$$

where lr is the learning rate and m is the batch size. When m is the size of the training set, this is basically a full-batch version of gradient descent. In practice, we rarely use the full-batch version of gradient descent because it does not scale well with regard to the size of the training set. When m is 1, it becomes the online version of gradient descent. The learning can become less stable when using online gradient descent, so we usually use the mini-batch version of the algorithm. SGD introduces noise in the estimation of gradient through random sampling of mini-batches. Because of this, it is common in practice to reduce the learning rate over time as the training goes on when using SGD. Smith and Le [109] showed that the noise introduced by mini-batches actually helps with generalisation and the optimal batch size is proportional to the learning rate and the size of the training set.

Although SGD is a popular optimisation method in neural network, it has its disadvantages. Learning can be slow and unstable if the noise in the gradient is high. And if the Hessian matrix of the loss function is ill conditioned, SGD tends to zigzag on the cost surface instead of taking a smooth and direct path to the minimum. Polyak [90] introduced **momentum** to help solve these problems. The update rule for SGD with momentum is

$$v \leftarrow \epsilon v - lr\nabla_\theta(\frac{1}{m}\sum_{i=1}^{m} J(y_i, \hat{y}_i))$$

$$\theta \leftarrow \theta + v$$

where v is velocity, ϵ is the momentum parameter and lr is the learning rate. The larger ϵ is compared to lr, the more the previous gradients will affect the direction of the current weight update. Momentum draws inspiration from physics. Imagine a ball is travelling at a speed of v on a surface. The friction of the surface determines the momentum parameter ϵ. If the surface is completely smooth, ϵ would be 1. If the surface is not smooth, the ball will continue moving in the same direction until its speed decreases to zero. The second term in the velocity update rule is like another force applied to the ball that can potentially change the direction and magnitude of the velocity of the travelling ball. A slight variant of the momentum method called **Nesterov** momentum was later introduced by Sutskever et al. [112]. The only difference in Nesterov momentum is that the current velocity is applied to the weights before the gradient is computed.

Jacobs [53] proposed a method to adaptively adjust the learning rate for each weight during training. If the partial derivative of a weight parameter changes sign, then the learning rate is decreased. We increase learning rate for a weight parameter if its derivative does not change sign. Recently, more advanced methods with adaptive learning rates have been developed. **AdaGrad** [28] accumulates the squared derivatives for weight parameters and scales the learning rates inversely proportional to the the accumulated squared derivatives. However, AdaGrad works the best for convex problems. **RMSProp** [45] modifies AdaGrad by using an exponentially weighted moving average to accumulate the squared gradient. Zeiler [128] introduced **AdaDelta** to address two drawbacks of AdaGrad: sensitivity to the choice of global learning rate and the continual decay of learning rates during the training. **Adam** draws inspirations from both the momentum method and RMSProp and attempts to adaptively adjust momentum instead of learning rate [57]. Similar to the choice of activation functions, we currently do not know which optimisation method is the best for a particular dataset without experimenting.

All the optimisation algorithms we have discussed are first-order methods. There are also many methods that take second-order derivatives into consideration. However, second-order optimisation methods are not commonly used in training deep neural networks mainly due to their poor scalability and convergence requirements [71]. Optimisation is not the centre of our research and we will not be using second-order optimisation methods in our research, so we are not reviewing them here. Unless specified otherwise, we always use SGD in our research.

2.7.1 Normalisation

Another important advancement in neural networks in the last few years is the invention of different normalisation methods that are designed to stabilise the training of deep models. When the depth of a neural network is big, the variance of the activation values and gradient values becomes large. This can result in training problems such as vanishing gradients or exploding activation values. Researchers attempted to solve this problem by using carefully designed initialisation, the usage of ReLU and a small learning rate. However, even though the initialisation methods described in Section 2.6 are designed to stabilise the training of deep neural networks, they only impose this stabilising property at the initialisation and the first few training iterations. The stabilising effect provided by good initialisation may not continue afterwards. The usage of ReLU and its variants helps reduce the problem of vanishing gradients, however, the stability of training very big models is still a problem. And using small learning rates obviously slows down the learning process.

Shimodaira [128] described a problem encountered in learning systems called **covariate shift**. Covariate shift happens when the input distribution of a learning system changes all the time and the model has to constantly adapt to new distributions. Inspired by this work, Ioffe and Szegedy [52] discussed the idea that treats each layer of a neural network as the input to the remaining sub-network, and proposed **batch normalisation** to normalise the activation values across examples in a mini-batch and therefore reduce the covariate shift during training. Batch normalisation can be applied to the input of a nonlinear layer or the output. It is necessary to experiment both to decide which approach works better for a specific problem. Even though it was proposed to stabilise the training for deep networks, batch normalisation has also been shown to have a regularisation effect [52].

Despite its effectiveness on feedforward neural networks, batch normalisation is difficult to apply to recurrent neural networks and the effectiveness is dependent on the batch size. **Weight normalisation** attempts to stabilise training deep neural networks by normalising the weights instead of the input or the output of activation functions [99]. Unlike batch normalisation, weight normalisation is a deterministic method and does not depend on mini-batches. **Layer normalisation** is another normalisation method that does not depend on mini-batches [2]. The difference between layer normalisation and batch normalisation is that layer normalisation computes the mean and standard deviation from all the inputs to a layer for a single training example, while batch normalisation computes these parameters from a batch of examples for each unit in a layer. Both weight normalisation and layer normalisation can be applied to recurrent neural networks.

2.8 Regularisation

In the previous sections, we reviewed recent advancements in improving training deep neural networks. However, optimisation in neural network, and machine learning in general, is different to optimisation in mathematics. In machine learning, the goal of optimisation is usually not to minimise the loss function for the training set. In other words, fitting the training data perfectly is not always desirable in machine learning. The goal of training on a training set is to learn a model that generalises on future unseen data. When the representation capacity of a model is large, and the training data is noisy or small in size, the model is prone to overfitting. A simple way to tell if a model is overfitting is check the training accuracy and compare it to the validation accuracy. If training accuracy is high but validation accuracy is low, then this is usually overfitting. However, if both training and validation accuracy are low then this is underfitting. The representation capacity of a deep neural network makes it prone to overfitting. In this section, we review commonly used regularisation methods to reduce overfitting in neural networks.

A common method used to reduce overfitting in neural networks is adding a penalty term to the loss function to limit the capacity of the model.

$$J'(\theta, x, y) = J(\theta, x, y) + aJ_{penalty}(\theta)$$

where J is the loss function that depends on the parameters θ, input values x and true labels y. The second term is a penalty function $J_{penalty}$ depending on the model parameters multiplied by a hyperparameter a. Commonly used penalty functions include L_2 norm and L_1 norm. The assumption here is that the smaller the norm of the weights, the "simpler" the model is. This method is often called **weight decay**. Using L_1 norm as the penalty term usually results in a lot of weight parameters reaching zero. This can be useful if a network with sparse connections is desirable.

Another common regularisation method is **early stopping**. When the capacity of a model is big enough, the model can eventually fit the training data perfectly. This is usually not desirable for most problems. We can use a validation set to determine when to stop the training before it fits the data perfectly. Usually we stop the training if the validation accuracy stops increasing for a certain number of weight updates, and save the model that gives the best validation accuracy. Even though validation accuracy is a commonly used metric in early stopping, other metrics can be used.

Dropout is an interesting regularisation method that draws inspiration from bagging in ensemble learning [111]. In bagging we use models with high representation capacity (high variance) and train each of them on a sample of the training data. Prediction is generated by averaging or voting of the outputs generated by the ensemble of models. The idea is that if the models in the ensemble make different types of mistakes, by combining them together the variance of the ensemble will be smaller than the individual models in the ensemble. However, bagging is not practical for large neural networks due to the high computational cost of training neural networks. Dropout can be seen as an extreme version of bagging where subnetworks in a large network are trained on different samples of the training data. During training, dropout sets a different portion of the units to zero for each iteration. Dropout can be applied to all types of neural networks. When applying dropout we usually have to use a large neural network and a large amount of labelled training data should be available. Dropout was shown to be less effective when training data is limited and it makes the training slower [111].

The best way to make a model with high representation capacity generalise is to train it on a big set of training data. However, labelled data is not always available and can be expensive to obtain. In this case, **data augmentation** can be used to artificially generate more training data. Data augmentation generates more data by modifying the original data in such a way that the classes of the data do not change. Data augmentation is effective and easy to implement for image recognition. Techniques such as rotation, stretching and mirroring can be applied to the original data to generate new images. Szegedy et al. [115] introduced a way to generate **adversarial examples** for image recognition problem. These adversarial examples were generated using gradient descent and backpropagation, but the input values were updated instead of the weights in order to increase the loss function. These generated images look just like the original images to a human, but a convolutional neural network will misclassify these adversarial examples. Training on this kind of adversarial examples was shown to improve the generalisation and stability of a neural network [37, 115].

Unsupervised pretraining was used to help solve the training problems of neural networks before ReLU became a popular activation function [5, 46, 92]. However, it was later discovered that unsupervised pretraining actually improves generalisation when labelled training data is limited [30]. When labelled data for a task is limited, supervised pretraining on a different but related task can also improve the generalisation of the original task [127].

2.9 Summary

In this chapter, we introduced what a neural network is and reviewed important components found in modern neural networks. They provide the essential background for understanding the remaining chapters of this thesis. We only consider FNN and

CNN architectures in this thesis. Because we only study classification problems, we use softmax as the output function and cross-entropy as the supervised loss function. Unless specified otherwise, we use ReLU as the activation function, the standard SGD as the optimisation algorithm and the initialisation method defined in He et al. [43]. We do not use any data augmentation technique in our experiments, since obtaining state-of-the-art results on benchmark datasets is not our goal. Having a simpler experimental setup makes interpreting the results easier.

3 Literature Review

3.1 Overview

This chapter reviews related works on three different strategies for using unlabelled data to improve the generalisation of neural networks. Section 3.2 reviews pretraining methods that learn reusable model parameters or feature representations. Section 3.3 reviews semi-supervised learning algorithms for neural networks. Some literature considers pretraining as a type of semi-supervised learning. We review them as two different strategies in this chapter for the following reasons. Pretraining typically involves at least two training phases, while modern semi-supervised algorithms for neural networks have only one phase. Semi-supervised algorithms reviewed in this chapter optimise a combined loss that has a supervised part and a part for the unlabelled data. Pretraining on the other hand optimises a single loss function in each training phase. The goal of pretraining in a classification task is typically to learn a “better” set of initial weights than random weights. Lastly, Section 3.4 reviews different active learning methods that strategically select unlabelled data to be labelled. Active learning is useful when resources are available to collect more labelled data. The three strategies reviewed in this chapter are not mutually exclusive to each other. In practice, one can experiment with combining all three strategies in order to make full use of the unlabelled data.

3.2 Pretraining

Pretraining is a general practice that trains a model on a related task, before reusing the learned weights or the latent feature representations on the original supervised learning task. If the related task is unsupervised in nature, it is referred to as *unsupervised pretraining*. If the related task is supervised in nature, but the supervision is derived directly from the unlabelled data, we call it *self-supervised pretraining*. If the related task is for a different domain or dataset, this is *transfer learning*. Table 3.1 shows a summary of related research on different types of pretraining. We will discuss these three types of pretraining in the following sections.

3.2.1 Unsupervised Pretraining

Unsupervised pretraining uses unsupervised models or generative models to learn latent features or model weights from unlabelled data. In early unsupervised pretraining methods, Restricted Boltzmann Machines and Autoencoders were used as layer-wise pretraining to initialise deep neural networks [5, 46, 92]. The learned layers were stacked together to form the initial weights of a deeper model. The model was then fine-tuned on the labelled data using supervised training. This two-phase learning procedure was done to solve the "vanishing gradient" problem in training deep neural networks, before piece-wise linear activation functions were widely adopted. It was later discovered that unsupervised pretraining was helpful in improving the generalisation performance on MNIST [71] when labelled training data were limited [30]. However, Dosovitskiy et al. [27] reported that the unsupervised pretraining using Autoencoders did not outperform random initial weights on harder datasets such as CIFAR [66].

Table 3.1: Summary of related research on pretraining.

	Unsupervised Pretraining	**Self-Supervised Pretraining**	**Transfer Learning**
Pretraining Task	Unsupervised learning	Supervised learning	Supervised/Unsupervised learning
Labelled Data	-	Created from original data	From a different dataset / not needed for unsupervised transfer learning
Key Research	Hinton et al. [46], Bengio et al. [5], Ranzato et al. [92], Erhan et al. [30], Radford et al. [91], Donahue et al. [26]	Gidaris et al. [34], Dosovitskiy et al. [27], Doersch et al. [25]	Yosinski et al. [127]

Recently, Generative Adversarial Networks (GANs) have also been used to learn reusable weights and feature representations [26, 91]. These methods train a GAN for image generation on unlabelled data, the weights or the latent features in the learned GAN can be used in a downstream task by fine tuning the weights with labelled data or applying a classifier on the features. However, Donahue et al. [26] reported that self-supervised pretraining outperformed the unsupervised pretraining (including methods using GAN) on datasets with higher-resolution images.

3.2.2 Self-Supervised Pretraining

Self-supervised pretraining applies supervised learning on a *pretext* task on the unlabelled data. A pretext task is a supervised task but the supervision (label information) is derived from the unlabelled data. For instance, a pretext task for images can be predicting the rotation applied to an image [34]. Dosovitskiy et al. [27] create multiple patches given an image using different transformation methods. A classifier is then trained to predict the transformation applied. Doersch et al. [25] extract a patch randomly from an image, they then extract multiple neighbouring but non-overlapping patches with tiny perturbations. A classifier is trained to predict the location of the neighbouring patches. Just like other pretraining methods, the learned weights and features from the pretext task can be reused or fine-tuned in the original task. The pretext task like prediction rotation can be trained with the original supervised task simultaneously on the same model without having to use a two-phase procedure.

3.2.3 Transfer Learning

When labelled data for a dataset are limited, pretraining on a different but related labelled dataset was found to improve the generalisation on the original dataset [127]. If the two datasets are similar to each other, then the weights learned on one dataset is highly transferable to the other dataset. For example, the weights of a model trained to recognise dogs are reusable to initialise a model that will be trained to recognise wolves. If two datasets are distant from each other, the weights or features learned from one dataset are less useful for the other dataset. For example, a model trained on images of whales is less transferable to the problem of recognising wolves. This is especially true for the top layers, because the top layers in a neural network tend to learn more specific and higher-level features for a dataset. The lower layers usually learn more lower-level features that are less specific to a dataset. Note that the pretraining task in transfer learning can be unsupervised learning as well. In this case, only unlabelled data from a different domain or dataset are required. Because transfer learning requires access to a different dataset or domain, this is a departure from the learning scenario studied in this thesis. In this thesis, we focus on the setting where the labelled data and the large pool of unlabelled data come from a single dataset and the same domain. We do not consider learning with multiple domains or different datasets. Therefore, we will not provide a detailed literature review on transfer learning and its applications in this thesis.

3.3 Semi-Supervised Learning

Semi-supervised learning (SSL) is different from supervised learning and unsupervised learning in that an algorithm is provided with both labelled data and unlabelled data. Semi-supervised learning can be *transductive* or *inductive* in nature [13]. In transductive learning, the goal is to make predictions on the unlabelled data given the labelled data. No explicit model or function is learned in transductive learning. Algorithms for transductive semi-supervised learning typically construct graphs based on similarity between data examples, and then propagate the label information from labelled data to unlabelled data. Contrary to transductive learning, an inductive learning algorithm learns an explicit model using both labelled and unlabelled data and performs predictions on future unseen data. In this thesis, we only focus on inductive semi-supervised learning, because our goal is to improve the generalisation of the learned neural network model by utilising unlabelled data. For anyone interested in transductive learning, Chapelle et al. [13] provide a detailed review on semi-supervised algorithms in the transductive setting.

3.3.1 Assumptions in Semi-Supervised Learning

The ultimate goal of semi-supervised learning, regardless if it is inductive or transductive, is to infer the posterior probability $P(y|x)$ given an example x. In order for semi-supervised learning to improve on the generalisation performance of supervised learning, it is necessary that the information about the marginal probability of an example $P(x)$ is useful for the inference of $P(y|x)$. If this condition is not met, the additional unlabelled data will not improve the generalisation of a classifier.

Chapelle et al. [13] formulated three common assumptions found in semi-supervised learning algorithms. We explain these three assumptions below.

The Smoothness Assumption

One of the commonly used assumptions in semi-supervised learning is the *smoothness assumption*. The smoothness assumption states that if two data points are close in the input space, the outputs of a model on the two data points should also be close. The smoothness assumption can be applied to both classification tasks and regression tasks. However, we only consider classification in this thesis. In classification tasks, we can interpret the smoothness assumption as if two examples are similar to each other then they are likely to share the same label.

The Cluster Assumption

The *cluster assumption* states that if two data points belong to the same cluster, they are also likely to belong to the same class. The cluster assumption is related to the smoothness assumption, because data points in the same cluster are likely to be similar to each other. The cluster assumption also implies that the target decision boundary should lie in the low-density regions [13]. If a decision boundary can exist in dense regions of the data, we no longer can assume that data points in the same cluster belong to the same class.

The Manifold Assumption

The *manifold assumption* is related to the cluster assumption, but is used for high-dimensional data. The datasets found in machine learning tasks can be of high dimensionality, such as images. The manifold assumption states that the original high-dimensional data lie on low-dimensional manifolds, and the data points that lie on the same manifold are likely to belong to the same class. The manifold assumption is useful whenever the curse of dimensionality comes into play. This affects the performance of algorithms that require distance computation, such as the graph-based methods used in the transductive setting and the methods that need density estimation. The manifold assumption suggests that dimensionality reduction techniques can be applied to obtain a lower-dimensional representation of the data before applying these semi-supervised algorithms.

3.3.2 Semi-Supervised Learning for Neural Networks

There is a rich body of literature on semi-supervised learning, it is not plausible to review all of them in this thesis. Instead, we will only review the semi-supervised algorithms specifically designed for neural networks in this section. The semi-supervised methods reviewed in this section are listed in Table 3.2 with their assumptions.

Table 3.2: Assumptions of semi-supervised methods in neural networks.

Methods	**Assumptions**		
	Smoothness	**Cluster**	**Manifold**
Pseudo-Label [73]		●	
Ladder Network [93]	●		
Γ-model [93]	●		
Π-model [69]	●		
Temporal Ensembling [69]	●		
Mean Teacher [116]	●		
Virtual Adversarial Training [85, 84]	●		
MixMatch [7]	●	●	
ReMixMatch [6]	●	●	
Deep Generative Model [59]		●	●

Pseudo-Label

Pseudo-Label [73] is a simple method that assigns "pseudo-labels" to unlabelled data using the model predictions, and then the model is trained on both the labelled data and the "pseudo-labelled" data. This process is repeated after each epoch or training step. Because the model predictions in the early stage of training are less reliable, the weights of the "pseudo-labelled" data are increased as the training goes on. This type of semi-supervised learning is also called *self-training*. Lee [73] showed that Pseudo-Label has an implicit effect of entropy regularisation. It attempts to minimise the entropy of the distribution of the predicted probabilities for unlabelled data. This indicates that Pseudo-Label is an application of the cluster assumption.

Ladder Network

As discussed in Section 3.2, pretraining of neural network layers using Denoising Autoencoders [120] has been shown to improve the generalisation performance. Inspired by this, Ladder Network [93] is proposed to combine Denoising Autoencoders with supervised training in a single phase. In other words, a separate pretraining phase is not needed before training on the labelled data. Ladder Network does this by combining the standard supervised loss with the reconstruction loss used in Denoising Autoencoders into a single loss function. The training process optimises the combined loss function on both the labelled and unlabelled data at the same time. A Ladder Network is comprised of two encoders and one decoder. One of the encoders is fed with clean data, the other is injected with Gaussian noise to the input of each of its layers. The decoder attempts to reconstruct the clean input of each layer. The discrepancy between the reconstructed input and the clean input is computed as the reconstruction loss. The final layer of the clean encoder can be used as input to a classifier for making predictions.

Ladder Network can be simplified by only computing the reconstruction loss on the last layer, instead of every layer of the network. This is referred to as the Γ-model [93]. The advantage of the Γ-model is that it can be applied to any feedforward network structure without having to implement the decoder. Consequently, the computation cost is also reduced. However, the generalisation performance was shown to be inferior than the full Ladder Network when the labelled dataset was small [93]. The difference in generalisation performance was reduced as the labelled dataset became larger.

Self-Ensembling

Laine and Aila [69] proposed the Π-model, that computes two different model outputs given a single input example by using stochastic data augmentation and dropout [111] during training, in addition to injecting Gaussian noise to the input. The Π-model then computes the difference between the two model outputs as the unsupervised loss. The unsupervised loss is minimised along with the supervised loss at the same time during training. The Π-model is similar to the Γ-model [93]. However, the Π-model does not have the corrupted encoder. It uses the same model to produce two different outputs from the same input by perturbing the input and using dropout.

Laine and Aila [69] also proposed Temporal Ensembling, a slight variation to the Π-model. In the Π-model, the two outputs are both computed from the same model during training. Temporal Ensembling keeps an exponential moving average of the model outputs across epochs as the "target output". The output of the current model given an input is then compared against this exponential moving average "target output". Laine and Aila [69] claimed this is more stable than the Π-model. The experimental results did show that Temporal Ensembling outperformed the Π-model by a small margin.

Mean Teacher

The problem with Temporal Ensemble [69] is that the storage or memory required to store the exponential moving average values grows linearly with the size of the training data (including unlabelled data). Mean Teacher [116] solves this problem by keeping an exponential moving average of the models (weights in a neural network) instead of model outputs. Now the memory requirements do not change with the size of the data. The unsupervised loss is then computed by calculating the difference between the current model output and the output of the exponential moving average model given an input example.

Virtual Adversarial Training

All the methods mentioned above perturb inputs by injecting isotropic Gaussian noise or applying random data augmentations. Virtual Adversarial Training [85, 84] attempts to inject tiny perturbations that are expected to affect the model output the most by using the idea of adversarial training [37]. Virtual Adversarial Training computes these tiny perturbations by applying gradient descent on the input space while holding the model constant. This optimisation process finds the small perturbation to the given input that changes the model output the most. The unsupervised loss computes the difference between the model output of the original input and the output of the input injected with the "adversarial perturbation". Just like the other methods mentioned above, this unsupervised loss is minimised along with the supervised loss at the same time during training.

MixMatch

Most of the methods above apply the smoothness assumption. They assume that small perturbations to the input should not change the model prediction. Pseudo-Label is an application of the cluster assumption. MixMatch [7] attempts to combine the smoothness assumption, the cluster assumption and weight decay. The smoothness assumption is applied by using consistency loss on perturbed inputs like most other methods above. For the cluster assumption, MixMatch uses a *sharpening function* to reduce the entropy of the prediction distribution given the average prediction over multiple augmentations when computing the pseudo labels. This encourages the model's output distribution to be low-entropy.

ReMixMatch

ReMixMatch [6] is a variant of MixMatch [7]. First, ReMixMatch attempts to align the prediction distribution on the unlabelled data to the distribution of the class labels of the labelled data. The second change is called augmentation anchoring, in which the predictions of weakly augmented unlabelled data are used as targets for the same data applied with stronger augmentations.

Deep Generative Model

Kingma et al. [59] proposed to use Variational Autoencoder [58] to learn a low-dimensional feature representation of the original data. The learned features can then be used as input to a classifier in supervised training. The features can also be used to improve the distance computation in some semi-supervised algorithms commonly found in transductive learning. This is an application of the Manifold Assumption. Kingma et al. [59] also proposed a probabilistic model that treats the labels of unlabelled data as a latent variable. It assumes the data is generated by the latent features as in standard Variational Autoencoders [58] and the latent class label. Lastly, Kingma et al. [59] showed that these two methods can be combined into a two-phase algorithm. In the first phase, the low-dimensional latent features are learned using Variational Autoencoder [58]. In the second phase, the probabilistic model is learned using the latent features instead of the original data.

3.4 Active Learning

Active learning is a subfield of machine learning that attempts to *efficiently* obtain more labelled training data by selecting unlabelled examples to be labelled by an oracle (for example, a human annotator). A classifier is then trained with the additional labelled data. The goal is to improve the generalisation of the classifier as much as possible given a certain budget of unlabelled examples to be queried.

3.4.1 Active Learning Settings

Depending on the learning scenarios, active learning has been studied under three general settings: membership query synthesis, stream-based selective sampling and pool-based active learning [103].

Membership Query Synthesis

Membership query synthesis generates new examples from the input space to be labelled [4, 56]. The advantage of membership query synthesis is that it can query examples from the entire input space. It can be used to potentially create training examples that are useful but rare in the natural world. However, the performance of membership query synthesis is dependent on the generator used. It is possible that a lot of the generated examples never occur in nature, even though they belong to the input space. For instance, the input space of a 256×256 image is enormous, but natural images are only a small subset of all the possible images. A bad image generator can generate images that cannot be recognised by human annotators. On the other hand, a generator that is trained to generate realistic examples might not be able to generate a diverse set of examples. A generator that always generates similar examples that are representative of the training data is not useful for active learning. If one can design a generator that generates *realistic*, *diverse* and *informative* examples, membership query synthesis can be useful and valuable in some applications where *edge cases* are rare or expensive to obtain. For instance, a learning-based autonomous driving system requires an enormous amount of training data, especially the edge cases. An edge case in this scenario usually means an environment that is unpredictable and difficult for the system to deal with. These edge cases can be rare and costly to obtain when they do occur. Therefore, collecting edge cases is a challenging problem. Developing an effective membership query synthesis system can help solve this problem.

Stream-Based Selective Sampling

Stream-based selective sampling assumes a stream of unlabelled data, for each incoming example in the stream it decides whether to query the example or discard it [18, 20]. Active learning methods belonging to this category differ in their *query strategy*. A query strategy defines a measure used to evaluate each unlabelled example and, a threshold for determining which examples to query based on the evaluation. Query strategies, especially the measures used to evaluate unlabelled examples, are discussed more in detail in Section 3.4.2.

The threshold used to determine if an example is to be queried can be a user provided hyperparameter or computed by the algorithm itself. The more naive and simpler approach is asking the user to set a threshold. This is a popular approach in active learning literature. The problem with this approach is that the threshold can be difficult or not intuitive to set. It can take some time and experience for users to learn how to set the threshold. Additionally, unlike pure supervised learning, we usually do not have a big validation set to tune the hyperparameters in active learning. Dasgupta et al. [21] is an example that derives its threshold in a principled way. Like Cohn et al. [18] it utilises the idea of *version space* [83] to find regions where two models of the same model class disagree the most. The threshold is determined by deriving the bound of the empirical error differences of the two models. In most active learning methods the threshold does not change for each example. Ienco et al. [51] uses a stochastic threshold by multiplying the user specified threshold by a random number drawn from a Normal distribution $N(0, 1)$ for each example. The introduced randomness allows the algorithm to occasionally select examples that are far from the decision boundary. This helps increase the diversity of the selected examples.

Stream-based selective sampling is suitable for data stream environment. A data stream is an infinite and continuous sequence of data examples. The speed of the incoming stream are different for different applications, and it can vary in different time periods. What makes data streams even more challenging is that the distribution of the data can change over time. Due to these challenges, researchers have proposed active learning methods specifically designed for data streams [50, 51, 64]. Ienco et al. [50] proposes a two-phase clustering-based active learning method: ACLStream. ACLStream clusters a batch of incoming data examples, and ranks the clusters based on the *homogeneity* of the predictions of the examples in each cluster. A cluster whose predictions are more balanced (less homogeneous) is ranked higher. The assumption is that a less homogeneous cluster covers a more difficult region of the data. Starting from the least homogeneous cluster, ACLStream ranks the examples in each cluster based on their uncertainty, and selects the top portion of the examples in each cluster until the budget is reached. Ienco et al. [51] proposes to only select examples from dense areas and it selects the most uncertain examples from these areas. When the distribution changes in the data stream, the performance of a classifier might degrade over time. Krawczyk et al. [64] attempts to solve this issue by utilising drift detection to select evolving examples to be labelled, so that the classifier can be retrained on the most up-to-date data.

Pool-Based Active Learning

Pool-based active learning assumes that there is a small set of labelled data and a large pool of unlabelled data. Pool-based active learning can be applied to a wide variety of applications, where an abundance of unlabelled data can be easily obtained. This is true for most machine vision tasks, speech recognition, text classification, etc. In stream-based active learning each example is evaluated sequentially and usually independently from other examples. The advantage is that the processing time and memory are small. However, this means that we have to decide whether to query an example without knowing if there will be more informative examples in the remaining stream. Consequently, the performance of stream-based active learning is dependent on the order in which the data examples are evaluated. Pool-based active learning avoids this problem by evaluating a large pool of unlabelled data at once, and ranks all the examples according to a certain measure. It then chooses the top example or a batch of top examples to be labelled. It is called **batch-mode active learning** when an algorithm selects a batch of examples at once to be labelled, instead of doing it one at a time. Pool-based active learning is the most common active learning setting in the literature. We review pool-based active learning methods in Section 3.4.2 when we discuss different query strategies.

3.4.2 Query Strategies

Different active learning methods adopt different measures for evaluating unlabelled data. These measures typically consider one of (or a combination of) the following three criteria: *informativeness*, *representativeness* and *diversity*. A visualisation of the taxonomy of active learning query strategies is shown in Figure 3.1.

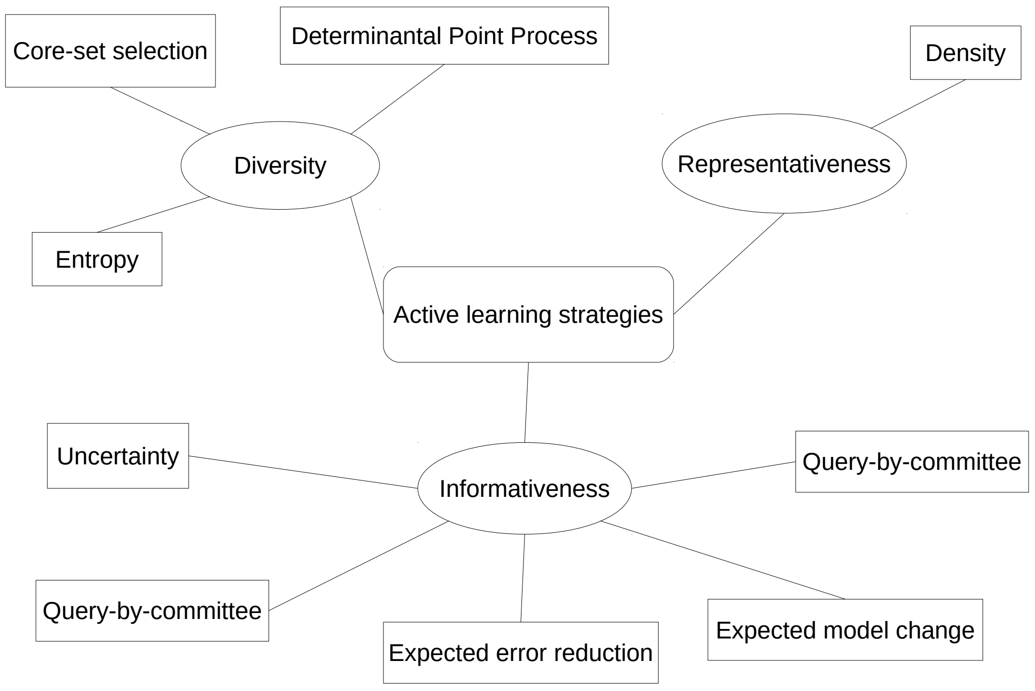


Figure 3.1: Taxonomy of active learning query strategies.

A general strategy can have different sub-strategies. Each active learning method has its own implementation of a sub-strategy or a combination of multiple strategies. We will review some important works for each of the query strategies in the following sections.

Informativeness

Many informativeness-based active learning methods attempt to select data examples that are most likely to reduce the **uncertainty** of the model. These methods assume that the example the current model is most uncertain about is most useful to improve the model's generalisation performance. The simplest uncertainty-based method selects the examples the model is least confident about [19, 31, 51, 76, 103, 104]. Confidence is formally defined as follows:

$$Confidence(x_j) = maxP(\hat{y}_c|x_j, \theta) \tag{3.1}$$

It is the maximum posterior probability of any class $\hat{y}_c$, given an unlabelled example x_j and the model θ. An alternative method for measuring uncertainty is computing the margin between the posterior probabilities of the two most likely classes [11, 31, 51]:

$$Margin(x_j) = P(\hat{y}_{c1}|x_j, \theta) - P(\hat{y}_{c2}|x_j, \theta) \tag{3.2}$$

Margin-based methods select examples whose margins are the smallest. Confidence only considers the posterior probability of the most likely class, while margin takes into account the two most likely classes. Ienco et al. [51] experimented with both margin-based and confidence-based uncertainty in their proposed method, and found that margin outperformed confidence in most cases. Entropy [107] can also be used to measure uncertainty [10, 48, 103, 121]:

$$Entropy(x_j) = -\sum_{c=1}^{C} P(\hat{y}_c|x_j, \theta) \log P(\hat{y}_c|x_j, \theta) \tag{3.3}$$

It measures the entropy of the distribution of the entire posterior probabilities. However, it is not difficult to conceive a scenario where entropy would not work as a definition of uncertainty. For instance, suppose we have a classification task with 6 classes, and two posterior probability distributions for x_1 and x_2 respectively:

$$P(\hat{y}_{\forall c \in C}|x_1, \theta) = [0.5, 0.45, 0.0125, 0.0125, 0.0125, 0.0125]$$
$$P(\hat{y}_{\forall c \in C}|x_2, \theta) = [0.5, 0.1, 0.1, 0.1, 0.1, 0.1]$$

The entropy values for the two distributions are:

$$Entropy(x_1) = 0.925$$
$$Entropy(x_2) = 1.498$$

According to the entropy values, the model is more uncertain about example x_2. However, one can argue that the model is actually more uncertain about example x_1, since it has a difficult time determining between the two most likely classes. A margin-based active learning method would choose x_1 over x_2 in this case.

Query-by-committee [106] is another way to identify uncertain or informative examples. Generally speaking, query-by-committee methods define uncertainty as the degree of disagreement among the models in an ensemble. Dagan and Engelson [20] propose to measure the disagreement using *vote entropy*:

$$Vote\_entropy(x_j) = -\sum_{c=1}^{C} \frac{V(\hat{y}_{c,x_j})}{K} \log \frac{V(\hat{y}_{c,x_j})}{K} \tag{3.4}$$

where $V(\hat{y}_{c,x_j})$ is the number of votes for a specific class $\hat{y}_c$ for example x_j, and K is the number of committee members. Kullback Leibler (KL) divergence [79] has been used to measure disagreement as well:

$$Average\_KL\_divergence(x_j) = \frac{1}{K} \sum_{k=1}^{K} D(P_{\theta_k} \| P_{\kappa}) \tag{3.5}$$

where:

$$D(P_{\theta_k} \| P_{\kappa}) = \sum_{c=1}^{C} P(\hat{y}_c | x_j, \theta_k) log \frac{P(\hat{y}_c | x_j, \theta_k)}{P(\hat{y}_c | x_j, \kappa)} \tag{3.6}$$

where

$$P(\hat{y}_c | x_j, \kappa) = \frac{1}{K} \sum_{k=1}^{K} P(\hat{y}_c | x_j, \theta_k) \tag{3.7}$$

$P(\hat{y}_c | x_j, \kappa)$ is the average posterior distribution of the committee for example x_j. $D(P_{\theta_k} \| P_{\kappa})$ is the KL divergence between the distributions of a committee member and the committee average. Intuitively, average KL divergence measures the average difference in posterior distributions between committee members and the committee average. The examples with higher average KL divergence values are considered to be more uncertain among committee members and therefore are more informative. Query-by-committee methods can be computationally expensive when we use a large committee and, each committee member takes a long time to train and takes up a lot of memory space. Because each member in the committee can be trained independently from each other, distributed training on many computers in parallel can be used to significantly reduce the running time. However, this means it is not a plausible method when computing resources are limited.

Another way to identify informative examples is by computing the **expected model change** if we add an instance to the labelled data pool. Settles et al. [105] and Käding et al. [54] define expected model change as the **expected gradient**

length:

$$Expected\_gradient\_length(x_j) = \sum_{c=1}^{C} P(\hat{y}_c|x_j,\theta)\|\Delta J(L^{+<x_j,y_c>},\theta)\| \qquad (3.8)$$

where J is the loss function, and $L^{+<x_j,y_c>}$ is the labelled data set L with the new training example $< x_j, y_c >$. We do not know the true label for x_j, we simply assign a label to x_j as if it were ground truth and add the training example to the labelled data. The model is trained with the additional example. We then compute the magnitude of the gradient, $\|\Delta J(L^{+<x_j,y_c>},\theta)\|$. This is repeated for each class, and then the expected gradient length is computed by multiplying the magnitude of gradient with the posterior probability, $P(\hat{y}_c|x_j,\theta)$. This method is also computationally expensive, because we have to compute the gradient for each available class and we have to do it for all of the unlabelled examples. If the model has already converged before on training data L, the gradient $\Delta J(L,\theta)$ is 0. Therefore, we can reduce the computational cost by computing gradient only on $< x_j, y_c >$ instead of the entire training set $L^{+<x_j,y_c>}$.

Expected error reduction based query strategy is similar to expected model change, but instead of selecting examples that minimise the expected model change it chooses examples that minimise the expected error of the model. This query strategy is computationally expensive. Roy and McCallum [94] proposed to use Monte Carlo sampling to estimate the expected error of the model on a labelled validation set in order to reduce the computational cost. This query strategy can be applied to different types of classifiers as long as the classifier provides a way to estimate the posterior label probabilities. However, the estimated posterior probabilities can be inaccurate especially when the labelled training set is small [12]. Furthermore, in order to obtain a reliable estimation of the expected error reduction, one would need a big validation set that is representative of the data. This is not always feasible in practice. We usually use active learning exactly because there is a lack of labelled data. Krempl et al. [65] proposed a probabilistic approach to estimate the expected gain in performance without using a validation set by modelling the label and posterior probability as random variables. However, the proposed method only works for datasets with binary classes. Kottke et al. [61] extended this approach to multi-class datasets.

Representativeness

Representativeness-based active learning methods query examples that are most representative of the data. This is usually done by selecting examples from the **dense** regions of the data. One of the most common ways to define density for an example is to compute the average similarity between this example and all the other examples [103]. Only the examples with high average similarity scores are queried. Ebert et al. [29] proposed to build a k-nearest neighbour graph weighted using a Gaussian kernel. It then selects the examples that have many edges with high weights. This method assumes that the examples that have many edges with high weights are most representative of the data. It avoids sampling outliers and noises points in the data.

Most active learning methods that use representativeness in their query strategy usually combine representativeness with informativeness. They choose examples that are informative and also representative of the data. McCallum and Nigam [80] combined density with query-by-committee to query examples in dense regions of the data and have the highest disagreement among members of the ensemble. Krempl et al. [65] and Kottke et al. [61] compute density weighted expected gain in performance when querying examples. Ienco et al. [51] used a two stage active learning method to select examples with high uncertainty scores only from dense regions of the data. Huang et al. [49] proposed a method specifically for support vector machines that combines uncertainty and representativeness.

Diversity

Diversity is another active learning strategy used to query examples. The goal is to make sure the pool of labelled examples are as diverse as possible. Sener and Savarese [101] consider the active learning problem as a *core-set selection* problem. In core-set selection, a subset of the data is selected such that a model learned over this subset of data is just as good as a model trained on the whole data. Intuitively, it tries to choose a set of cluster centers such that the greatest distance between an example and its nearest center is minimised. However, this problem is NP-Hard. Sener and Savarese [101] proposed a greedy algorithm to greedily select an example with the highest minimal distance to any other example. This proposed method does not use labels at all, therefore it can be used to initialise the labelled training set. Additionally, they also proposed a more robust version of the algorithm against outliers.

Wei et al. [122] formulated the active learning problem as an optimisation problem for *submodular functions* using Naive Bayes and Nearest Neighbour classifiers as examples. A submodular function is a set function where the growth of its value decreases as the size of the set increases. Submodular functions naturally model diminishing returns. Entropy can be shown to be a submodular function. Maximising the entropy promotes diversity. Unfortunately, maximisation of a submodular function is NP-Hard. However, a greedy algorithm with almost linear complexity can be used to approximate the results [81]. Wei et al. [122] proposed to select a candidate set first using uncertainty based active learning, then select a diverse subset from the candidate set. This approach can reduce the problem of selecting similar examples in the same batch.

An approach related to submodular functions is *Determinantal Point Processes (DPP)* [9, 68]. DPP is a probabilistic model that naturally models negative correlations, and offers exact algorithms for sampling, marginalisation, conditioning, and other inference tasks. A DPP models the probability distribution of all the possible subsets of a ground set. In active learning, we are usually interested in a specific type of DPP called k-DPP, where all the subsets must have the same cardinality of k. Sampling of k-DPP over the unlabelled data gives us a diverse subset of k examples from the unlabelled data. However, the exact sampling can be inefficient due to the computational complexity of eigendecomposition. Moreover, the size of the sample must be smaller than the rank of the kernel matrix used. In order to accelerate the computation, approximate algorithms have to be used [9, 68].

3.5 Summary

This chapter reviews research on improving the generalisation of neural networks using three different strategies: pretraining, semi-supervised learning and active learning. Pretraining trains a model on a different task and then reuses the learned weights or features on the original task. Semi-supervised learning regularises the model by adding an unsupervised loss to the supervised loss during training. Active learning strategically selects unlabelled data to be labelled and added to the labelled data set.

Most of the early works on pretraining were aimed to improve the stability of training by using unsupervised pretraining. More recent works on pretraining pretrains a model using a supervised pretext task whose supervision is automatically created from the data. However, a pretext task is specific to a task or even a dataset. A pretext task that works on one dataset or one type of task may not be suitable for a different dataset or task. Most of the research on self-supervised pretraining focuses on image and text data. In Chapter 4, we propose a pretraining method for tabular data.

All of the literature on semi-supervised learning reviewed in this chapter assumes that the unlabelled data come from the same distribution as the labelled data. This assumption may not hold in practice. Chapter 5 empirically investigates what would happen if we break this assumption.

There has been a rich body of literature on active learning. However, most of it is not specifically designed for neural networks. Additionally, almost all of the literature on active learning focuses on improving the generalisation of a classifier when labelled dataset is small. Not enough attention is paid to improving a classifier when there is already a large labelled dataset. In Chapter 6, we propose an uncertainty-based active learning method specifically for neural networks that is more competitive when the initial labelled dataset is large.

4 Supervised Pretraining with Unlabelled Data

4.1 Overview

Apart from novel network architectures, the success of neural networks is a result of many advancements in the basic components such as activation functions and initialisation. During the training process deep neural networks are sensitive to the initial weights. A poor choice of initial weights can result in slow training speed, "vanishing gradient", "dead neurons" or even numerical problems. A few initialisation methods have been proposed to stabilise the training process [35, 43, 82]. However, they do not aim to improve the generalisation of neural networks. It is possible that even though these methods can successfully improve the stability and speed of training, they are not necessarily effective for improving the generalisation of neural networks. Initialisation of neural networks is a difficult problem to study due to the complex nature of the cost surface and the optimisation dynamics. However, more research needs to be devoted to exploring the connection between the initialisation and the generalisation of neural networks. We argue that the initialisation affects the generalisation of neural networks. Intuitively speaking, the initial weights of a neural network can be thought of as the prior of a model. If this intuition is correct, the choice of initial weights would have a bigger effect on the learned model after training, when there is a lack of labelled training data. Hence, studying the effect of initialisation on generalisation is valuable for domains where labelled data are either costly or difficult to collect.

This chapter answers the following two **research questions**:

1. How do the initial weights affect the generalisation performance of a neural network?

2. How can we create a supervised pretraining task from the unlabelled data to pretrain a model, and then reuse the learned weights as initial weights in order to improve the generalisation on the original supervised task?

The **main contributions** of this chapter are listed below:

- We firstly demonstrate that initialisation affects the generalisation performance of neural networks. A poorly initialised model can lead to a lower test accuracy.

- We propose a supervised pretraining method that improves the generalisation of neural networks on tabular data when labelled data is limited, by taking advantage of unlabelled data.

The proposed method trains a neural network to distinguish real data from shuffled data. The learned weights are reused as initial weights when training on the labelled training data. The intuition is that during the pretraining, the model has to learn joint distributions of the real data in order to identify real data points from shuffled ones. And the learned weights might be a better prior than standard initialisation methods that sample values from normal or uniform distributions. The improvement in generalisation by using the proposed method is shown to be more obvious when there is a lack of labelled data and the class separation is small. Experimental results also suggest that the proposed pretraining is most effective when the data has high dimensionality and contains noisy features. We only consider binary classification problems in this work. We measure generalisation using test accuracy in all our experiments.

The rest of the chapter is organised as follows. Section 4.2 introduces some background information about research on the initialisation of neural networks. In Section 4.3, we demonstrate the initialisation of neural networks affects generalisation. In Section 4.4, we propose a supervised pretraining method to improve the generalisation of FNNs on binary classification problems. We conducted experiments on both UCI datasets and synthetic datasets to evaluate the proposed method in Section 4.5 and Section 4.6. In Section 4.7, we discuss some issues with the proposed pretraining method. Finally, we summarise this chapter in Section 4.8.

4.2 Background

Glorot and Bengio [35] derived a way to initialise weights depending on the number of input units and output units of a layer. They derived this method by assuming only linear activation functions are used, and attempting to initialise the weights in such a way that the variances of activation values and gradient across all layers are the same at the beginning of training. Despite the unrealistic assumption of a linear activation function, this initialisation works well in practice. Using the same idea, He et al. [43] derived an initialisation method specifically for the ReLU activation function depending on the number of input units of a layer. It draws weight values from a normal distribution with mean value of 0 and the standard deviation $sqrt(2/fan\_in)$, where fan_in is the number of input units of a layer. We call this initialisation method *he_normal*. They showed that *he_normal* performed well even for really deep networks using ReLU while the Glorot and Bengio method [35] failed.

Unsupervised methods such as restricted Boltzmann machines and autoencoders were used as layer-wise pretraining to initialise deep neural networks [5, 46, 92]. This was done to solve the "vanishing gradient" problem in training deep neural networks, before piece-wise linear activation functions were widely adopted. It was later discovered that unsupervised pretraining has a regularisation effect on MNIST [71] when labelled training data is limited [30]. However, it has been shown that the unsupervised pretraining is not as effective on harder datasets with high-resolution images [26, 27].

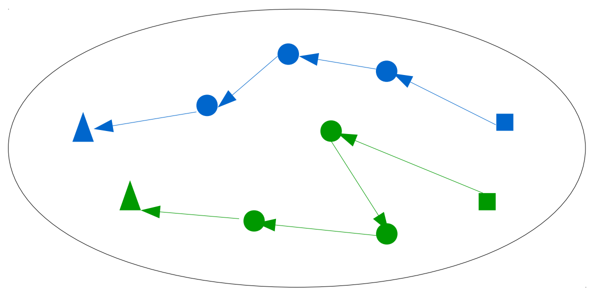


Figure 4.1: Machine learning represented as a search process. The area inside the elliptical boundary is the representation capacity of the model chosen. The circular dots are different model (parameter) states a learning algorithm has visited. The square is the initial state of the model while the triangle is the final state of the model.

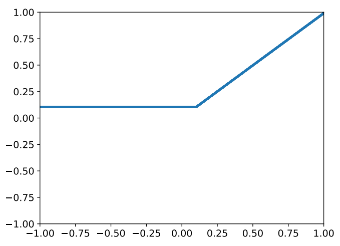
4.3 Initialisation Affects Generalisation

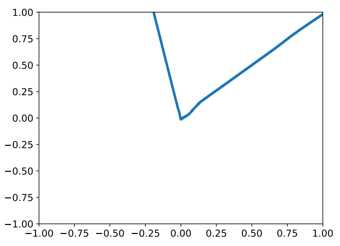
Most machine learning problems can be reduced to search problems, especially when an iterative learning algorithm is used to find the final model. A search problem usually consists of a search space, an initial state, a target state and a search algorithm. Figure 4.1 describes training neural networks as a search process. The area inside the ellipse represents the representation capacity of a neural network. The larger the neural network, the bigger the representation capacity and the larger the search space. The square represents the initial state of the model and the triangle is the final state of the model. The circular dots are the model states the learning algorithm has visited. The model search space can be discrete or continuous. However, the search space in training neural networks is usually continuous. Additionally, the search space or representation capacity of a neural network is not completely independent of the search process. The bigger and more complex a neural network is, the more difficult it is to train. So the representation capacity can potentially affect where the model will end up while holding the initial state and learning algorithm constant.

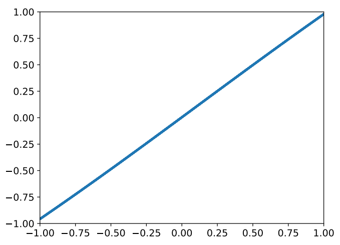
Unlike many search problems where the target state is known, we usually do not know the target state in machine learning. In the case of neural networks, we cannot analytically find the optimal model state due to the non-convex loss surface. So iterative optimisation algorithms are used in training neural networks. The most common learning algorithms in training neural networks are stochastic gradient descent and its variants. The first order derivatives of the model parameters are used as a heuristic for determining the next step. Obviously, different learning algorithms can result in different final models even if both the search space and initial state are kept the same.

Another factor that affects the final state of the search space is the initial state of the model. Different initial weights can lead a neural network to converging to different states, as illustrated in Figure 4.1. We argue that initialisation affects the generalisation of neural networks, especially when the training data is limited. The initial weights of a neural network can be seen as the prior of the model. The choice of the initial weights does not affect the generalisation much if there is plenty of training data, as long as the initial weights do not lead to training problems. However, when there is a lack of training data, the prior will have a larger effect on the final state of the model. Note that the size of the training data required is relative to the difficulty of the dataset and the classifier used. A more difficult dataset usually requires more training data to achieve a satisfactory performance. A classifier that tends to overfit also requires more training data. Figure 4.2 shows the plots of the functions learned by training four fully-connected networks to fit the same two data points but with different initial weights and activation functions. The two data points were (0.1, 0.1) and (0.5, 0.5). All four of the models had 128 units in each of the first two layers and one output unit. The initial weights were drawn from truncated normal distribution with mean of 0 and different standard deviation values (0.004 and 0.4). Any value that was more than two standard deviations away from the mean was discarded. All biases were set to 0. We used both rectified linear unit (ReLU) and hyperbolic tangent (tanh) as activation functions. We trained the model until the training loss became 0 and plotted the model. It can be seen that when the standard deviation of the initial weights is small, the learned function is relatively simple. When the standard deviation of the initial weights is big, the final model is more "complex" and less likely to generalise well on unseen data.

Similar experiments cannot be applied to real datasets and larger networks, because initial weights drawn from a normal distribution with a large standard deviation leads to various training problems such as "dead neurons" and exploded activation values. Inspired by transfer learning [127], we decided to learn inappropriate initial weights for the original classification task by pretraining a model on randomly shuffled data. More specifically, we shuffled the attribute values of the original data and randomly attached labels to the shuffled data according to the


(a) Mean 0 and sd 0.004 (ReLU)


(b) Mean 0 and sd 0.4 (ReLU)


(c) Mean 0 and sd 0.004 (tanh)

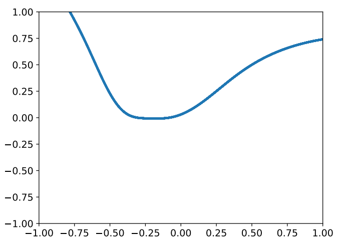
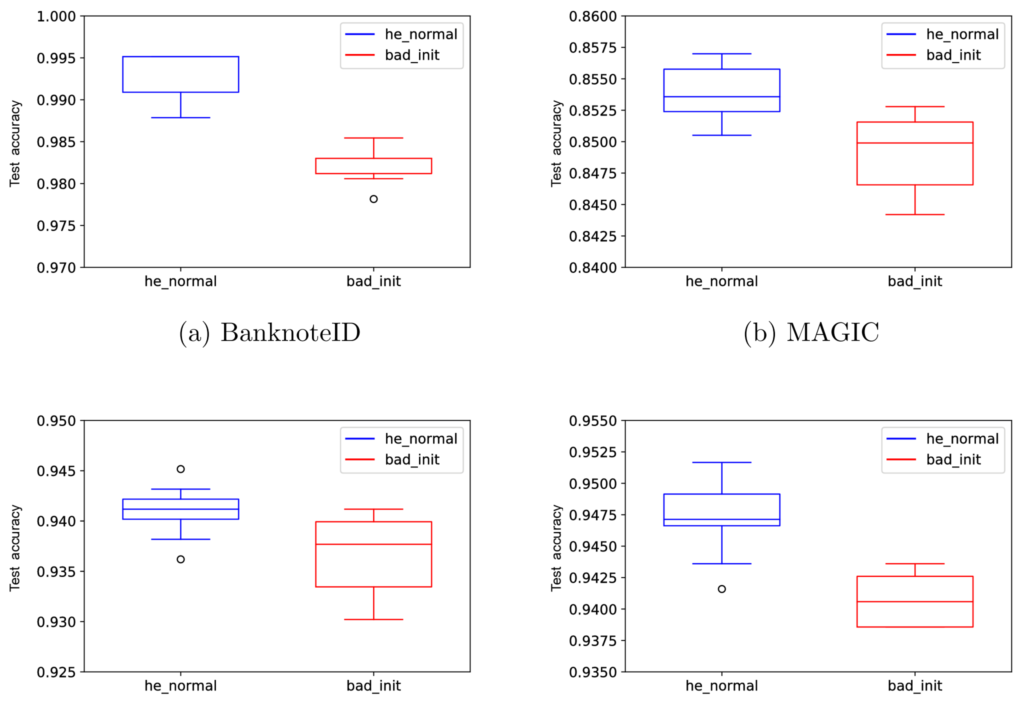

(d) Mean 0 and sd 0.4 (tanh)

Figure 4.2: Plots of the functions learned by training a three-layer fully-connected neural network to fit two data points (0.1, 0.1) and (0.5, 0.5), with different initial weights and activation functions. The activation functions used were ReLU and tanh. The initial weights were drawn from normal distributions with different mean and standard deviation values.

uniform distribution. We obtained a set of weights by training a model on this shuffled data. We expected this set of weights to be a poor prior for the original classification problem and thus would lead to lower test accuracy. When training a model on the shuffled data, we used *he_normal* as the initialisation method. These learned weights were then used as initial weights when training on the original data. We then compared the model initialised with these transferred weights against the model initialised using *he_normal* [43].

We carried out the experiments on four different datasets: BanknoteID, MAGIC, Waveform and Waveform_noise. Details of these datasets can be found in Section 4.5. We used a five-layer fully connected network for all the experiments. Each layer has 1024 units except the last layer. The last layer is a softmax layer. The architecture is reported in Table 4.3. The activation function used was ReLU. We used the full-batch gradient descent with a constant learning rate of 0.01. We chose the number of epochs to make sure both models had converged. The number of epochs we used for each dataset can be found in the second column of Table 4.4. The training on the shuffled data was run 10000 epochs for all the cases. We ran each experiment 10 times. Figure 4.3 shows the distribution of test accuracies obtained using the transferred initial weights and *he_normal* on four datasets. As can be seen, in all four cases, the model initialised with the transferred weights performed worse.



(a) BanknoteID (b) MAGIC

(c) Waveform (d) Waveform_noise

Figure 4.3: Comparison of the model initialised with transferred weights (red) against the model initialised with *he_normal* (blue) in terms of test accuracy. We ran each case 10 times.

In this section, we demonstrated that a poorly initialised model does indeed lead to lower test accuracy. In the next section, we propose a supervised pretraining method that improves generalisation, especially when labelled training data is limited.

4.4 Supervised Pretraining

We propose a supervised pretraining method that takes advantage of unlabelled data to improve the generalisation of the original classification problem. The underlying assumption of this method is that the weights learned during the supervised pretraining are a good prior for the original classification problem. This can happen if the pretrained model contains information about the joint distributions of the attributes in the original data.

The pretraining phase is basically a binary classification that attempts to identify real data from shuffled data. Shuffled data does not provide us with real patterns but instead are considered as noise. The learned weights are then reused in the original classification problem.

Table 4.1: An example of how the pretraining data is generated. Table (a) is the original unlabelled data and Table (b) is the labelled training data. Table (c) is the shuffled unlabelled data. Note that no data point in the shuffled data is from the unlabelled data. Table (d) is the pretraining data. The pretraining data is created by stacking the original unlabelled data and shuffled data, and labelling them as 1 and 0 respectively.

(a) unlabelled data

Attr 1	Attr 2	Attr 3
1	1	1
2	2	2
3	3	3

(b) Labelled data

Attr 1	Attr 2	Attr 3	Label
4	4	4	1
5	5	5	0

(c) Shuffled data

Attr 1	Attr 2	Attr 3
1	2	3
2	3	1
3	1	2

(d) Pretraining data

Attr 1	Attr 2	Attr 3	Label
1	1	1	1
2	2	2	1
3	3	3	1
1	2	3	0
2	3	1	0
3	1	2	0

We discuss the supervised pretraining method in detail here.

1. We start by randomly shuffling the attribute values of the unlabelled data across rows. Each attribute is shuffled independently from other attributes. This is to break the joint distributions of the original data, but to keep the distributions of individual attributes unchanged. Table 4.1(a) shows an example of unlabelled data and Table 4.1(c) shows an example of shuffled data. Note that the shuffling is random, so there can be many different examples of shuffled data generated from the same unlabelled data. In our implementation,

we ensured that there was no original real data points in the shuffled data due to chance collision. Any data points in the shuffled data that collided with the original data were replaced by new shuffled data points. This was done to avoid confusing the model during the pretraining process. Note that labelled training data can be added to the unlabelled data after taking out the labels, in order to create a larger set of unlabelled data. A shuffled dataset can then be created from this enlarged unlabelled dataset.

2. The unlabelled data and shuffled data are stacked together. Then all the real data points are assigned one label and all the shuffled data points are assigned a different label. This is the pretraining data used in our method. The specific values being used as labels do not matter, as long as all the real data points are assigned the same label and all the shuffled data points are assigned a different label. This is because one hot encoding is used to encode the labels in our models. In our implementation, the real data is labelled as 1 and the shuffled data is labelled as 0. The pretraining data always has a balanced class distribution. Table 4.1(d) shows an example of pretraining data.

3. The pretraining data is then split into a training set and a validation set using stratified sampling. In our implementation, 70% of the pretraining data was used as training data and the rest was used as a validation set.

4. During the pretraining, a neural network model is trained on the pretraining data. The validation set is used to stop the pretraining early. For the pretraining, the model was initialised using *he_normal* in our implementation.

5. Finally, the weights learned in the pretraining are reused as initial weights when training a model on the labelled training data shown in Table 4.1(b). This is sometimes called transfer learning, where weights learned in one problem are transferred to another problem [127]. In our implementation, all the layers except the last one were transferred from the pretrained model. The last layer was replaced with weights initialised using *he_normal*. We did not freeze any layers during training.

Note that the negative instances in the pretraining data are generated by shuffling the attribute values of the original data. So the distributions of the attributes in the shuffled data are exactly the same as the original data, but the joint distributions of the attributes are different. By training the model to distinguish real data from the shuffled data, we hypothesise that the model has to learn the joint distributions between attributes. We also postulate that if the weights learned during pretraining are a better prior than commonly used initialisation methods such as *he_normal*, then they should lead to better generalisation.

4.5 Experiments on UCI datasets

We conducted experiments on four UCI datasets to evaluate the proposed supervised pretraining against the *he_normal* [43] initialisation method. The goal of the experiments is to test if the proposed supervised pretraining leads to higher test accuracy compared to the *he_normal* initialisation. All the datasets were obtained from the UCI repository [24]. In our experiments, we simulate a learning environment where labelled training data is limited but there is plenty of unlabelled data. The source code and data can be found on GitHub: github.com/superRookie007/supervised_pretraining.

Table 4.2: Data information. Note that the second column reports the numbers of the labelled training examples, not the sizes of the full datasets.

Dataset	# Labelled Examples	Class1	Class2	# Attributes
BanknoteID	288	55.5%	44.5%	4
BanknoteID_small	10	55.5%	44.5%	4
MAGIC	3995	64.8%	35.2%	10
MAGIC_small	10	64.8%	35.2%	10
Waveform	702	49.3%	50.7%	21
Waveform_small	10	49.3%	50.7%	21
Waveform_noise	695	50.0%	50.0%	40
Waveform_noise_small	10	50.0%	50.0%	40

Datasets

BanknoteID is a dataset for identifying genuine banknotes from the forged banknotes. It has four continuous variables and 1372 data points in total. MAGIC is a simulated dataset for discovering primary gammas from the hadronic showers initiated by cosmic rays. It has 10 continuous variables and 19020 data points in total. Both Waveform and Waveform_noise are datasets for identifying different types of waves. Waveform has 20 attributes while Waveform_noise has 19 additional noisy variables drawn from a standard normal distribution. Both of them have 5000 examples in total. The original datasets of Waveform and Waveform_noise have three classes. We extracted two classes (class 1 and class 2) from the original datasets and treated them as binary classification problems. So the Waveform and Waveform_noise datasets in our experiments have 3343 and 3308 instances in total respectively.

Preprocessing

Thirty percent (30%) of the data was used as the test set. The remaining data was split into labelled training data (21%) and unlabelled data (49%). Additionally, we created a smaller training set with only 10 instances for each of the four datasets by sampling the labelled training data. When sampling the datasets, we used stratified sampling. Table 4.2 shows some data characteristics of the training sets. When creating the pretraining data for our experiments, we combined the training data (ignoring the labels) and unlabelled data, and then applied the method described earlier in Section 4.4 to this combined data. This gave us an even bigger pretraining dataset. The pretraining data was scaled to a range [0, 1], then the same scaling parameters were applied to the test set.

Table 4.3: The FNN architecture used in the experiments.

Layer	**Parameters**
Dense	input dimension $\rightarrow$ 1024, ReLU
Dense	1024 $\rightarrow$ 1024, ReLU
Dense	1024 $\rightarrow$ 1024, ReLU
Dense	1024 $\rightarrow$ 1024, ReLU
Softmax	1024 $\rightarrow$ 2

Setup

We used a five-layer fully connected network for all the experiments. The architecture used is reported in Table 4.3. Each layer has 1024 units except the last layer. The last layer is a softmax layer. The activation function used was ReLU. We used the standard full-batch gradient descent with a constant learning rate. Because the learning rate can potentially affect the generalisation of neural networks, we used the same constant learning rate of 0.01 for all the experiments. We chose the number of epochs to make sure both models had converged. We consider the model has converged if the training loss stops decreasing and the training accuracy reaches 100%. In practice, a validation set is usually used to terminate the training early to avoid overfitting. However, we trained the models until convergence because we wanted a fair stopping criterion for both cases, and we wanted to eliminate the generalisation effect and uncertainty of early stopping in the results. The number of epochs we used for each training set can be found in the second column of Table 4.4. Furthermore, the pretraining data was split into a training set (70%) and a validation set (30%). The pretraining stopped if the validation accuracy stopped increasing for the subsequent 100 epochs, and the model with the best validation accuracy was saved. We did not fine tune which layers to reuse or freeze the reused weights for a few epochs before updating them. We always transferred all the weights except the last layer and did not freeze any layer during training. The same experiment was run 10 times on each dataset. The experiments were all implemented using Keras [15] with the Tensorflow [1] backend.

Results

The metric we used to measure generalisation was accuracy on the test set. We have evaluated all the models with F1-Score, AUC-ROC and Cohen's Kappa [17], but they always moved in line with the test accuracy in this experiment. They did not add interesting information to the results, so they are not reported here. However, the results on these measures can be found in Appendix A. The experimental results on test accuracy are shown in Table 4.4. The higher mean test accuracy is shown in bold, but the difference is not necessarily statistically significant. As expected, the test accuracy is generally higher when the training set is larger. And the standard deviation of the test accuracy tends to be larger when limited training data is available. The base models tend to have much lower training loss while the pretrained models tend to have lower test loss. When there is not a lack of labelled training data, there is no clear improvement on generalisation using the supervised pretraining. On BanknoteID and Waveform_noise, the pretraining actually hurt the generalisation. However, when the training set is small, only 10 instances in this case, the mean test accuracy for BanknoteID and Waveform_noise of the pretrained

Table 4.4: Comparison of the model initialised using the supervised pretraining method (Pretrained) against the model initialised with *he_normal* (Base). We ran each case 10 times. The training accuracy reached 100% for all the runs.

Data	Epochs	Method	Train loss	Test loss	Test acc.
BanknoteID	10000	**Base**	9.78E-04 ±2.57E-05	5.79E-02 ±1.38E-03	**99.32%** ±0.28%
		Pretrained	1.04E-02 ±2.99E-04	4.30E-02 ±5.53E-03	98.74% ±0.39%
BanknoteID_small	5000	Base	1.93E-04 ±5.40E-06	9.02E-01 ±3.16E-02	82.18% ±0.80%
		Pretrained	1.96E-02 ±7.88E-04	6.13E-01 ±2.67E-02	**82.45%** ±0.81%
MAGIC	100000	Base	4.15E-03 ±4.88E-04	9.15E-01 ±1.71E-02	85.42% ±0.23%
		Pretrained	8.59E-03 ±3.21E-03	7.12E-01 ±2.54E-02	**85.69%** ±0.27%
MAGIC_small	5000	Base	2.37E-04 ±3.93E-06	1.22E+00 ±5.87E-02	78.12% ±0.20%
		Pretrained	2.04E-02 ±7.59E-04	7.20E-01 ±3.54E-02	**78.62%** ±0.51%
Waveform	20000	Base	5.71E-04 ±3.82E-05	3.02E-01 ±6.99E-03	94.10% ±0.25%
		Pretrained	9.89E-03 ±6.39E-03	2.36E-01 ±1.24E-02	**94.38%** ±0.25%
Waveform_small	10000	Base	8.97E-02 ±2.83E-01	4.06E-01 ±3.02E-02	88.48% ±0.58%
		Pretrained	1.02E-02 ±2.20E-04	4.13E-01 ±2.96E-02	**89.10%** ±0.67%
Waveform_noise	20000	**Base**	3.13E-04 ±1.70E-05	3.15E-01 ±9.78E-03	**94.72%** ±0.30%
		Pretrained	1.38E-02 ±6.81E-03	2.23E-01 ±9.31E-03	94.54% ±0.34%
Waveform_noise_small	10000	Base	4.85E-05 ±1.35E-06	6.33E-01 ±4.43E-02	82.38% ±0.57%
		Pretrained	1.01E-02 ±4.17E-04	5.03E-01 ±1.58E-01	**86.97%** ±3.33%

model became higher than that of the base model. It is interesting that the improvement in generalisation by using the pretraining was big on Waveform_noise_small, but this was not the case on Waveform_small. Recall the difference between Waveform_noise and Waveform is that Waveform_noise has 19 additional noisy features drawn from the standard normal distribution. It is not clear if the difference in the results was caused by the increased dimensionality or the additional noisy features. We conducted further experiments on synthetic data in order to investigate when the supervised pretraining helps improve generalisation. These experiments are presented in the next section.

4.6 Experiments on Synthetic Datasets

The experiments discussed here investigate the circumstances, where the proposed pretraining holds an advantage over the *he_normal* initialisation in terms of test accuracy. More specifically, we test the effect of data dimensionality, different types of noisy features, size of labelled training data and distance between class clusters on the performance of the proposed pretraining. All the experiments were conducted on synthetic datasets generated using the *make_classification* API in *sklearn* library. This is an adapted implementation of the algorithm used to generate the “Madelon” dataset [42].

Firstly, we test how data dimensionality, noisy features and redundant features affect the effectiveness of the supervised pretraining compared to *he_normal*. We generated 6 different datasets. All of the datasets have only two classes and have balanced class distribution. Each of the raw datasets had 10000 instances. The *class_sep* parameter was set to 0.01 for all datasets (the smaller the *class_sep*, the more difficult the classification). We applied the same preprocessing as the previous experiment. But in this experiment, we sampled a training set with 50 instances for each dataset. All the other experiment setups were exactly the same as described in Section 4.5. The characteristics of the 6 datasets are described below.

- Informative_10 has 10 attributes and all of the attributes are informative for the classification task.
- Informative_20 has 20 attributes and all of the them are informative.
- Redundant_20 has 20 attributes, but 10 of them are the exact copy of the other 10 informative attributes.
- In Std_normal_20, 10 of the 20 attributes are random values drawn from the standard normal distribution (mean 0, standard deviation 1), while the other 10 are informative.
- In Normal_20, 10 of the attributes are drawn from normal distributions whose means and standard deviations are kept the same as the other 10 informative attributes.
- In Shuffled_20, instead of drawn randomly from normal distributions, the 10 noisy attributes are simply shuffled informative attributes (the distribution of each of the noisy attributes is the same as the corresponding informative attribute).

The results are shown in Table 4.5. The base models for Informative_10 and Redundant_20 performed similarly. This is not too surprising considering Redundant_20 basically concatenated two copies of the features in Informative_10 together.

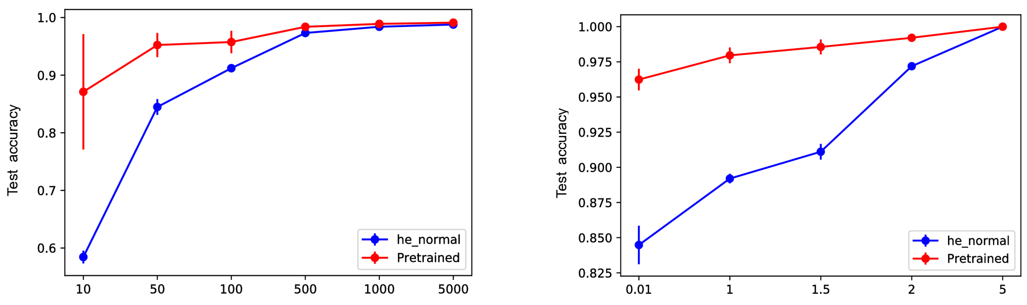
Table 4.5: Investigating the effect of dimensionality, noisy and redundant features on the performance of the supervised pretraining. We ran each case 10 times and all of the models were trained for 10000 epochs.

Data	Method	Train loss	Train acc.	Test loss	Test acc.
Informative_10	Base	6.50E-04 ±2.46E-05	100.00%	5.79E-01 ±6.89E-02	84.48% ±1.37%
	Pretrained	1.02E-02 ±1.41E-04	100.00%	1.43E-01 ±2.69E-02	**96.24%** ±0.77%
Informative_20	Base	3.56E-04 ±1.76E-05	100.00%	1.82E+00 ±7.84E-02	63.61% ±0.72%
	Pretrained	1.03E-02 ±2.87E-04	100.00%	4.71E-01 ±1.02E-01	**87.19%** ±2.64%
Redundant_20	Base	6.07E-04 ±2.50E-05	100.00%	5.86E-01 ±4.82E-02	84.41% ±0.93%
	Pretrained[1]	1.10E-02 ±7.33E-04	100.00%	4.64E-01 ±4.32E-02	**85.48%** ±1.35%
Std_normal_20	Base	3.87E-04 ±2.01E-05	100.00%	2.40E+00 ±7.50E-02	59.94% ±0.64%
	Pretrained	1.02E-02 ±2.83E-04	100.00%	2.13E-01 ±3.53E-02	**94.18%** ±1.09%
Normal_20	Base	4.12E-04 ±1.43E-05	100.00%	1.24E+00 ±5.77E-02	65.40% ±0.92%
	Pretrained	1.02E-02 ±1.60E-04	100.00%	1.84E-01 ±2.92E-02	**94.93%** ±0.87%
Shuffled_20	Base	3.42E-04 ±1.35E-05	100.00%	2.42E+00 ±8.56E-02	57.95% ±0.39%
	Pretrained	1.02E-02 ±1.15E-04	100.00%	1.74E-01 ±2.34E-02	**95.19%** ±0.55%

Compared to Informative_10, Redundant_20 neither has any additional information nor lacks any useful information. The improvement on generalisation using the supervised pretraining was around 11% on Informative_10. But for Redundant_20, the pretrained model failed to converge in 4 out of 10 runs. And there is no obvious improvement on test accuracy when pretraining is used. The generalisation gain on Informative_20 was more than 20%, while the gain on the Std_normal_20 was around 34%. The biggest gain (around 37%) occurred in Shuffled_20 dataset. The results suggest that both the increased dimensionality and the noisiness of the additional features both contribute to the generalisation improvement we observed on Waveform_noise_small in the previous experiment.

Next, we investigated how the size of labelled training data and the class separation would affect the generalisation advantage of the proposed pretraining over the *he_normal* initialisation. The experimental setup was exactly the same as the previous experiments, except the datasets used. When testing the effect of the size of labelled training data, we held all other factors constant including the class separation (*class_sep* = 0.01) and unlabelled data. The data had 10 attributes and all of them were informative. We tested six labelled training sets with 10, 50, 100, 500, 1000 and 5000 instances respectively. We ran each experiment 10 times. The results

[1]When running the pretrained model of Redundant_20, 4 out of 10 runs failed to converge, the



(a) Size of training set (b) Class separation

Figure 4.4: Comparison of the model trained with supervised pretraining (red) against the model initialised with *he_normal* (blue) in terms of test accuracy. The plots show the distributions of test accuracy. Each experiment was run 10 times.

can be found in Figure 4.4(a). The results show that as the size of labelled training data increases, the advantage of the pretraining gets smaller. And as expected, as the training set gets larger, the standard deviation of test accuracy shrinks. Finally, we tested the effect of class separation. Class separation in this case means the distance between classes. The smaller the class separation, the harder the classification. We created three datasets with three different levels of class separation by setting the *class_sep* parameter to 0.01, 1, 1.5, 2 and 5 respectively. All of the datasets had only 10 informative attributes. The labelled training sets had 50 instances for these experiments. Again, all the other experiment setups were the same as described earlier. Figure 4.4(b) shows the results of these experiments. The pretraining had the biggest gain in test accuracy over *he_normal* when the class separation is 0.01. However, as the class separation gets larger the advantage becomes smaller. Both of these experiments support the hypothesis that what we are learning during pretraining is a good prior for the model. The advantage of a good prior is larger when the data size is small and when the classification problem is difficult.

4.7 Discussion

By inspecting the results closely, we noticed that the standard deviation of the test accuracy for the pretrained model was bigger than the base model without pretraining for a few cases. This is contradictory to the findings in unsupervised pretraining.

results shown here were collected from the 6 converged runs.

Unsupervised pretraining was shown to help lower the variance of test accuracy [30]. The results in Table 4.4 are not statistically significant. One possible explanation for this might be the weights learned during the pretraining across different runs were very different. Recall that we used a pretraining validation set to stop the pretraining. Another way to do this is to run the pretraining using different epochs and choose the epoch that gives the best validation accuracy on the original classification problem. Then we can run the experiments multiple times using this fixed number of epochs. This may be a more stable method. Another possible explanation for the large standard deviation is the small size of the training data. The standard deviation of the test accuracy tends to be larger when training data is small. The difficulty of the classification problem may also affect the statistical significance of the results. As the experiments on synthetic data indicate, the advantage of the proposed pretraining method disappears when the clusters are far apart from each other. The complexity of the target decision boundary that separates different classes may also affect the effectiveness of the pretraining method. Lastly, we also expect label noise in the data to reduce the advantage provided by the pretraining. The exact reasons why the proposed method is more effective on certain datasets need to be explored further.

The pretraining phase means more training time. The proposed pretraining only adds a small amount of additional training time in our experiments. For instance, the pretraining phase for the MAGIC dataset took on average 241 seconds, about 4.67% of the average total training time of 5061 seconds. The computational complexity of the random permutation of features is $O(dim * perm)$, where dim is the number of features in the data and $perm$ is the complexity of the random permutation on a single feature. Suppose the vanilla stochastic gradient descent and a simple fully-connected neural network (with the same number of hidden nodes in all layers) are used, the complexity of pretraining is $O(epochs * batch * (l * M))$, where l is the number of layers, M is the complexity of matrix multiplication, $epoch$ is the number of epochs to train the model for and $batch$ is the batch size (the number of examples in a batch). Note that the number of epochs required for the training to converge depends on the size and complexity of the data and the chosen architecture. It also depends on the hyperparameters such as learning rate and batch size. Larger and more complex models and data generally take longer to converge. However, the pretrained models can be potentially reused multiple times and for multiple downstream tasks. This means that the time and cost spent on pretraining can be amortised for the downstream tasks.

Note that in all of our experiments, we did not fine tune the pretraining or the weight transferring process. One can possibly improve the results further by carefully and tediously choosing when to stop the pretraining, which layers to reuse and whether to freeze some layers for a certain number of epochs. Our main goal is to show that the proposed method can help improve generalisation when only limited training data is available, instead of achieving state-of-the-art result on a particular dataset. We used the same experimental setup across all the experiments (except number of epochs) to not bias any particular setting.

The proposed pretraining requires well-defined features. This makes the proposed method more suitable for tabular data. To apply the proposed pretraining to unstructured data such as images and audio clips, one can firstly extract manually-engineered features from the raw and unstructured data, and then apply the proposed pretraining method on the extracted features. Alternatively, techniques such as autoencoders can be used to learn features from the raw data before applying the proposed pretraining.

4.8 Summary

Current default initialisation methods such as *he_normal* are designed to stabilise the training, especially for deep neural networks. However, they are not necessarily optimal for generalisation. When labelled data is limited, the choice of initial weights becomes more important in deciding what final model we will end up with. Note that a training set can be large but still be limited if the task is difficult. We demonstrated that inappropriate initial weights of neural networks do indeed lead to lower test accuracy. We then proposed a supervised pretraining method to improve the generalisation in binary classification problems for tabular data. During the pretraining, a model is learned to identify real data from shuffled data. Then the learned weights are reused in the original problem. Based on the experimental results on four UCI datasets and synthetic datasets, the supervised pretraining leads to higher test accuracy than *he_normal* initialisation, when there is a lack of labelled training data. The experiments on synthetic datasets showed that the proposed supervised pretraining is more effective on datasets with higher dimensionality and noisy features. Finally, the proposed method has a bigger advantage over *he_normal* in terms of test accuracy when the class separation is small.

5 Investigating the Effect of Novel Classes in Semi-Supervised Learning

5.1 Overview

Neural networks have been successfully applied to challenging tasks such as image or speech recognition. However, in order to achieve good performance, it is necessary to have a large amount of labelled training data. This requires humans to painstakingly label many examples. The labelling process can be costly and time consuming. To address this issue, many different semi-supervised learning (SSL) algorithms have been proposed in recent years [59, 69, 73, 93, 116]. The reported results on some of the benchmark datasets for semi-supervised learning using a small amount of labelled examples are approaching the performance of supervised learning with all the examples labelled. For instance, Tarvainen and Valpola (2017) managed to achieve an error rate of 9.11% on ImageNet 2012 with only 10% of the instances being labelled [116].

Most research on semi-supervised learning assumes that the distribution of unlabelled data is the same as the distribution of labelled data. However, this assumption does not always hold in practice. We use semi-supervised learning when we do not have the resources to label all of the available data. This means that we cannot check all the data points to make sure they are only from the classes we are interested in. Therefore novel classes might be present in the unlabelled data when we apply semi-supervised learning in practice. We define **novel classes** as classes that are only present in the unlabelled training data, but not in the labelled data (including test data). We assume future unseen data will always be from the same distribution as the labelled training data. Hence, our test datasets only contain classes that are present in the labelled training data. Note that our learning scenario is different to that of novelty detection [8] and domain shift or transfer [96], where the future unseen data can be distributed differently from the labelled training data.

We answer two **research questions** in this chapter:

1. How do novel classes in unlabelled data affect the generalisation performance of semi-supervised algorithms for neural networks?

2. How can we come up with a method to mitigate the negative effect of novel classes on the generalisation performance of neural networks?

By answering these questions, we make the following **contributions**:

- We empirically show that the presence of novel classes can degrade the performance of semi-supervised algorithms for neural networks.

- We propose a distance-based weighting framework that assigns weights to unlabelled data according to the distance between an unlabelled example and the labelled dataset. This framework assumes that the unlabelled examples that are far away from the labelled examples are more likely to belong to the novel classes, and should be assigned lower weights in training.

- We propose a 1-nearest-neighbour based implementation of the framework. The experimental results show that when the proposed method is applied to semi-supervised algorithms, the degradation in generalisation performance caused by novel classes becomes statistically insignificant. The proposed method can be applied to any semi-supervised learning algorithm that includes unlabelled data in the loss function.

The rest of the chapter is organized as follows. In Section 5.2, we introduce two semi-supervised learning algorithms that are used in our experiments. We propose a method to reduce the effect of novel classes in Section 5.3. In Section 5.4, we describe the experiments designed to address the two research questions. We analyze the experimental results in Section 5.5. We give further discussions on our experiments as well as drawbacks of our proposed method in Section 5.6. Finally, we summarise this chapter in Section 5.7.

5.2 Background

We use Pseudo-Label [73] and Mean Teacher [116] as two example algorithms in our experiments to investigate the effect of novel classes in semi-supervised learning for neural networks. Pseudo-Label is based on the *cluster assumption* while Mean Teacher is based on the *smoothness assumption* (assumptions in semi-supervised learning are reviewed in Section 3.3). We limit the scope of this research to self-training based methods such as Pseudo-Label and Mean Teacher. We leave the experimentation with other types of semi-supervised learning to future research. In this section, we describe how both Pseudo-Label and Mean Teacher work.

5.2.1 Pseudo-Label

Pseudo-Label [73] is a self-training based semi-supervised learning method [73]. *Pseudo-labels* are defined as predictions made by the current model for unlabelled data. The Pseudo-Label method treats these *pseudo-labels* as true labels for the unlabelled data during training. Pseudo-labels are updated after each epoch. So the training process is similar to that of supervised training, except that pseudo-labels are used for the unlabelled data instead of true labels. Cross-entropy loss is usually used for classification problems in supervised learning. Let us define cross-entropy loss as

$$H(y, \hat{y}) = -\sum_{c=1}^{C} y_c \log \hat{y}_c \tag{5.1}$$

where y is the class label encoded in one-hot encoding, $\hat{y}$ is the model output and C is the number of classes. Then the loss function for Pseudo-Label is defined as

$$J = \frac{1}{m}\sum_{i=1}^{m} H(y_i, \hat{y}_i) + a(t)\frac{1}{m'}\sum_{j=1}^{m'} H(y'_j, \hat{y}_j) \tag{5.2}$$

where m is the number of labelled examples and m' is the number of unlabelled examples. The class label for the ith labelled example is denoted by y_i while y'_j denotes the *pseudo-label* for the jth unlabelled example. Model outputs for the labelled and unlabelled examples are denoted by $\hat{y}_i$ and $\hat{y}_j$ respectively. The first term in Equation 5.2 corresponds to the cross-entropy loss for the labelled data, and the second term is the cross-entropy loss for the unlabelled data. The importance of the loss related to unlabelled data is controlled by a hyperparameter $a(t)$. Intuitively speaking, the second term of the loss function tries to reduce the difference between the predictions of the current model and that of the model in the previous epoch on unlabelled data. The scheduling of the hyperparameter $a(t)$ is important when applying Pseudo-Label. If $a(t)$ is set too high in the beginning, the model will fail to learn because the model is likely to be bad at prediction initially. We use a scheduling function to schedule $a(t)$ as defined in Lee [73].

$$a(t) = \begin{cases} 0 & \text{if } t < T_1 \\ \frac{t-T_1}{T_2-T_1}a & \text{if } T_1 \le t < T_2 \\ a & \text{if } T_2 \le t \end{cases}$$

where a, T_1 and T_2 are set by users, and t is the current training epoch. So $a(t)$ is initially set to be 0, and gradually increased after each epoch before being set to be the user selected a for the remaining epochs. Lee [73] showed that Pseudo-Label has an effect of entropy regularisation on MNIST dataset, which provides an explanation for the success of Pseudo-Label.

5.2.2 Mean Teacher

Mean Teacher [116] is also a self-training based semi-supervised method. Mean Teacher keeps an exponential moving average (EMA) of the model that is considered as the teacher model. Mean Teacher then tries to reduce the difference between the student model (current model) and the teacher model. The rationale for using a EMA model as the teacher model is that the EMA model provides more stable “target” outputs. The loss function of Mean Teacher is defined as

$$J = \frac{1}{m}\sum_{i=1}^{m} H(y_i, \hat{y}_i) + a(t)\frac{1}{2(m+m')}\sum_{j=1}^{m+m'} (\hat{y}_j - \hat{y}_j{}')^2 \tag{5.3}$$

where $\hat{y}$ is the output of the student model, and $\hat{y}'$ is the output of the teacher model (the exponential moving average of the student model). The first term in the loss function is the cross-entropy loss of the labelled data. The second term is the mean squared error (MSE) that measures the distance between the outputs of the student model and the teacher model. Again, a hyperparameter $a(t)$ is used to balance the two losses. Mean Teacher uses a Gaussian ramp-up function to schedule hyperparameter $a(t)$ as defined in Tarvainen and Valpola [116] and Laine and Aila [69]. It is defined as follows.

$$a(t) = \begin{cases} a * exp[-5(1 - \frac{t}{T})^2] & \text{if } t \leq T \\ a & \text{if } t > T \end{cases}$$

where t is the current epoch, T is the ramp-up period chosen by user, and a is the base parameter also set by user. We applied the same ramp-up function in our implementation of Mean Teacher. Apart from a, Mean Teacher introduced another hyperparameter, the EMA decay rate η, that controls the coefficient λ in EMA.

$$\lambda(s) = min(1 - \frac{1}{s+1}, \eta)$$

where s is the training step (number of weight updates) and η is chosen by user. $\lambda(s)$ controls the importance of the EMA model relative to the current model when updating the teacher model. Note that in Mean Teacher, the teacher model is updated for each training step instead of each epoch. The above function indicates that we use the true average of the models learned so far as the teacher model in the beginning, before we start using the EMA model.

5.3 Distance Based Weighting Framework

As described in Section 5.2, Pseudo-Label and Mean Teacher attempt to train the current model (student model) to match the predictions (outputs) of the model in the last epoch (teacher model) for unlabelled data. The existence of novel classes in the unlabelled data can potentially push the decision boundary away from the one learned using clean unlabelled data. The experimental results in Section 5.4.1 suggest that the presence of novel classes can hurt the performance of the algorithms. In this section, we propose a distance based weighting framework and our implementation of it to reduce the negative effect of novel classes.

We attempt to reduce the negative effect of novel classes by assigning individual weights to unlabelled data. The intuition is simple: if we can lower the weights of the novel examples in the training loss, these novel examples will have less impact on the training. Our proposed framework works as follows.

1. Compute the distance d_j for each unlabelled data point to the known classes.
2. Apply a function to transform distance d_j into weight w_j.
3. Assign weight w_j to each unlabelled data point in the loss function.

The specific implementation of this framework depends on the definitions of distance d_j and transformation function that transforms the distance into a weight. We assume that the distance to the known classes is higher for an example from the novel classes. The transformation function should transform high-distance values into low-weight values, and ideally be bounded to a certain range. Next, we describe our implementation of this framework.

Table 5.1: Architecture A. The architecture used for MNIST and Fashion-MNIST.

Layer	**Parameters**
Input	28 × 28 grayscale images
Convolutional	20 filters, 5 × 5, valid padding, ReLU
Pooling	Maxpool 2 × 2
Convolutional	50 filters, 5 × 5, valid padding, ReLU
Pooling	Maxpool 2 × 2
Dense	800 → 500, ReLU
Softmax	500 → 5

5.3.1 1-Nearest-Neighbour Based Weighting

We define the distance d_j as the Euclidean distance between an unlabelled example and its closest neighbour in the labelled training data. Hence, we essentially run a 1-Nearest-Neighbour (1NN) algorithm with the labelled data being the training set, and compute the 1NN distance d_j for each unlabelled example. Note that we basically have n clusters, where n is the number of labelled examples in the training set. The centroid for each cluster is one of the labelled examples. We then need to define a function to transform all the distances into weights.

A naive way to transform distance into a weight can be defined as follows.

$$w_j = 1 - scale(d_j)$$

where *scale* normalises d_j to the range [0, 1] within its cluster. This transformation is unlikely to work. Because it always assigns 0 to the unlabelled example that is furthest away from its centroid, regardless of the value of the distance. However, it is possible that every unlabelled example is close to the centroid (labelled example) and belongs to the same class, hence all of the unlabelled examples should be assigned large weights. We use the following transformation instead:

$$w_j = \frac{1}{exp(\beta d_j)} \tag{5.4}$$

where β is a hyperparameter that controls how quickly the weight approaches zero as the distance increases. Using this transformation we can avoid the problem described above. The pseudocode for computing the weights is shown in Algorithm 1.

Algorithm 1 Computing the weights for unlabelled data using 1-nearest-neighbour

```
Require: $L, U, \beta$
1: for $x_j \in U$ do
2:     Compute the distance $d_j = \min_{x_i \in L} distance(x_j, x_i)$
3:     Compute the weight $w_j = \frac{1}{exp(\beta d_j)}$
4: end for
```

Lastly, we apply the computed weights to the loss function. For instance, the loss function for Pseudo-Label now becomes

$$J = \frac{1}{m}\sum_{i=1}^{m} H(y_i, \hat{y}_i) + a(t)\frac{1}{m'}\sum_{j=1}^{m'} w_j H(y'_j, \hat{y}_j) \tag{5.5}$$

where w_j is weight for the jth unlabelled example. And the loss function for Mean-Teacher is now

$$J = \frac{1}{m}\sum_{i=1}^{m} H(y_i, \hat{y}_i) + a(t)\frac{1}{2(m+m')}\sum_{j=1}^{m+m'} w_j(\hat{y}_j - \hat{y}_j{}')^2 \tag{5.6}$$

where w_j is the weight for the jth example. The experiments designed to evaluate the proposed method are explained in Section 5.4.2, and the experimental results are analysed in Section 5.5.2.

5.4 Experimental Setup

In this section, we describe the experiments that investigate the effect of novel classes on Pseudo-Label and Mean Teacher, and test the effectiveness of the proposed method in reducing the effect of novel classes. All the experiments were implemented in Pytorch [89][1]. All the hyperparameters used in the experiments are reported in Appendix B. The experimental results are discussed in Section 5.5.

Table 5.2: Architecture B. The architecture used for CIFAR-10.

Layer	**Parameters**
Input	32 × 32 RGB images
Convolutional	32 filters, 3 × 3, same padding, ReLU
Convolutional	32 filters, 3 × 3, valid padding, ReLU
Pooling	Maxpool 2 × 2
Convolutional	64 filters, 3 × 3, same padding, ReLU
Convolutional	64 filters, 3 × 3, valid padding, ReLU
Pooling	Maxpool 2 × 2
Dense	2304 → 512, ReLU
Softmax	512 → 5

5.4.1 Experiments to Demonstrate the Effect of Novel Classes

We conducted experiments on three popular image-recognition datasets: MNIST [71], Fashion-MNIST [123] and CIFAR-10 [66]. The characteristics of these datasets and the preprocessing are explained below.

- **MNIST** is a 28 × 28 grayscale image dataset of handwritten digits. The dataset contains 60,000 training images and 10,000 test images. There are 10 classes in total (0 to 9). The classes are distributed evenly. In our experiments, a validation set containing 5,000 images is created from the training

[1]The source code can be found on GitHub: https://github.com/superRookie007/novel-classes-ssl. Supplementary materials and additional experimental results are also published there.

images using stratified sampling. This leaves 55,000 images for training. We treat classes {5, 6, 7, 8, 9} as novel classes. This means the labelled training set, validation set and test set only contain classes {0, 1, 2, 3, 4}, while the unlabelled data includes all classes. The validation set and test set are around half of their original sizes after taking out the novel classes. The number of labelled examples in the labelled training set is set to 50.

- **Fashion-MNIST** follows the same format and structure as MNIST, but it contains fashion items instead of handwritten digits. The ten classes are t-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. Again, we created a validation set containing 5,000 examples (including novel classes) from the 60,000 training images. Classes {sandal, shirt, sneaker, bag, ankle boot} are considered as novel classes. The number of labelled examples in the labelled training set is set to 100.

- **CIFAR-10** is a 32×32 RGB image dataset. There are 50,000 training images and 10,000 test images. There are 10 balanced classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. We split the training images into a validation set of 5,000 examples (including novel classes) and a training set of 45,000 examples. Classes {dog, frog, horse, ship, truck} are treated as novel classes. The labelled training set has 3000 examples.

An unlabelled dataset is **pure** if it does not contain any example from the novel classes. A **dirty** unlabelled dataset includes all the examples from the novel classes. For each dataset, we experimented with three training scenarios as follows.

- *Exclude*: supervised training without unlabelled data.
- *Include_pure*: semi-supervised training with pure unlabelled data.
- *Include_dirty*: semi-supervised training with dirty unlabelled data.

Each experiment was run 50 times for MNIST and Fashion-MNIST, and 20 times for CIFAR-10 because of the much longer training time. The seed for splitting the training data into labelled and unlabelled data was different in each run. The seeds were kept the same for each of the three training scenarios.

Both Pseudo-Label and Mean Teacher are model or architecture agnostic. This means that we can use whatever architecture we want with these algorithms without changing the rest of the code. We used Architecture A defined in Table 5.1 for MNIST and Fashion-MNIST. Architecture B defined in Table 5.2 is used for CIFAR-10. We use the standard mini-batch stochastic gradient descent (SGD) to train our models. We use a batch size of 100 for all our experiments. The following learning rate scheduler is used:

$$lr(t) = lr \times \gamma^t$$

where lr is the base learning rate set by the user, t is the current epoch and γ is the multiplicative factor of the learning rate decay process. All hyperparameters were set using validation sets. The specific hyperparameters used in each experiment can be found in Appendix B. Experimental results are discussed in Section 5.5.1. All results were obtained from the test sets.

Note that the architectures used in this paper are different from the ones used in the original papers. We used simple architectures to limit the computing power and running time for our experiments. It is possible that different architectures can potentially change how novel classes affect these algorithms. However, this is a different research question and we will leave it to future research. Lee [73] applied unsupervised pretraining in addition to the Pseudo-Label algorithm in order to improve the performance further. Data augmentation was applied in the original implementation of Mean Teacher [116]. We do not apply any data augmentation or unsupervised pretraining in our implementation, because our goal is not to achieve state-of-the-art results on benchmark datasets. The additional complexity only makes it more difficult to interpret the experimental results. We only intend to test if novel classes have any negative effect on the performance of semi-supervised algorithms, while holding other factors constant.

5.4.2 Evaluation of the Proposed Method

We evaluated the proposed method using both Pseudo-Label and Mean Teacher on MNIST [71], Fashion-MNIST [123] and CIFAR-10 [66]. The data processing is exactly the same as described in Section 5.4.1. The only difference is that we have to compute the weight for each unlabelled data point using the proposed method. And we now have an additional hyperparameter β to tune.

Note that when we apply the 1-Nearest-Neighbour algorithm, we can use raw images or low-dimensional vector representations of the images. Since distance based methods suffer from *the curse of dimensionality*, we postulate that we can get better performance if dimensionality reduction is applied beforehand. We tested using Variational Autoencoder (VAE) [58] to learn low-dimensional representations from the images, and then applied the 1-Nearest-Neighbour to these representations. For MNIST and Fashion-MNIST, we used a simple multi-layer perceptron (MLP) based encoder and decoder as introduced by Kingma and Welling [58]. The dimensionality of the hidden representation was set to 10. For CIFAR-10, we used convolutional encoder and transposed convolution for up-sampling in the decoder. The dimensionality of the vector representation for CIFAR-10 was set to 20.

To test the effectiveness of the proposed method, we apply the computed weights in Pseudo-Label and Mean Teacher, and check if this improves the performance on include_dirty where novel classes are present in unlabelled data. We add two additional training scenarios to our experiments as follows.

- *With_weights_raw*: weights are computed using raw images, and model is trained with dirty unlabelled data.
- *With_weights_vae*: weights are computed using low-dimensional representations learned using VAE, and model is trained with dirty unlabelled data.

Both with_weights_raw and with_weights_vae were trained using the same dirty unlabelled data on which include_dirty was trained. Similar to the experiments in Section 5.4.1, we ran each experiment 50 times for MNIST and Fashion-MNIST, and 20 times for CIFAR-10 due to longer training time. The data seed was different for each run. We used the same set of seeds across all experiments. Again, all hyperparameters were set using validation sets, the hyperparameters used in each experiment can be found in Appendix B. Experimental results are reported in Section 5.5.2. All the results reported were obtained from the test sets.

5.5 Experimental Results

In this section, we discuss the experimental results obtained from the experiments described in the last section. Test accuracy results for Pseudo-Label and Mean Teacher are reported in Tables 5.3 and 5.4 respectively. Since the differences in test accuracies across different settings are small, we provide more detailed analysis of the results using the *Friedman test* and *Nemenyi post-hoc test* in the next two subsections. Demšar [22] provides a great discussion on Friedman and Nemenyi tests along with other statistical tests for comparing classifiers.

Table 5.3: All the experimental results in the paper for Pseudo-Label. It shows the mean and standard deviation of test accuracy for each dataset and experiment setting.

	MNIST	Fashion-MNIST	CIFAR-10
exclude	93.63% ±1.51%	77.15% ±1.56%	63.85% ±1.20%
include_pure	94.52% ±1.71%	77.02% ±1.99%	65.37% ±1.19%
include_dirty	92.15% ±2.27%	76.14% ±2.02%	64.45% ±1.11%
with_weights_raw	92.74% ±2.11%	77.01% ±1.55%	65.14% ±1.01%
with_weights_vae	93.79% ±1.39%	76.86% ±1.76%	65.88% ±1.22%

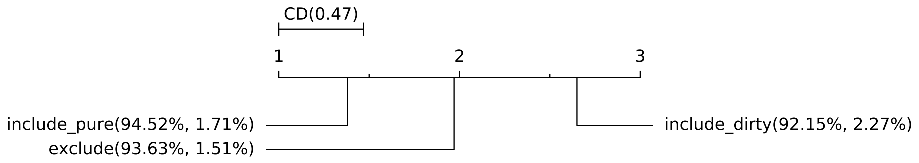
Table 5.4: All the experimental results in the paper for Mean Teacher. It shows the mean and standard deviation of test accuracy for each dataset and experiment setting.

	MNIST	Fashion-MNIST	CIFAR-10
exclude	93.63% ±1.51%	77.15% ±1.56%	63.85% ±1.20%
include_pure	94.06% ±1.68%	78.58% ±1.93%	66.56% ±0.96%
include_dirty	92.97% ±2.03%	76.95% ±2.04%	64.94% ±1.32%
with_weights_raw	93.11% ±1.99%	77.19% ±1.83%	66.04% ±1.11%
with_weights_vae	93.42% ±1.46%	77.14% ±1.85%	66.24% ±0.84%

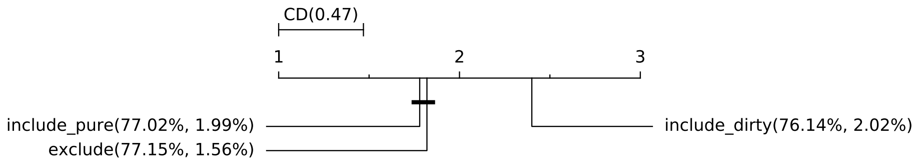
5.5.1 Effect of Novel Classes

To find out if novel classes really affect the performance, we used the Friedman test ($p = 0.05$) to test if the test accuracies under the three different settings (include_pure, include_dirty and exclude) are statistically different for each dataset. The details of different settings can be found in Section 5.4. The test suggested that the distributions of test accuracies are indeed different. Then we performed the Nemenyi post-hoc test for pairwise comparisons. We used 0.05 as the confidence level. The results for the Nemenyi tests are shown as CD graphs in Figure 5.1 for Pseudo-Label and Figure 5.2 for Mean Teacher. The CD graph shows the average ranking of each setting. The lower the ranking the better. The difference in average ranking is statistically significant if there is no bold line connecting the two settings. The mean and standard deviation of test accuracy are shown in parenthesis.

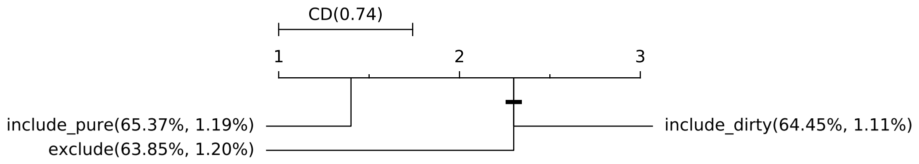
Figure 5.1 shows that include_dirty is statistically significantly worse than exclude and include_pure on both MNIST and Fashion-MNIST for Pseduo-Label. For CIFAR-10, there is no significant difference between include_dirty and exclude, but include_dirty is still significantly worse than include_pure. The results for Mean



(a) Pseudo-Label on MNIST



(b) Pseudo-Label on Fashion-MNIST



(c) Pseudo-Label on CIFAR-10

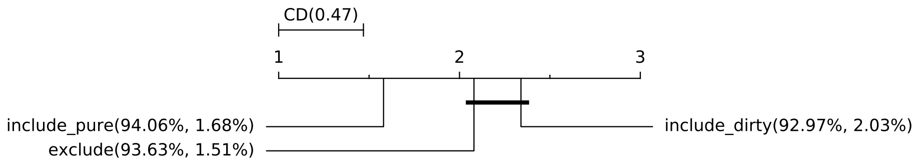
Figure 5.1: Comparison of supervised training without unlabelled data, Pseudo-Label with clean unlabelled data and Pseudo-Label with dirty unlabelled data on MNIST (a), Fashion-MNIST(b) and CIFAR-10 (c).

Teacher are shown in Figure 5.2. For all three datasets, include_dirty is not significantly different from exclude, but it is significantly worse than include_pure.

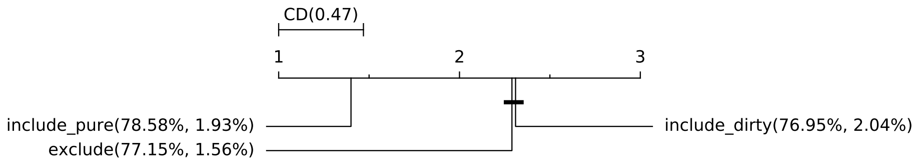
Overall, we can clearly see that novel classes in unlabelled data have a negative effect on the performance of Pseudo-Label and Mean Teacher. When novel classes are present in the unlabelled data, the performance of these algorithms is significantly lower than that trained using clean unlabelled data without novel classes.

5.5.2 Effectiveness of 1NN Based Weighting

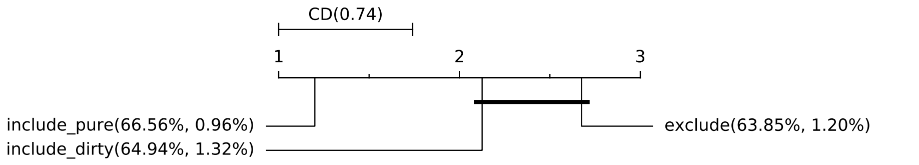
We have shown that the presence of novel classes lower the performance of Pseudo-Label and Mean Teacher, we now test if our proposed weighting method can reduce this negative effect. Figure 5.3 shows the Nemenyi test results for Pseudo-Label. For all three datasets, with_weights_vae consistently outperformed include_dirty. This means that when the low-dimensional representations are used for weight computation, our proposed method improved the performance of Pseudo-Label when novel classes are present. When raw images were used to compute the weights,



(a) Mean Teacher on MNIST



(b) Mean Teacher on Fashion-MNIST



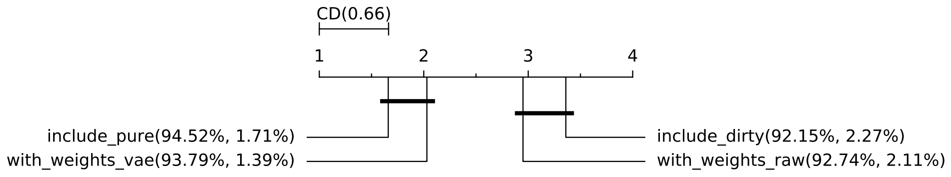
(c) Mean Teacher on CIFAR-10

Figure 5.2: Comparison of supervised training without unlabelled data, Mean Teacher with clean unlabelled data and Mean Teacher with dirty unlabelled data on MNIST (a), Fashion-MNIST(b) and CIFAR-10 (c).

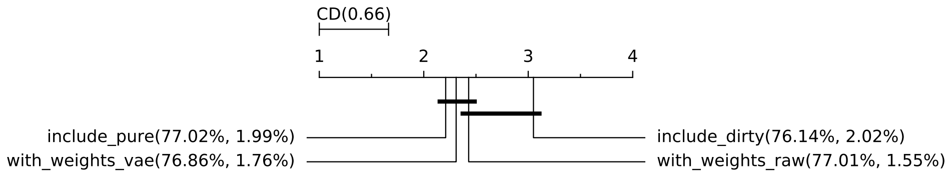
our proposed method did improve the average ranking, but the improvement is not statistically significant for all datasets. There is no significant difference between with_weights_vae and include_pure for all three datasets. This suggests that with our proposed weighting method, the performance of Pseudo-Label does not suffer significantly when novel classes are present in unlabelled data.

The experimental results for Mean Teacher are shown in Figure 5.4. For MNIST, with_weights_vae is not significantly better than include_dirty, but is statistically the same as include_pure. Include_pure is significantly better than all the other three settings on Fashion-MNIST. Include_dirty performed significantly worse than the other settings on CIFAR-10 and include_pure is statistically the same as with_weights_vae and with_weights_raw.

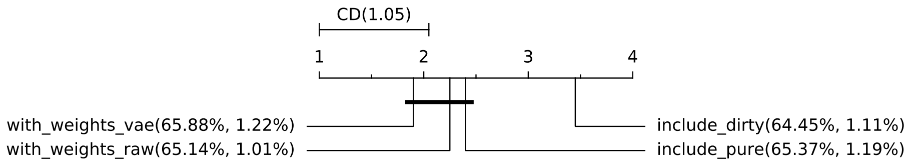
Overall, when our proposed method is applied to Pseudo-Label and Mean Teacher, the performance does not suffer as much when novel classes are present. There is actually no significant difference in performance regardless of the presence of novel classes except for one out of the 6 cases (when Mean Teacher was used on



(a) Pseudo-Label on MNIST



(b) Pseudo-Label on Fashion-MNIST



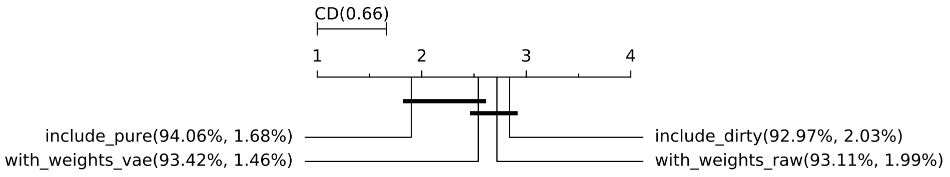
(c) Pseudo-Label on CIFAR-10

Figure 5.3: Comparison of Pseudo-Label models trained in the following conditions: dirty unlabelled data without weights, dirty unlabelled data with weights computed from raw images, dirty unlabelled data with weights computed from low-dimensional representations and clean unlabelled data without weights.

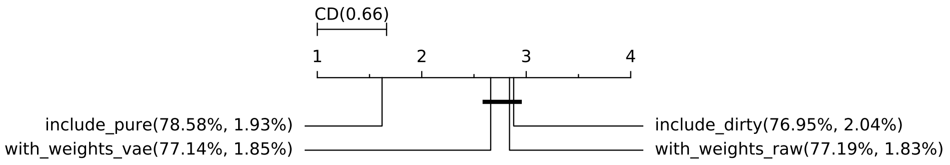
Fashion-MNIST in Figure 5.4), when our proposed method is used. The advantage of using low-dimensional representations over raw images to compute the weights is not statistically significant, according to our experimental results.

5.6 Discussion

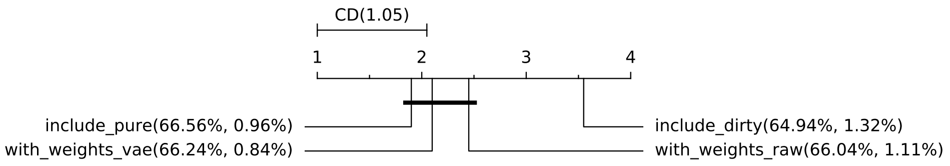
The experimental results in Section 5.5.1 suggest that novel classes in unlabelled data can hurt the performance of semi-supervised learning algorithms. This is in agreement with the findings by Oliver et al. [88]. Oliver et al. [88] evaluated semi-supervised algorithms in fair and realistic settings. The implementations of all the algorithms evaluated used the same architectures, data augmentations, and optimisation methods. They showed that the supervised baselines used in semi-supervised



(a) Mean Teacher on MNIST



(b) Mean Teacher on Fashion-MNIST



(c) Mean Teacher on CIFAR-10

Figure 5.4: Comparison of Mean Teacher models trained in the following conditions: dirty unlabelled data without weights, dirty unlabelled data with weights computed from raw images, dirty unlabelled data with weights computed from low-dimensional representations and clean unlabelled data without weights.

learning literature were too weak. In their implementation, the gap between semi-supervised training and the supervised baseline was much smaller than the results reported in the original research. They also studied the effect of distribution mismatch between the labelled data and the unlabelled data. They found that adding unlabelled data from the novel classes can hurt the performance. While Oliver et al. [88] only tested the effect of novel classes on CIFAR-10, we also conducted experiments on MNIST and Fashion-MNIST. In addition to demonstrating the negative effect of novel classes, we also propose the first novel solution for reducing this effect.

Oliver et al. [88] did not re-tune the hyperparameters for the experiments that included novel classes, however, we did tune the a hyperparameter while holding other parameters unchanged. The hyperparameter a controls the importance of the regulating term in the loss function. It is essentially a universal weight applied to all the unlabelled data. If we lower a, apart from reducing the negative effect of novel classes, it will also lower the positive effect of the rest of the unlabelled data. Figure 5.5 shows the comparison of Mean Teacher models with different a values (holding other hyperparameters constant) trained on CIFAR-10 with novel classes. The a value of 50 was used in our experiments in Section 5.5.1. Lowering the hyperparameter a did not improve the performance of Mean Teacher. Hence, we need to assign individual weights to each unlabelled example to keep the benefit of the clean unlabelled data and at the same time weaken the impact of the novel classes.

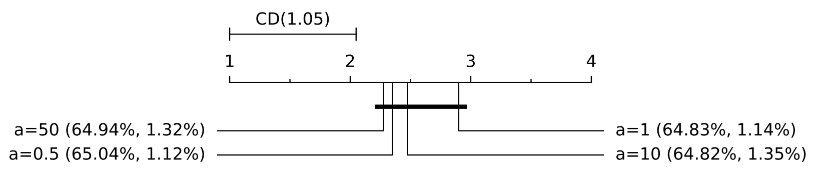
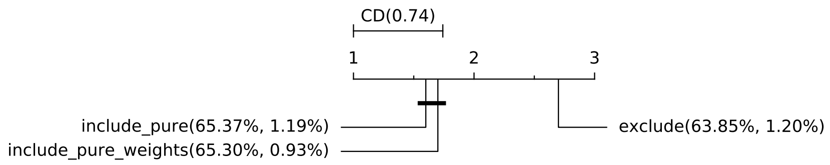


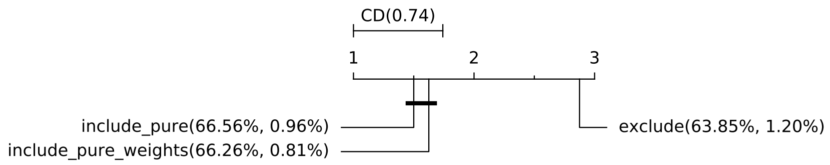
Figure 5.5: Mean Teacher models with different a values were trained on CIFAR-10 with dirty unlabelled data. We used a value of 50 in our experiments in Section 5.4.1.

According to the experimental results in Section 5.5.2, the advantage provided by learning low-dimensional representations of images when computing weights over using raw images is not significant. It could be due to the hyperparameter tuning or the choice of architectures. The advantage can potentially be bigger when more advanced architectures are used in training the VAE. It would be interesting to explore other unsupervised learning algorithms to learn the latent representations. A major downside of using raw images to compute weights and distances is the running time due to the big dimensionality of raw images. The brute force method for computing distances has a time complexity of $O(|U| * |L| * dim)$, where $|U|$ and $|L|$ are the sizes of the unlabelled and labelled sets, and dim is the dimensionality of the data. This is an obstacle if we have to compute the weights frequently. Learning latent representation by training an autoencoder takes time. However, this training time can be amortised since we can reuse the encoder every time we compute weights. We only tested the proposed method on Pseudo-Label and Mean Teacher, however, it can also be applied to other semi-supervised learning algorithms.

We have so far shown that the proposed method can improve the accuracy of Pseudo-Label and Mean Teacher when novel classes are present in the unlabelled data. However, it is not clear whether the improvement is a result of the reduced negative effect of the novel classes or the gain provided by the individual weights on the clean unlabelled data. In order to answer this question, we applied the proposed method to both Pseudo-Label and Mean Teacher on CIFAR-10 with clean unlabelled data. We only used the implementation that computes weights using the VAE encodings instead of the raw images, because this implementation always had better average rankings in the previous experiments. The experiments were repeated 20 times. The results are shown in Figure 5.6. The proposed method did not improve the test accuracy when the unlabelled data was clean (without novel classes) for both Pseudo-Label and Mean Teacher. This suggests that the improvement on accuracy when the unlabelled data contains novel classes is due to the reduced negative effect of the novel classes in the unlabelled data.



(a) Pseudo-Label on CIFAR-10



(b) Mean Teacher on CIFAR-10

Figure 5.6: Comparison of models trained in the following conditions: supervised training without unlabelled data (*exclude*), semi-supervised training on clean unlabelled data (*include_pure*) and semi-supervised learning with weights computed from low-dimensional representations on clean unlabelled data (*include_pure_weights*).

Lastly, we introduced a hyperparameter β in our proposed method. A validation set is needed to tune β along with other hyperparameters in semi-supervised algorithms. In order to be confident that the tuned hyperparameters will work well on future unseen data, the validation set has to be sufficiently large. We only use semi-supervised algorithms when labelled data is scarce. It is possible that we will have larger gain in performance by using most of the labelled data as training data instead of validation set. However, in most of the literature on semi-supervised learning the validation set used is usually larger than the labelled training set. This problem has been studied in Oliver et al. [88].

5.7 Summary

Even though it is a common practice to assume unlabelled data has the same distribution as the labelled data in semi-supervised learning literature, this assumption is not always true in practice. In this chapter, we empirically showed that novel classes in unlabelled data can hurt the generalisation performance of semi-supervised learning algorithms. We then proposed a 1-nearest-neighbour based method to assign weights to unlabelled data, in order to reduce the negative effect of novel classes. The experimental results showed that when our proposed method is applied to Pseudo-Label and Mean Teacher, the decrease in performance due to the presence of novel classes becomes statistically insignificant. This suggests that assigning individual weights to unlabelled data is a promising approach to dealing with novel classes in semi-supervised learning.

6 Measuring Output Fluctuation During Training for Active Learning

6.1 Overview

An obvious strategy to improve the generalisation of a classifier, when the labelled training set is small but there is an abundant pool of unlabelled data available, is collecting more labelled data. A simple strategy is to sample unlabelled data randomly and label the sampled examples. However, doing so may be inefficient and wasteful, because not all examples are equally useful in improving the performance of the classifier. Additionally, labelling can be a time-consuming and expensive practice. Ideally, we should only query data examples that are more likely to improve the performance of a classifier. This is called **active learning**, whose goal is to improve the performance of a machine learning model as much as possible by strategically querying as few examples as possible. Different types of active learning and specific algorithms are reviewed in Chapter 3.4.

We consider a specific type of active learning in this chapter: **pool-based** active learning. In pool-based active learning, a small amount of labelled data and a large pool of unlabelled data are available. All examples in the unlabelled data are evaluated at once before each query, unlike **stream-based** active learning where unlabelled examples are evaluated individually, independently and sequentially. This allows us to evaluate each example against all the other examples and only select the top examples. Because it evaluates unlabelled examples sequentially and independently from each other, stream-based active learning is affected by the order in which the examples are evaluated.

We also only consider **batch-mode** active learning, where a batch of examples are queried at once and added to the labelled dataset. Selecting a single example to be labelled and added to the training set at a time is not practical for classifiers like neural networks due to their long training time.

All active learning methods have their own measures to evaluate unlabelled data. In Chapter 3.4, we categorise the measures studied in the literature into three general categories: *informativeness*, *representativeness* and *diversity*. Informativeness-based methods select examples that are most likely to reduce the uncertainty or expected error of the model [11, 31, 51, 74, 75]. Representativeness-based methods select examples that are most representative of the data [29, 79, 102, 104]. Diversity-based methods attempt to select examples that are a diverse cover of the data [9, 101]. There are also many methods that attempt to combine informativeness with representativeness or diversity [49, 51, 61, 65, 77].

To the best of our knowledge, most literature on active learning evaluates algorithms in one of two ways. The first one is running active learning for multiple rounds and plotting the accuracy of the model trained after each round of active learning. The active learning method that provides the fastest improvement on accuracy is considered the winner. The second evaluation method only runs active learning once and tests the statistical difference among the competing algorithms. However, the number of initial labelled examples and the budget size are almost always arbitrarily chosen without any explanation. We think these types of evaluation methods have limitations. The second method only shows the results on a particular experimental setting. It does not inform us what would happen in other settings. The first one only tells us which method can improve the performance of a classifier the fastest from no or few labelled examples. However, one might still be interested in applying active learning when the labelled training set is large and the model's predictions are already accurate. For instance, even after a model is deployed in production a company would most likely still want to improve on it continuously. Since the model is good enough to be in production, it means there must be a big training set already. Active learning methods that accelerate the performance of a classifier the fastest when labelled dataset is small are not necessarily as effective when there is already a big labelled dataset available. In order to improve on a model that is already accurate, we usually have to find edge cases and those examples that are in the difficult regions of the data. We believe this particular setting of active learning has been overlooked by most active learning literature.

This chapter answers the following **research questions**:

1. How do different types of active learning methods perform under different settings on the number of initial labelled examples and the number of queried examples?

2. How can we apply recent theoretical and empirical studies on the convergence properties of neural networks to active learning?

In this chapter, we make the following **contributions**:

- We propose a new active learning framework for neural networks trained using stochastic gradient descent (SGD), that selects examples whose model outputs fluctuate the most during training. This framework assumes that the unlabelled examples whose model outputs fluctuate the most during training are more difficult to learn, and therefore are more useful to be added to the labelled set.

- We propose an implementation of the framework that queries unlabelled examples whose predictions change the most during training. The experimental results show that the proposed method is more effective when the initial labelled dataset is large.

- We empirically show that different types of active learning algorithms perform differently under different settings. This suggests that active learning algorithms should be evaluated under different settings in order to fully understand their characteristics.

The rest of this chapter is organised as follows. Section 6.2 discusses the motivation of our proposed active learning method. Section 6.3 introduces our proposed active learning framework that measures output fluctuation during training. Section 6.4 reports several implementations of the framework that did not outperform uniform sampling. Section 6.5 introduces our final proposed implementation of the framework. Section 6.6 describes the setup of the experiments. Section 6.7 analyses the experimental results. We discuss some interesting patterns in the results and limitations of our proposed method. Finally, we summarise this chapter in Section 6.9.

6.2 Motivation

Because our goal is to propose an active learning method that improves the generalisation of neural networks, it pays to draw some inspiration from studies on the generalisation of neural networks. Although researchers still do not have a consensus on how exactly neural networks generalise, trying to understand the generalisation properties of neural networks has been an active research area in recent years. Among the recently published interesting and promising theoretical works on neural networks, we are most interested in the studies on the convergence of neural networks trained using gradient descent or stochastic gradient descent [86, 110, 124]. Soundry et al. [110] and Nacson et al. [86] studied the convergence of linear fully-connected neural networks with exponential-tailed loss functions (a family of loss functions such as exponential, logistic and cross-entropy loss) trained using gradient descent on linearly separable problem. They found that under these conditions the classifier converges to the L_2 maximum-margin solution. The L_2 maximum-margin solution is equivalent to a hard-margin support vector machine classifier with a L_2 penalty on the parameters. It means that the linear fully-connected neural network behaves like a hard-margin support vector machine when trained under these conditions, and gradient descent has an implicit regularisation effect on the model. Xu et al. [124] studied the convergence of both gradient descent and stochastic gradient descent with non-linear ReLU networks on binary linearly separable problems. They found that due to the non-convex nature of the loss function for an ReLU network, the model is not guaranteed to converge. But when it does converge, it converges to either the global or local maximum-margin solution.

These theoretical results seem to be supported by some empirical works. Toneva et al. [117] empirically investigated the “example forgetting” phenomenon during training deep neural networks. They defined **unforgettable examples** as the examples that once learned (correct prediction) are never forgotten during the training. The examples that are forgotten (wrong prediction) after having been learned are

forgettable examples. They found that most of the examples in simple datasets are unforgettable examples (91.7% in MNIST [71]), while a much smaller portion of a hard dataset are unforgettable (31.3% in CIFAR-10 [66]). Furthermore, the authors tested the performance of a deep neural network trained on CIFAR-10 with unforgettable examples gradually being removed. They found that The model's performance did not suffer even after more than 30% of the data were removed from the training set (less than 0.2% drop in accuracy). On MNIST, the performance was maintained even after 80% of the training data were removed. These results suggest a lot of the examples (unforgettable examples) in the data are not useful for the generalisation of neural networks. Mostly the forgettable examples are useful for the generalisation of neural networks. The forgettable examples seem to play the role of support vectors as in support vector machine. This suggests that forgettable examples are likely to be near the target decision boundary of a dataset. Yaghoobzadeh et al. [125] applied these findings to increase the robustness of natural language understanding models in a natural language inference task by fine tuning the models with only forgettable examples. All of these empirical results are in support of the theoretical findings explained above.

Both the theoretical and empirical results make us question if active learning methods based on representativeness or diversity alone are really suitable for neural networks trained using gradient descent or its variants, especially when there is a large training set and the model already has a high test accuracy. Pure representativeness-based methods typically choose examples in the dense regions of the data while diversity-based methods select examples to cover the entire space of the data. These methods are expected to be more effective when labelled data is scarce and exploration of the space is more important. However, the effectiveness of these methods are expected to drop when the labelled dataset is large. This is because the exploration of the data space is less important when there is already a large labelled dataset. Furthermore, the decision boundary of the model trained on the large labelled dataset should be close to the target decision boundary. As explained before, the forgettable examples are likely to be around the target decision boundary. However methods based on representativeness or diversity would still choose examples that are far away from the decision boundary (unforgettable examples).

Uncertainty-based active learning methods like *margin* and *confidence* (defined in Chapter 3.4.2) choose examples close to the decision boundary of the current model. This means that their performance depends on the current model. If the current model is accurate, then the selected examples are close to the target decision boundary (forgettable examples). When the current model is less accurate (trained on a smaller labelled dataset), the selected examples are less likely to be forgettable examples. Because they only select examples around the decision boundary of the current model, uncertainty-based methods lack the ability to explore the data space. Therefore, we expect uncertainty-based active learning methods to be less effective when the labelled dataset is small.

In light of the theoretical and empirical findings discussed above, an ideal active learning method for neural networks trained using gradient descent or its variants should select instances from the forgettable regions of the data. Unfortunately, we usually cannot tell which examples are forgettable just by visualising the data. Even if it were possible, it would not be a feasible practice at a large scale. We have to find a way to estimate how likely an unlabelled example is to be forgettable. We propose to estimate the forgettable regions by measuring the **output fluctuation** of the unlabelled data during training. We propose a new active learning framework for measuring output fluctuation in Section 6.3. Section 6.4 reports several failed attempts at implementing the framework. In Section 6.5, we propose an implementation that measures the prediction changes of unlabelled data during training.

6.3 Framework for Measuring Output Fluctuation During Training

Inspired by the theoretical and empirical results described in Section 6.2, we propose a new active learning framework that queries the unlabelled examples whose model outputs fluctuate the most during training. The framework assumes that the examples whose model outputs fluctuate the most during training are more likely to be forgettable examples. This framework can be considered as an approximation of the method used in Toneva et al. [117] to find forgettable examples. All labels for a dataset are available in Toneva et al. [117], while we only have a small labelled dataset. We can only train a model on the small labelled dataset, therefore the learned model is less accurate. Toneva et al. [117] counts the number of times the model forgets the true label of an example during training. However, labels are not available to the unlabelled data in our learning scenario. In order to solve this problem, we approximate this by accumulating the distance between two consecutive model outputs on the same example after each epoch during training.

Algorithm 2 Framework of measuring output fluctuation for unlabelled data

```
Require: $L, U, epochs, skip$
 1: $fluctuation[j] \leftarrow 0, j \in U$
 2: $pre\_outputs[j] \leftarrow 0, j \in U$
 3: for $t \leftarrow 0$ to $epochs$ do
 4:     if $t > skip$ then
 5:         for example $j \in U$ do
 6:             compute $outputs[j]$
 7:             compute $distance_j \leftarrow distance(pre\_outputs[j], outputs[j])$
 8:             $fluctuation[j] \leftarrow fluctuation[j] + distance_j$
 9:             $pre\_outputs[j] \leftarrow outputs[j]$
10:         end for
11:     end if
12:     gradient update model on $L$
13: end for
```

The general framework is shown in Algorithm 2. It requires a labelled dataset L, an unlabelled dataset U, the number of *epochs* to train the model and a user specified parameter *skip*. Lines 1-2 initialise the fluctuation scores and previous outputs. For each epoch, we compute the current outputs for all the unlabelled data and calculate the distances between the previous outputs and the current outputs. Then the distances are accumulated for all the unlabelled data. Lastly, the model is updated on the labelled dataset L for one epoch. This process is repeated until the training is complete. We only accumulate the distances if the current epoch is larger than user specified *skip*. This is done to avoid the large swings in outputs during the first a few epochs caused by random initial weights of the model.

Algorithm 2 is only a framework of our proposed method. The specific implementation depends on the definition of **model output** and the choice of **distance function**. We will discuss some of the implementations we have tried in Section 6.4, before introducing our final proposed implementation in Section 6.5.

6.4 Preliminary Implementations

This section details the implementations we tried but which failed to outperform uniform sampling. We defined the model output for an unlabelled example j to be the probability output of the *softmax* output layer, P_j, or the input of the *softmax* layer, Z_j (introduced in Chapter 2.3). We experimented with both because the squashing effect of the *softmax* function affects the distance computation in a non-linear way.

The first distance function we tested was *squared L2 norm (squared Euclidean distance).* It is computed as follows:

$$squared\_L2(P_j^t, P_j^{t-1}) = \sum_{c=1}^{C}(p_{cj}^t - p_{cj}^{t-1})^2, or$$
$$squared\_L2(Z_j^t, Z_j^{t-1}) = \sum_{c=1}^{C}(z_{cj}^t - z_{cj}^{t-1})^2$$

where C is the number of classes, t is the current epoch and j is the index of the unlabelled example. We also experimented with *Kullback-Leibler divergence (KL-divergence)* and *symmetrised KL-divergence*:

$$kl\_div(P_j^t \parallel P_j^{t-1}) = \sum_{c=1}^{C} p_{cj}^t log(\frac{p_{cj}^t}{p_{cj}^{t-1}})$$
$$sym\_kl\_div(P_j^t \parallel P_j^{t-1}) = \sum_{c=1}^{C} p_{cj}^t log(\frac{p_{cj}^t}{p_{cj}^{t-1}}) + \sum_{c=1}^{C} p_{cj}^{t-1} log(\frac{p_{cj}^{t-1}}{p_{cj}^t})$$

where $kl\_div(P_j^t \parallel P_j^{t-1})$ measures the divergence of the current output P_j^t from the previous output P_j^{t-1}. Because KL-divergence is asymmetric, we also experimented with symmetrised KL-divergence. KL-divergence measures the divergence of a probability distribution from another reference probability distribution, therefore only the *softmax* outputs are used in the computation of KL-divergence and symmetrised KL-divergence.

All of these implementations did not outperform uniform sampling in our experiments. All of these measures of fluctuation have a common problem that they do not care about the direction of change in the output. We use Figure 6.1 to illustrate why this is a problem. There are two data points A and B, and two decision boundaries, classifier 1 and classifier 2, representing a classifier at different time stamps during training. Most of the methods mentioned above cannot distinguish between moving from classifier 1 to classifier 2 and moving from classifier 2 to classifier 1. In other words, the distance between the model outputs are the same regardless

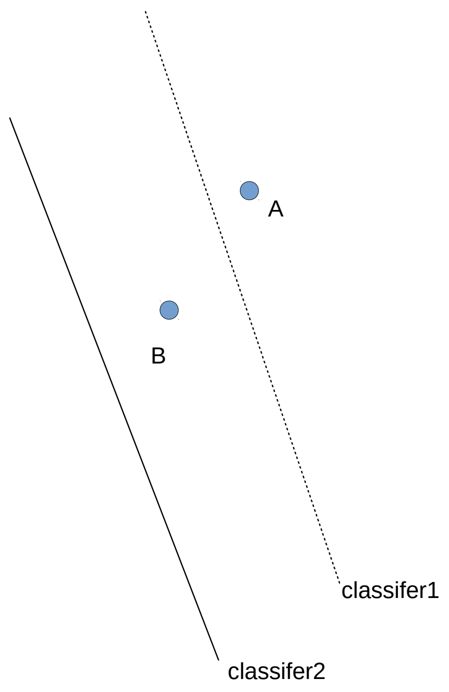


Figure 6.1: A and B are two different data points. The decision boundary of a classifier moves during training. The decision boundaries at different epochs are represented by classifier 1 (solid line) and classifier 2 (dashed line).

whether the classifier moved from classifier 1 to classifier 2 or from classifier 2 to classifier 1. However, if the classifier moved from classifier 1 to classifier 2 it is now more confident (confidence and other measures of uncertainty are defined in Section 3.4.2) about its prediction of example A, we therefore should reduce the likelihood of querying example A. We should increase the chance of querying A if the classifier moved from classifier 2 to classifier 1, because the model has become less confident about example A. Suppose the model has changed from classifier 1 to classifier 2. The change of model output for example A is likely to be larger than that for example B. This is because the model's confidence of example B has not changed much (although the prediction has changed), while it has changed a lot for example A. However, one should actually pick example B over example A, since the prediction of B has changed while the prediction for A has stayed the same and even has become more confident.

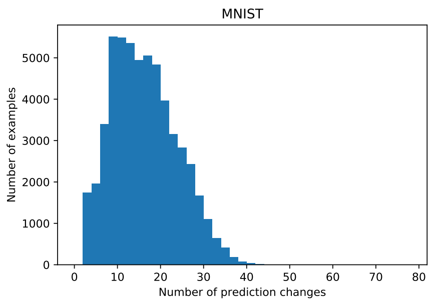
6.5 Prediction Fluctuation as Uncertainty Measure

This section introduces our proposed implementation of measuring output fluctuation. The failed attempts suggest that the direction of change in model outputs and whether it results in a change in prediction are really important in measuring output fluctuation. A simple and seemingly crude way to solve this problem is measuring the prediction fluctuation of unlabelled data during training. In measuring prediction fluctuation, we count the number of times the prediction of each unlabelled example changes during training. We essentially define model output as the prediction for an example. The distance function returns 1 if the current prediction changes from the last prediction, and returns 0 otherwise. We then query the examples with the highest number of prediction changes during training.

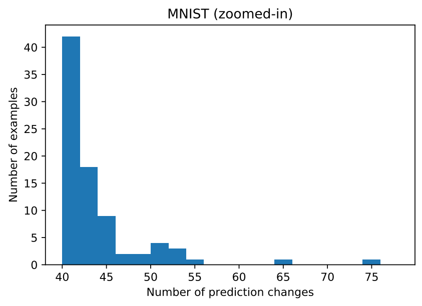
Algorithm 3 Measuring prediction fluctuation for unlabelled data

```
Require: $L, U, epochs, skip$
 1: $pred\_changes[j] \leftarrow 0, j \in U$
 2: $pre\_pred[j] \leftarrow 0, j \in U$
 3: for $t \leftarrow 0$ to $epochs$ do
 4:     if $t > skip$ then
 5:         for example $j \in U$ do
 6:             compute $\hat{y}_j$
 7:             if $\hat{y}_j \neq pre\_pred[j]$ then
 8:                 $pred\_changes[j] \leftarrow pred\_changes[j] + 1$
 9:             end if
10:             $pre\_pred[j] \leftarrow \hat{y}_j$
11:         end for
12:     end if
13:     gradient update model on $L$
14: end for
```

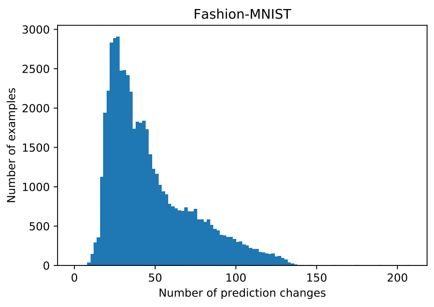
Measuring prediction fluctuation during training is related to ensemble or committee based active learning method. Each model after each epoch can be seen as a member of the ensemble. If the members of the "temporal ensemble" disagree with each other on the prediction of an example, the prediction fluctuation must also be high. However, prediction fluctuation is also different from ensemble based active learning method in that it takes into account the information about the sequence of predictions. Suppose there are two sequences of predictions with the same length: "1111122222" and "1212121212". Standard ensemble methods would consider the two sequences as exactly the same, even though the second sequence fluctuates much more. This is why we do not use prediction disagreement or vote entropy to compute prediction fluctuation.



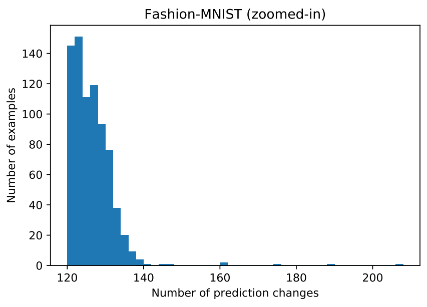
(a) MNIST



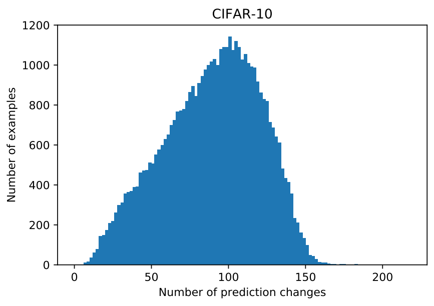
(b) MNIST (zoomed-in)



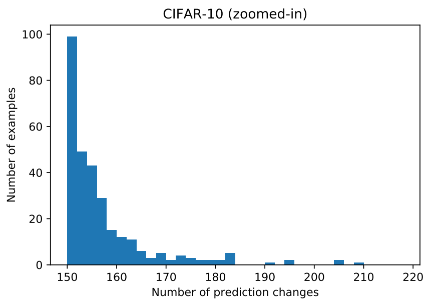
(c) Fashion-MNIST



(d) Fashion-MNIST (zoomed-in)



(e) CIFAR-10



(f) CIFAR-10 (zoomed-in)

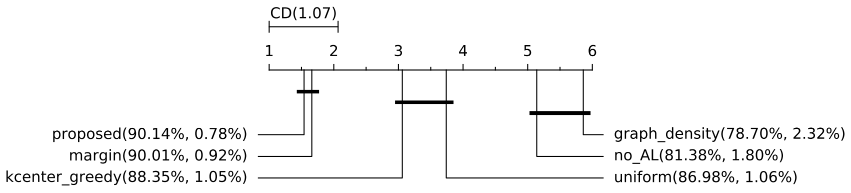
Figure 6.2: Plots of the distributions of the number of prediction changes occurred during training for MNIST, Fashion-MNIST and CIFAR-10. Pictures on the right show the zoomed-in portions of the plots. The classifiers were trained with 100 labelled examples for all three datasets.

The pseudo code for measuring prediction fluctuation for unlabelled data is shown in Algorithm 3. Lines 1-2 initialise containers used to accumulate prediction changes and store previous predictions. During training, the current predictions for unlabelled data are computed and compared to previous predictions. If the prediction for an unlabelled example has changed, we add 1 to the prediction change score for that example. Again, we skip the first few epochs before we start accumulating prediction changes. The process is repeated until the training is complete.

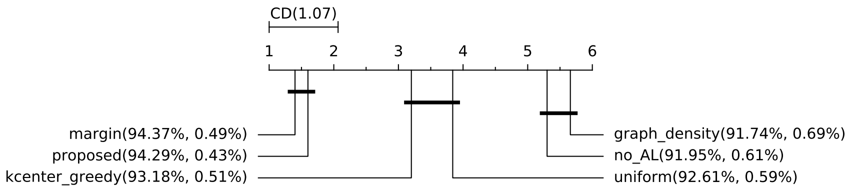
Figure 6.2 shows the distributions of the number of prediction changes which occurred during training for MNIST [71], Fashion-MNIST [123] and CIFAR-10 [66]. The plots are based on a single run. The models were trained using only 100 labelled examples for all three datasets. For a simple dataset like MNIST, the majority of the examples did not have many prediction changes during training. The examples in the slightly harder dataset, Fashion-MNIST, tend to have more prediction changes. For the hardest dataset out of the three, CIFAR-10, the distribution looks completely different. The distribution is more balanced and is skewed towards higher number of changes. This agrees with the findings in Toneva et al. [117]. The examples in the harder datasets tend to have more prediction changes during training, while the examples in the easier datasets have less prediction changes.

The proposed method of measuring prediction fluctuation is an uncertainty based active learning method. It defines uncertainty as the number of times the prediction of an unlabelled example changes during training. It tends to choose examples around the final decision boundary just like other uncertainty based methods like selecting examples with the least confidence or with the smallest margins between the highest probabilities and the second highest probabilities. Additionally, it also favours the examples that the model is likely to forget during training. This is beneficial because the examples that are closest to the decision boundary are not necessarily the most difficult to learn (the easiest to forget).

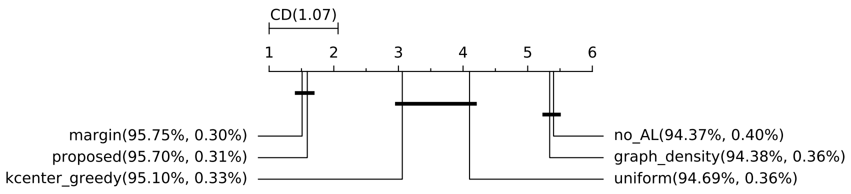
The performance of the proposed method depends on the available labelled training data. When the labelled dataset is really small, the proposed method would fail to explore all of the regions close to the target decision boundary and may lead to poor estimation of forgettable examples. Therefore, we expect the proposed method to be more effective when a large labelled dataset is available. Experiments and analyses have been conducted to evaluate the proposed method against different types of active learning methods. We describe the experimental setup in Section 6.6 and analyse the experimental results in Section 6.7.



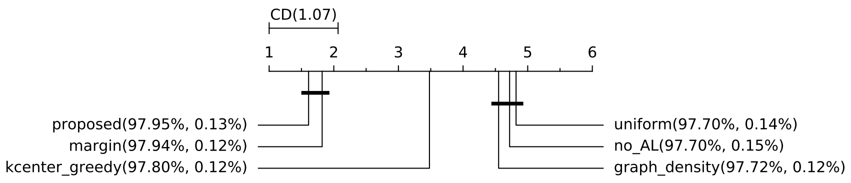
(a) 100 initial labelled examples



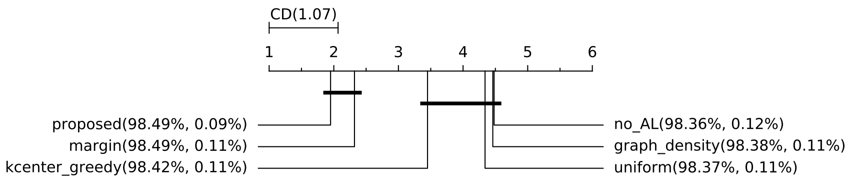
(b) 500 initial labelled examples



(c) 1000 initial labelled examples

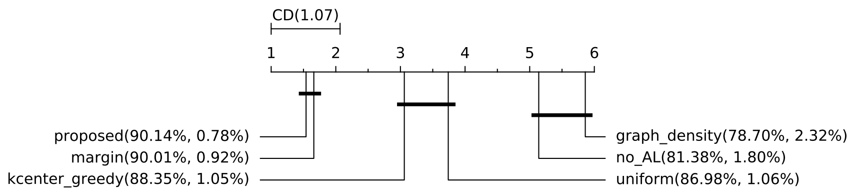


(d) 5000 initial labelled examples

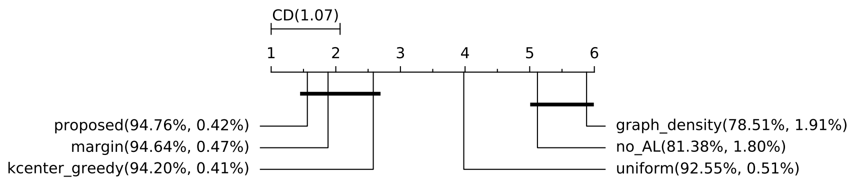


(e) 10000 initial labelled examples

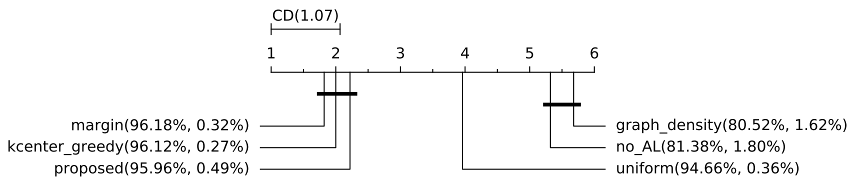
Figure 6.3: Comparison of active learning algorithms on MNIST. The number of queried examples is held constant at 100 while changing the number of initial labelled examples.



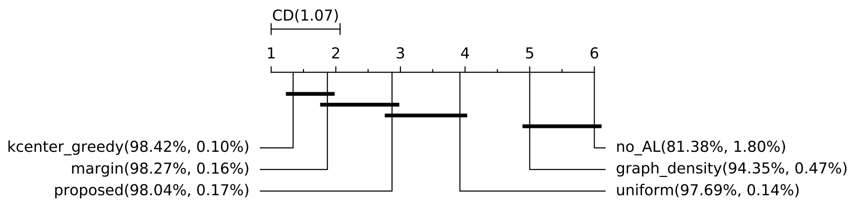
(a) 100 queried examples



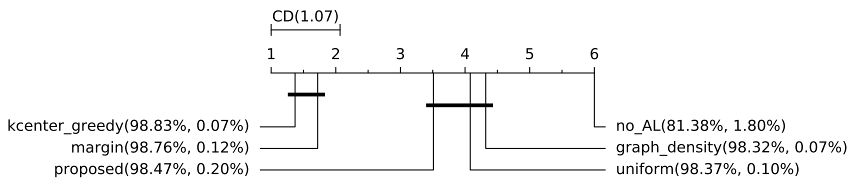
(b) 500 queried examples



(c) 1000 queried examples



(d) 5000 queried examples



(e) 10000 queried examples

Figure 6.4: Comparison of active learning algorithms on MNIST. The number of initial labelled examples is held constant at 100 while changing the number of queried examples.

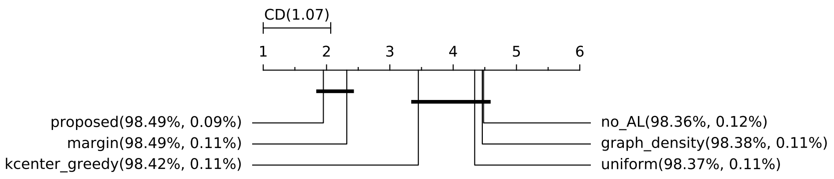
6.6 Experimental Setup

In this section, we describe the experiments conducted to evaluate the effectiveness of the proposed active learning method against various baseline algorithms. The experimental results are discussed in Section 6.7.

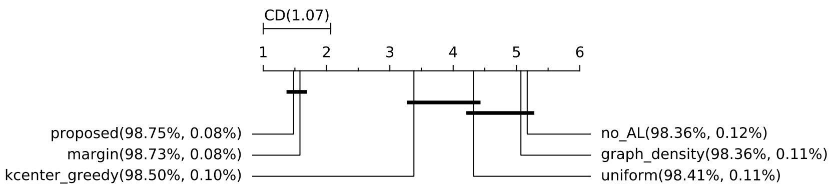
6.6.1 Baselines

We compared the proposed active learning method against baselines using different types of query strategies. The algorithms evaluated in our experiments are listed below. The details of the baseline algorithms are reviewed in Section 3.4.

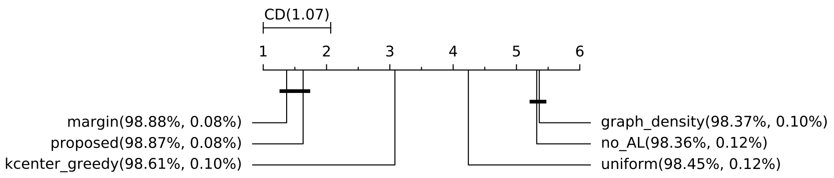
- *proposed.* This is our proposed uncertainty-based active learning method. It chooses the examples whose predictions change the most during training.
- *margin.* We used the margin algorithm as a representative for uncertainty-based query strategies. We chose margin over confidence as a measure of uncertainty, because Ienco et al. [51] found that margin-based uncertainty outperformed confidence-based uncertainty in most of their experiments. *Margin* computes the margin between the posterior probabilities of the two most likely classes and chooses the examples that have the smallest margins.
- *kcenter_greedy.* This is a diversity-based active learning method. We used the greedy version of the algorithm proposed by Sener and Savarese [101]. It greedily selects an example with the highest minimal distance to any other example.
- *graph_density.* This is a density-based method proposed by Ebert et al. [29]. It builds a k-nearest neighbour graph weighted by a Gaussian kernel and selects the examples that have many edges with high weights.
- *uniform.* This is a baseline that randomly selects unlabelled examples according to the uniform distribution.
- *no_AL.* We also added the results of pure supervised learning using the initial labelled training sets without any active learning in our evaluation. This helps determine if the additional data queried by an active learning algorithm actually improve the accuracy.



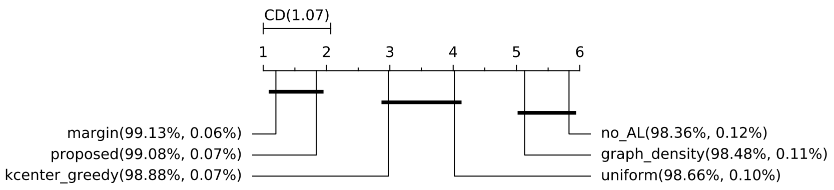
(a) 100 queried examples



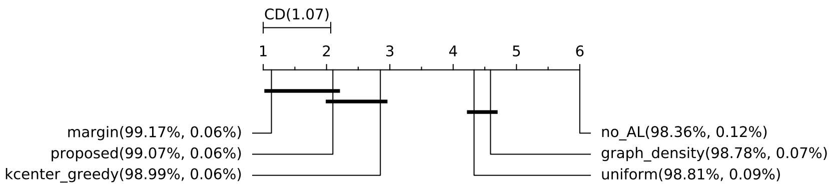
(b) 500 queried examples



(c) 1000 queried examples

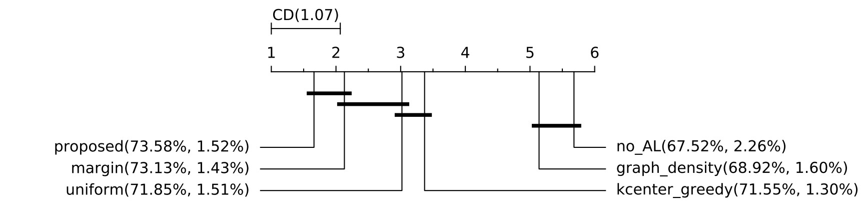


(d) 5000 queried examples

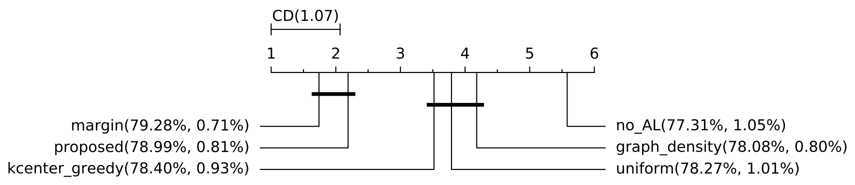


(e) 10000 queried examples

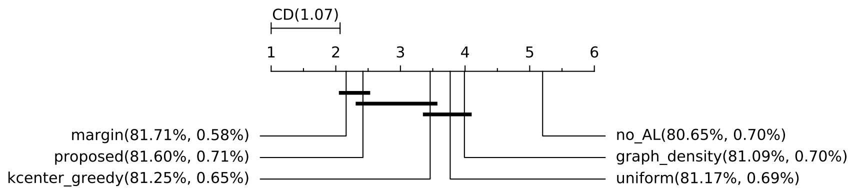
Figure 6.5: Comparison of active learning algorithms on MNIST. The number of initial labelled examples is held constant at 10000 while changing the number of queried examples.



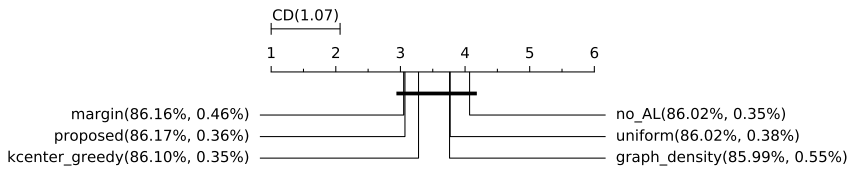
(a) 100 initial labelled examples



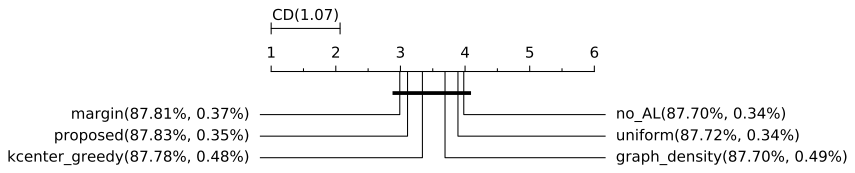
(b) 500 initial labelled examples



(c) 1000 initial labelled examples



(d) 5000 initial labelled examples



(e) 10000 initial labelled examples

Figure 6.6: Comparison of active learning algorithms on Fashion-MNIST. The number of queried examples is held constant at 100 while changing the number of initial labelled examples.

6.6.2 Datasets

We conducted experiments on the same image-recognition datasets used in Section 5.4: MNIST [71], Fashion-MNIST [123] and CIFAR-10 [66]. The basic characteristics of these datasets and the preprocessing are explained below.

- **MNIST** is a 28 × 28 grayscale image dataset of handwritten digits with 10 evenly distributed classes. The test set has 10,000 labelled images. We created a validation set of 5,000 labelled images used for hyperparameter tuning. The remaining 55,000 images were used for simulating active learning.

- **Fashion-MNIST** uses the same format and structure as MNIST, but it contains fashion items instead of handwritten digits. The ten classes are t-shirt/top, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. The test set contains 10,000 images. Again, we created a validation set containing 5,000 examples. This left us 55,000 images for training.

- **CIFAR-10** is a 32 × 32 RGB image dataset. There are 10 balanced classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. There are 10,000 test images. The validation set has 5,000 images. The remaining 45,000 images were used for training.

These datasets are originally intended for pure supervised learning. All of the images are labelled. However, active learning requires a pool of unlabelled data. We simulated this by splitting the training data into a labelled training set and a pool of unlabelled data by removing the labels for the remaining images. Finally, the pixel values of an image were normalised to the range of [0, 1].

6.6.3 Evaluation Method

As explained in Section 6.1, the common evaluation methods found in most active learning literature have their limitations. Some researchers report results after one round of active learning, but the number of initial labelled examples or the number of examples to query are arbitrarily chosen. Other researchers perform multiple rounds of active learning, and report the results after each round of active learning. The problem with this evaluation method is that there are two variables at each round of active learning: the algorithm and the labelled set. Even though this evaluation shows the rate of change in performance across multiple rounds of active learning, it does not tell us how competitive an algorithm is given different numbers of labelled examples. Additionally, researchers usually do not vary the number of initial labelled examples or the number of examples to sample. This means we have no idea how the algorithms would perform under other settings.

In order to avoid the limitations mentioned above and test the characteristics of the evaluated algorithms, we conducted three sets of experiments for each dataset. We only ran one round of active learning in our experiments but with different settings on the number of initial labelled examples and the number of examples to query. The details are explained below.

Varying Initial Labelled Examples

In the first set of experiments, we varied the number of initial labelled examples while holding the the number of examples to query constant. These experiments can tell us how competitive an algorithm is given different sizes of labelled training set.

For both MNIST and Fashion-MNIST, we held the number to query constant at 100, and set the number of initial labelled examples to 100, 500, 1000, 5000 and 10000. For CIFAR-10, we set the number to query constant at 500, and varied the number of initial labelled examples with 100, 500, 1000, 5000 and 10000. Because CIFAR-10 is a much harder dataset, a small number of additional training examples are unlikely to result in a significant improvement in accuracy. Note that all the remaining examples in the training set were treated as unlabelled data by removing their labels.

Varying Number to Query When Labelled Set Is Small

In the second set of experiments, we set the number of labelled examples at 100 and changed the number to query with the values of 100, 500, 1000, 5000 and 10000. These experiments can tell us how the performance of an algorithm changes with the number to query when the labelled set is small.

Varying Number to Query When Labelled Set Is Big

In the third set of experiments, we set the number of labelled examples at 10000 and varied the number of queried examples with the values of 100, 500, 1000, 5000 and 10000. These experiments tell us how the performance of an algorithm changes with the number to query when the labelled set is big.

These experiments make sure that the only variable in comparing the algorithms in a specific setting is the algorithms themselves. The experimental results should reveal much more information about the evaluated algorithms than the common evaluation methods. For MNIST and Fashion-MNIST, each experiment was repeated 50 times with different seeds. Each experiment was run 20 times for CIFAR-10, because of the much longer training time. The seed changes the split of labelled set and unlabelled set.

6.6.4 Implementation Detail

We used the same architectures defined in Table 5.1 and Table 5.2 for MNIST, Fashion-MNIST and CIFAR-10 respectively. We used the vanilla stochastic gradient descent with a constant learning rate in training. Both *kcenter_greedy* and *graph_density* require distance computation. We used Variational Autoencoder (VAE) [58] to reduce the dimensionality of the images just like we did in Section 5.4.2. For MNIST and Fashion-MNIST, we used a simple multi-layer perceptron (MLP) based encoder and decoder as proposed in Kingma and Welling [58]. The dimensionality of the hidden representation was set to 10. For CIFAR-10, we used a convolutional encoder and transposed convolution for up-sampling in the decoder. The dimensionality of the vector representation for CIFAR-10 was set to 20. The validation sets were used to tune the hyperparameters. All the experiments were implemented in Pytorch [89]. The hyperparameters used in the experiments are reported in Appendix C. The implementation of the proposed method can be found on GitHub: https://github.com/superRookie007/pred-change-al.

6.7 Experimental Results

In this section, we discuss the experimental results of the experiments described in the previous section. The results are presented in CD (Critical Difference) graphs [1]. A CD graph is a graphical presentation of the Nemenyi test [22]. Nemenyi test is a post-hoc rank-based statistical test that compares multiple algorithms at the same time. We used 0.05 as the confidence level. The CD graph shows the average

[1]All of the results are also reported in the tables in Appendix D

ranking of each algorithm. The lower the ranking the better. The difference in average ranking is statistically significant if there is no bold line connecting the two algorithms. The mean and standard deviation of test accuracy are shown in parenthesis. The test accuracy was computed after training the model with the additional queried examples, apart from *no_AL* (no active learning was applied). For instance, if the initial number of labelled examples is 100 and the number of queried examples is 100, then the test accuracy is obtained from the model that is trained on both the initial labelled data and the queried data (200 labelled examples in total). The test accuracy for *no_AL* is computed from the model only trained on the 100 initial labelled examples. We will analyse the experimental results for each dataset separately.

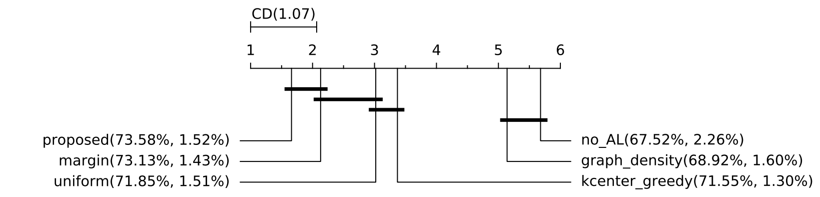
6.7.1 MNIST

We discuss the experimental results on MNIST for each experiment in this section.

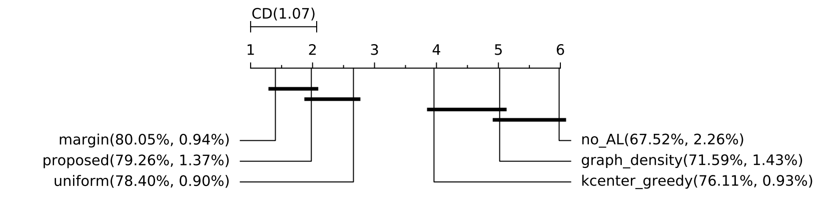
Varying Initial Labelled Examples

Figure 6.3 shows the results of the experiments on MNIST that varied the number of initial labelled examples while holding the queried examples constant at 100. Both uncertainty-based methods, *margin* and *proposed*, outperformed the other methods significantly in all cases. We had expected *kcenter_greedy*, *graph_density* and *uniform* to perform better when the initial labelled set is small, because these methods choose examples to be diverse or representative of the data. They provide better exploration of the data when labelled training set is small. One possible explanation of this observation is that the MNIST dataset is really simple; even 100 labelled examples are enough to train model with high accuracy. Uncertainty-based active learning methods lack the ability of exploring the entire data space, because they tend to choose examples close to the current decision boundary. However, exploration is not as important when the initial dataset is large or the current model is already accurate.

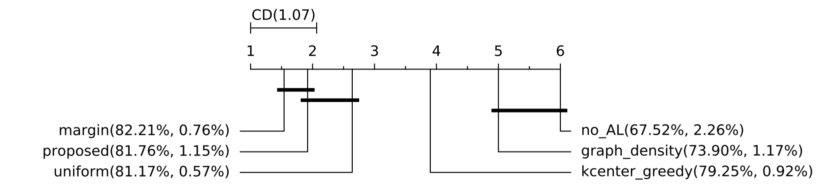
The additional labelled examples queried by both *kcenter_greedy* and *uniform* significantly outperformed *no_AL* when the number of initial labelled examples was small. However, as the size of the initial labelled set increased, the differences disappeared. This behaviour was expected, because data exploration provides less value when the initial labelled dataset is already big. The method *graph_density* failed to improve the model trained on the initial labelled data in all cases with the settings in this set of experiments.



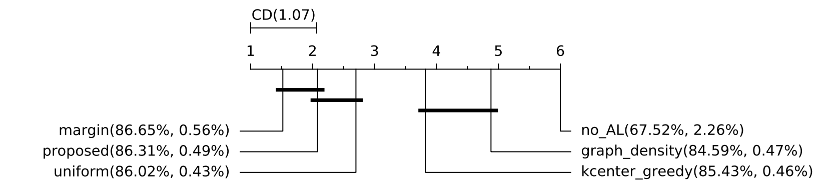
(a) 100 queried examples



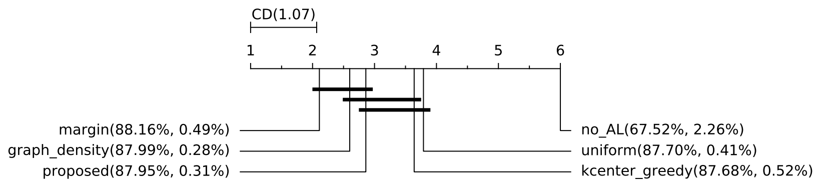
(b) 500 queried examples



(c) 1000 queried examples

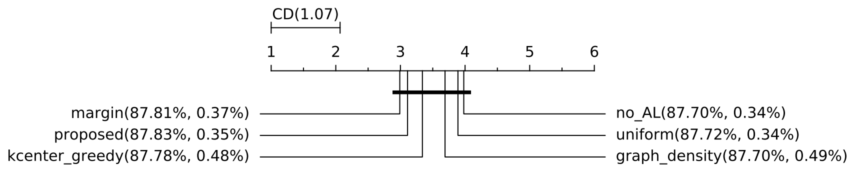


(d) 5000 queried examples

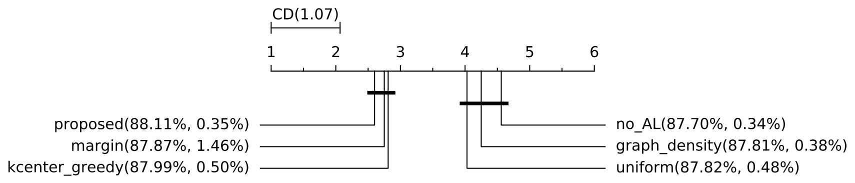


(e) 10000 queried examples

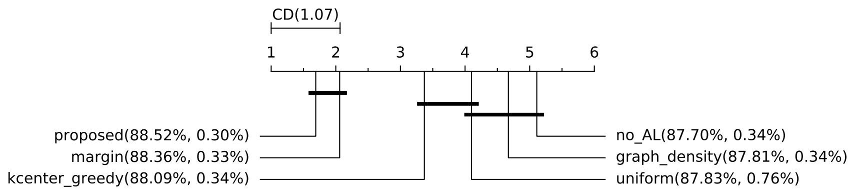
Figure 6.7: Comparison of active learning algorithms on Fashion-MNIST. The number of initial labelled examples is held constant at 100 while changing the number of queried examples.



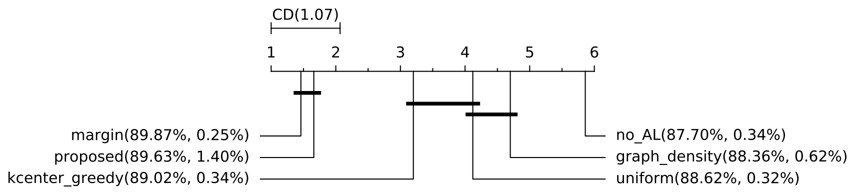
(a) 100 queried examples



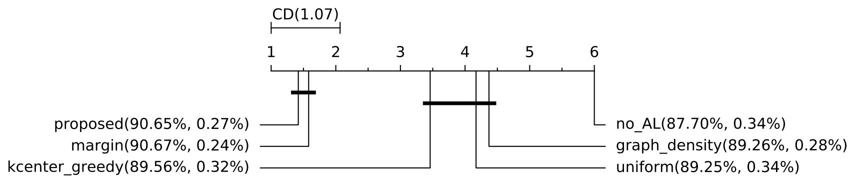
(b) 500 queried examples



(c) 1000 queried examples



(d) 5000 queried examples



(e) 10000 queried examples

Figure 6.8: Comparison of active learning algorithms on Fashion-MNIST. The number of initial labelled examples is held constant at 10000 while changing the number of queried examples.

Varying Number to Query When Labelled Set Is Small

The experimental results on MNIST with different numbers of queried examples while holding the number of initial labelled examples constant at 100 are shown in Figure 6.4. The uncertainty-based methods, *proposed* and *margin* outperformed the other methods significantly when only 100 examples were queried. However, the performance of the diversity-based method *kcenter_greedy* improved as the number of queried examples increased, while the advantage of *proposed* disappeared. The method *graph_density* did not provide improvement on models trained on the initial labelled sets when the number of labelled examples was small. The method *margin* was always among the group of the most effective methods.

Varying Number to Query When Labelled Set Is Big

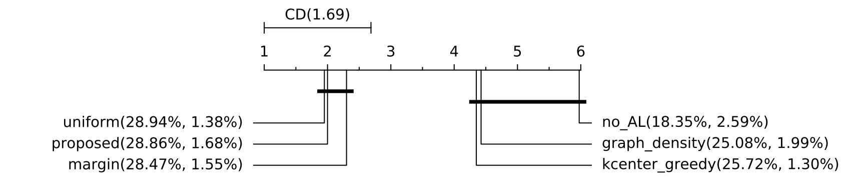
Figure 6.5 shows the experimental results on MNIST with different numbers of queried examples when the number of initial labelled examples was 10000. The average ranking of *proposed* was the best when the number of queried example was 100 and 500. The average rankings of *proposed* and *margin* swapped when the number of queried examples was large. However, there was no statistically significant difference between *proposed* and *margin* in all cases. When 10000 examples were queried, the advantage of *proposed* over *kcenter_greedy* became insignificant. Methods *uniform* and *graph_density* only improved the test accuracy of the model trained on the initial labelled set when a large amount of examples were queried.

6.7.2 Fashion-MNIST

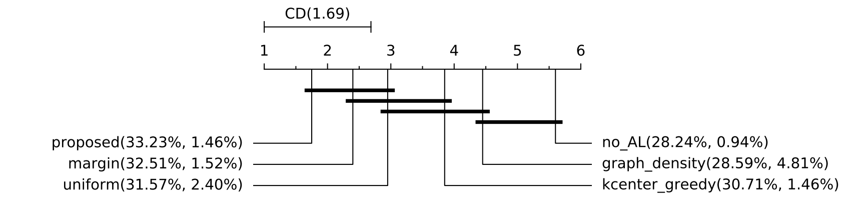
This section analyses the results on Fashion-MNIST for each experiment.

Varying Initial Labelled Examples

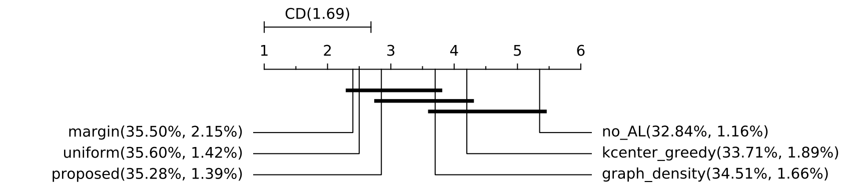
Figure 6.6 shows the results on Fashion-MNIST with different numbers of initial labelled examples when the number of queried examples was 100. Methods *proposed* and *margin* were among the best performing methods when the number of initial labelled examples was small. The advantage of these methods disappeared as the number of initial labelled examples increased. There was no significant difference among all methods when the initial labelled set was big (5000 or 10000 initial labelled examples). Furthermore, no method managed to significantly outperform the model trained on 5000 or 10000 initial labelled examples.



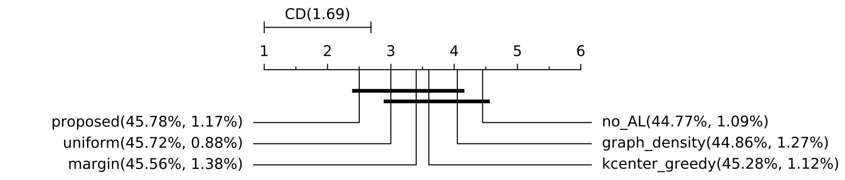
(a) 100 initial labelled examples



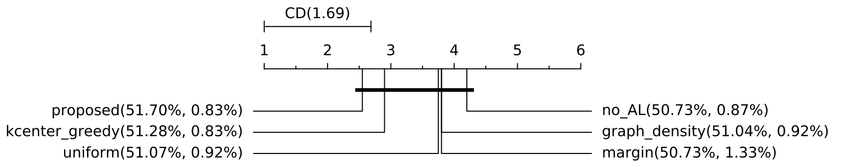
(b) 500 initial labelled examples



(c) 1000 initial labelled examples

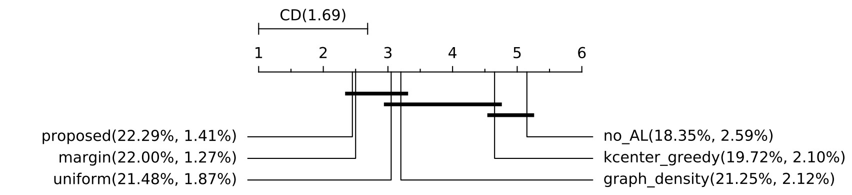


(d) 5000 initial labelled examples

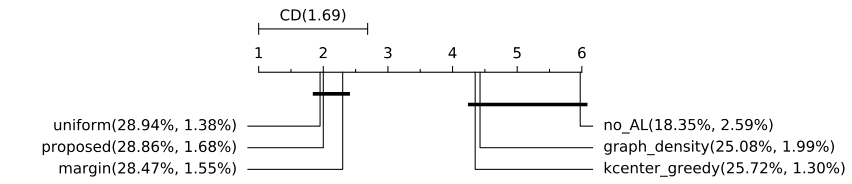


(e) 10000 initial labelled examples

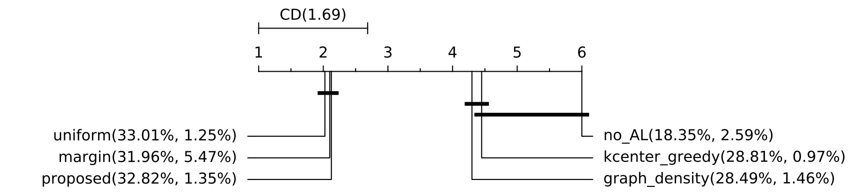
Figure 6.9: Comparison of active learning algorithms on CIFAR-10. The number of queried examples is held constant at 500 while changing the number of initial labelled examples.



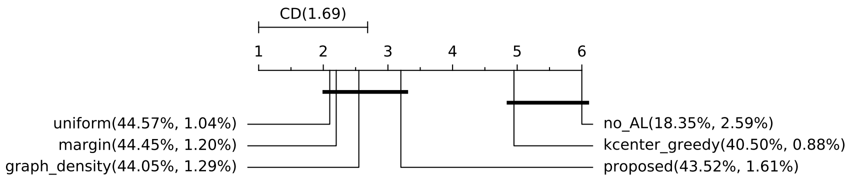
(a) 100 queried examples



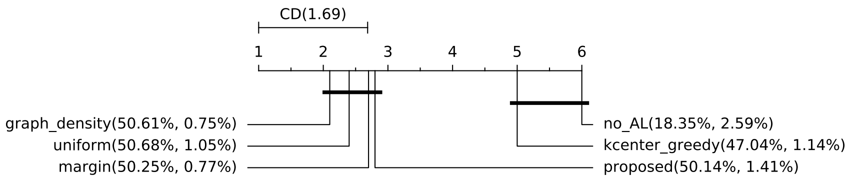
(b) 500 queried examples



(c) 1000 queried examples



(d) 5000 queried examples



(e) 10000 queried examples

Figure 6.10: Comparison of active learning algorithms on CIFAR-10. The number of initial labelled examples is held constant at 100 while changing the number of queried examples.

Varying Number to Query When Labelled Set Is Small

Figure 6.7 presents the experimental results on Fashion-MNIST with different number of queried examples when the number of initial labelled examples was 100. The *proposed* method had the best ranking when only 100 examples were sampled even though the difference between *proposed* and *margin* was not significant. Interestingly, even though there was no significant difference between *graph_density* and *no_AL* when the number of queried examples was small, *graph_density* was among the most effective methods when 10000 examples were queried.

Varying Number to Query When Labelled Set Is Big

Figure 6.8 shows the experimental results on Fashion-MNIST with different number of queried examples when the number of initial labelled examples was 10000. No method significantly outperformed the model trained on the initial labelled set when only 100 examples were queried. Both *proposed* and *margin* outperformed the other methods when the number of queried examples was 1000 and higher. As the number of queried examples increased, all the other active learning methods also started to provide significant improvement over the model trained on the initial labelled set.

6.7.3 CIFAR-10

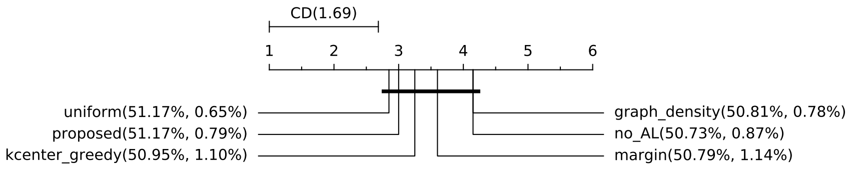
This section analyses the results on CIFAR-10 for each experiment.

Varying Initial Labelled Examples

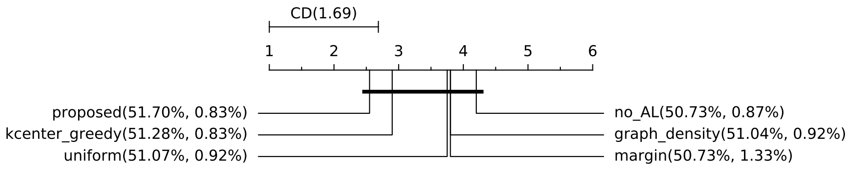
Figure 6.9 shows the results on CIFAR-10 with different number of initial labelled examples while the number of queried examples was 500. Methods *uniform*, *proposed* and *margin* significantly outperformed the other methods when the initial labelled set only had 100 examples. As the number of initial labelled examples increased, the differences among all the methods gradually shrank. Eventually, no method was significantly better than any other method when the number of initial labelled examples was 10000.

Varying Number to Query When Labelled Set Is Small

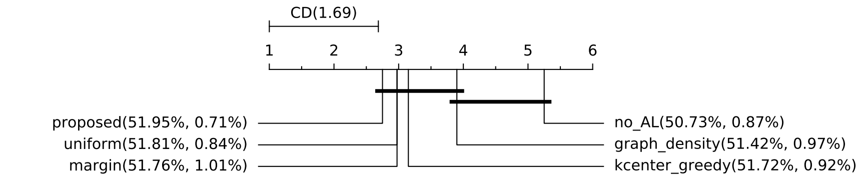
The results on CIFAR-10 with different number of queried examples when the initial number of labelled examples was 100 were shown in Figure 6.10. Methods *proposed*,



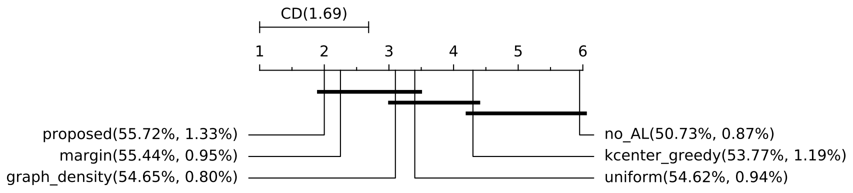
(a) 100 queried examples



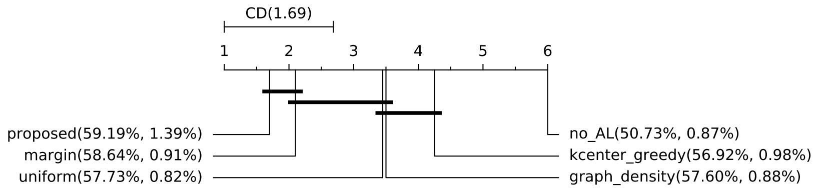
(b) 500 queried examples



(c) 1000 queried examples



(d) 5000 queried examples



(e) 10000 queried examples

Figure 6.11: Comparison of active learning algorithms on CIFAR-10. The number of initial labelled examples is held constant at 10000 while changing the number of queried examples.

margin and *uniform* were always in the group of best performing methods. The average ranking of *proposed* dropped gradually as the number of queried examples increased. The performance of *graph_density* improved when the number of queried examples was large. When 10000 examples were queried, *graph_density* had the best average ranking.

The CD graphs show that *Kcenter_greedy* was never significantly different from *no_AL* in all cases in terms of **average ranking** across 20 runs. Note that this does not necessarily mean *kcenter_greedy* did not significantly improve the **test accuracy** over *no_AL*. In fact, *kcenter_greedy* did improve the accuracy significantly when the number of queried examples was large. For instance, the p-value for paired t-test on *no_AL* and *kcenter_greedy* is 1.69e-18 when 5000 examples are queried. The p-value is 5.89e-21 when 10000 examples are queried. The Nemenyi test is a rank-based test. If the average ranking between two algorithms is smaller than the critical difference (CD), the Nemenyi test considers the difference between the two algorithms insignificant. In this case, *kcenter_greedy* was consistently outperformed by other active learning methods, its average ranking was never significantly difference from that of *no_AL* (ranked 6th).

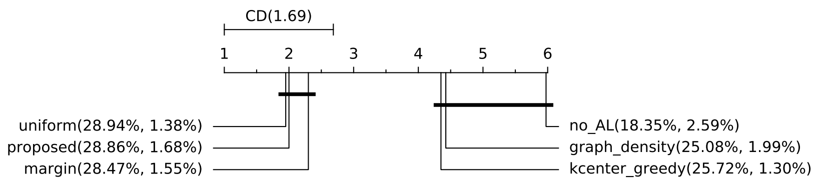
Varying Number to Query When Labelled Set Is Big

Figure 6.11 shows the experimental results on CIFAR-10 with different number of queried examples when the number of initial labelled examples was 10000. There was no significant difference among all the methods when only 100 or 500 examples were queried. As the number of queried examples increased, *proposed* and *margin* gradually pulled ahead from the rest of the methods. When 10000 examples were queried, *proposed* was significantly better than all the other methods except *margin*.

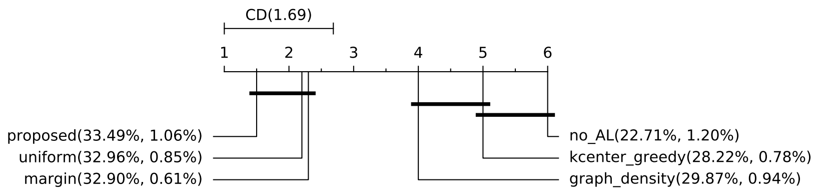
6.8 Discussion

We observed some interesting and consistent patterns from the experimental results on three different datasets. When the number of queried examples is held constant, as the size of the initial labelled set increases the performances of all algorithms gradually converge together. This is because if the labelled training examples are abundant additional labelled examples are unlikely to improve the results further. The methods based on diversity and density such as *kcenter_greedy* and *graph_density* are more effective when the number of initial labelled examples is small and the number of queried examples is large. However, they may not be both effective on the same dataset. For instance, *kcenter_greedy* performs better than *graph_density* in most cases on MNIST. However, *graph_density* outperforms *kcenter_greedy* on CIFAR-10 when the number of initial labelled examples is 100 and the number of queried examples is over 1000. The uncertainty-based methods (*margin* and *proposed*) are most effective when the initial labelled set is big. Overall, the results show that different types of query strategies are suitable for different active learning settings. This suggests that active learning algorithms should be evaluated under different settings. Otherwise the evaluation cannot show the full characteristics of the algorithms. Practitioners would have no idea if a winning algorithm in a particular setting would also be effective in their use cases.

The algorithms *kcenter_greedy* and *graph_density* were not as effective at improving the generalisation performance as we expected, especially when the labelled set was small. They both require distance computation. Hence, the features or encodings used for computing the distance can have a big influence on the effectiveness of these algorithms. It is possible that by using more advanced dimensionality reduction techniques or learning better encodings once can improve their performances.



(a) Architecture used in the experiments (a variant of LeNet)



(b) ResNet18

Figure 6.12: Comparison of the results using ResNet18 and the original architecture used in the experiments on CIFAR-10. The number of initial labelled examples is 100 and the number of queried examples is 500.

We did not use a state-of-the-art architecture (architecture used is defined in Table 5.2) on CIFAR-10 in order to reduce the computational cost. It was not our goal to achieve state-of-the-art results on certain datasets. We only wanted to keep all factors consistent for all algorithms in the evaluation. In order to make sure the choice of architecture would not have a big impact on the experimental results, we compared the architecture used in our experiments against ResNet18 [44] on CIFAR-10 when the number of initial labelled examples was 100 and the number of queried examples is 500. The results are shown in Figure 6.12. The test accuracies have increased for all algorithms but the statistical test results are similar to before. The only major difference is *graph_density* becomes statistically significantly better than *no_AL* when ResNet18 is used. We expect this to be consistent in other settings. The test accuracies would increase but the statistical test results would not change drastically.

A limitation of the proposed active learning method is its computation complexity: $O(|U| * epochs * \Upsilon)$, where $|U|$ is the number of unlabelled examples, *epochs* is the number of epochs and Υ is the computation cost for making one prediction. Data parallelism with multiple machines and GPU acceleration can reduce the running time on really large unlabelled dataset, however, it would require expensive computing equipment. In comparison, the *margin* active learning method has a computational complexity of $O(|U| * \Upsilon)$.

Another potential drawback of the proposed method is that it might select examples in the regions of the data with high Bayes error rates. Bayes error rate is the theoretical optimal error rate achievable for any classifier on a given classification problem; it is the irreducible error of a classification problem [32, 118, 119]. The examples in the region of high Bayes error rate are likely to be difficult to learn and are forgettable. However, it is unlikely to improve the generalisation performance of a model by adding these examples to the labelled dataset.

6.9 Summary

In this chapter we have proposed a new uncertainty-based active learning method that queries the examples with the highest prediction changes during training. The proposed method is motivated by the theoretical studies that show neural networks tend to converge to maximum-margin solutions and the empirical observation that the performance of a neural network is mainly determined by the forgettable examples in a dataset. The experimental results showed that the proposed method and the margin-based algorithm were more effective compared to the diversity-based algorithm (*kcenter_greedy*) and the density-based algorithm (*graph_density*) when the initial labelled dataset was large.

Our experiments revealed that different active learning algorithms prefer different active learning settings. A winning algorithm in one setting may not be as effective on the same data but with a different setting. This strongly suggests that active learning algorithms should be evaluated in a wide range of settings. A more comprehensive evaluation may also motivate the active learning community to design algorithms that are optimised for specific settings.

7 Conclusion

This thesis focuses on improving the generalisation of neural networks using unlabelled data. In particular, we study three different strategies of utilising unlabelled data in order to achieve this goal: pretraining, semi-supervised learning and active learning. In this chapter, we conclude this thesis by detailing our achievements, discussing the limitations of the thesis and proposing potential future works.

7.1 Achievements and Contributions

The following list highlights the major achievements of this thesis:

Chapter 4

- We empirically showed that the initial weights have an impact on the generalisation of neural networks. Ill-initialised neural networks can lead to lower test accuracy even though there is no instability issue during training.
- We proposed a supervised pretraining method that automatically creates a labelled training set for the pretraining task from unlabelled data. The pre-

training task trains a model to identify the real data from the randomly shuffled data. The learned weights are then reused as initial weights and fine-tuned on the labelled dataset. The experimental results show that the proposed pretraining method can improve the generalisation performance, especially when the labelled dataset is small. The experimental results on synthetic data suggest that this supervised pretraining works best on datasets with higher dimensionality and noisy features.

Chapter 5

- We broke the common assumption in semi-supervised learning that the labelled data and unlabelled data come from the same distribution. We empirically showed that the presence of novel classes in the unlabelled data can degrade the generalisation performance of semi-supervised algorithms for neural networks.

- We proposed a general distance-based weighting framework that assigns weights to unlabelled data according to how far away they are from the labelled data. The framework assumes that the unlabelled data that are further away from the labelled data are more likely to belong to the novel classes, and therefore should be assigned smaller weights during training. The proposed framework can be applied to any semi-supervised learning algorithms that optimise a combined loss of a supervised loss function (with labels as an input) on the labelled data and an unsupervised loss function (without using labels) on the unlabelled data.

- We proposed a 1-nearest-neighbour based implementation of the proposed framework. It defines the distance between an unlabelled example and the labelled data as the smallest distance between this unlabelled example and any example in the labelled dataset. The experimental results show that when the proposed method is applied to Pseudo-Label [73] and Mean Teacher [116], the degradation in the generalisation performance due to the presence of novel classes becomes statistically insignificant. This indicates that assigning weights to unlabelled data based on the distances between them and the labelled data is a promising approach in dealing with the problem of novel classes in semi-supervised learning.

Chapter 6

- We proposed a new uncertainty-based active learning framework for neural networks that queries examples whose model outputs fluctuate the most during training. This framework assumes that the unlabelled examples whose model outputs fluctuate the most are more likely to be near the decision boundary and are also more difficult to learn. Therefore, adding these examples to the labelled dataset are more likely to be useful for improving the generalisation of the model.

- We proposed a specific implementation of the framework that selects examples whose predictions change the most during the training. It is an approximation of the method used in Toneva et al. [117] to compute "forgettable" examples in supervised learning. The experimental results suggest that the proposed method is most effective when the initial labelled training set is large. Most of the literature on active learning focus on improving the performance of a classifier when the initial labelled dataset is small. However, we think the setting where a large labelled dataset is already available should not be overlooked. Practitioners usually want to continuously improve the generalisation performance of a classifier even after the classifier is deployed.

- Our experiments also show that different types of active learning methods perform differently under different settings with different number of initial labelled examples and number of queried examples. It suggests that to fully evaluate the characteristics of an active learning algorithm, experiments under a wide range of settings are required. A more comprehensive evaluation method can also potentially motivate researchers to come up with algorithms optimised for specific settings. Furthermore, it helps practitioners choose the most suitable algorithm given a specific setting.

7.2 Limitations

There are some limitations of the methods proposed in this thesis. We discuss them below.

- Although our proposed supervised pretraining method has been shown to work really well on certain datasets, we do not know exactly when and why it would be most effective. The proposed method also cannot be applied directly to other types of data such as images and texts. This seems to be a common limitation of self-supervised pretraining. The pretext task used in the pretraining is usually specific to the type of data or task at hand.

- The method we proposed to mitigate the effect of novel classes in semi-supervised learning requires distance computation between the unlabelled data and the labelled data. This means the effectiveness depends on the features used to compute the distance. Distance computation can be a problem if the dimensionality of the data is large due to "curse of dimensionality". To solve this, a dimensionality reduction technique has to be applied. The effectiveness also depends on the size of the labelled dataset. The larger the labelled dataset is, the more accurate we can compute the similarity between the labelled dataset and any unlabelled example.

- A potential limitation of the proposed active learning method is that it might choose examples in the regions of high Bayes error rates [32, 118, 119]. Bayes error rate is the theoretical lowest error rate achievable by any classifier on a given classification problem. Even though the examples in the region of high Bayes error rate are difficult to learn, adding them to the labelled dataset is unlikely to improve the test error significantly.

- Finally, a limitation shared by almost all literature on machine learning with no or small amount of labelled data is hyperparameter tuning. We categorise hyperparameters into two categories: *explicit* and *implicit*. An explicit hyperparameter is a hyperparameter required explicitly by an algorithm and has to be specified by an user. An example of explicit hyperparameter is learning rate. An implicit hyperparameter is implicit in nature. For instance, the *margin* active learning method does not have any explicit hyperparameters. However, its performance depends on the learned model. The model itself is an implicit hyperparameter. This means that the choice of architecture, learning rate and even the number of epochs can all have an impact on its performance. Like most other literature, we tuned the hyperparameters in our experiments using a validation set. However, a large validation set is often not available in practice. A set of hyperparameters that are optimised for a small validation set may not be as effective on future unseen data. There is no clear solution to this problem when the labelled dataset is really small. This problem deserves more attention and research efforts from the research community.

7.3 Future Directions

Here, we identify and propose potential future directions based on the research reported in this thesis.

Data agnostic pretraining

Most of the current self-supervised pretraining methods are data specific in that the pretext task can only be used in a specific type of data or task (for instance, images for image recognition). Our proposed pretraining method currently only works on tabular data. In future works, we would like to propose a pretraining framework that can be applied to different types of data and tasks.

Investigate further on novel classes in semi-supervised learning

An important work in the future is to investigate when novel classes can have a larger impact on the performance of semi-supervised learning algorithms. Synthetic data can be useful for this type of work, because we can control the properties of the data. It is also worth exploring other methods in order to deal with novel classes in semi-supervised learning. Existing outlier detection and novelty detection techniques can be potentially applied to deal with novel classes in unlabelled data.

Active learning designed for semi-supervised learning algorithms

Currently, the semi-supervised learning literature and active learning literature are mostly independent from each other. Chapter 3.3 introduces three different assumptions applied in common semi-supervised learning algorithms. Can we design an active learning algorithm for a specific semi-supervised learning algorithm by exploiting its assumptions? It is possible that this purposely designed active learning algorithm is more effective when used with the semi-supervised algorithm than simply running an off-the-shelf active learning algorithm on top of the semi-supervised algorithm.

Unbalanced class distribution

Like most other literature on semi-supervised learning and active learning, the datasets in our experiments have balanced class distributions. Considering that datasets used in practice do not always have balanced class distributions, it is worth investigating what would happen if the datasets are extremely unbalanced.

Alternative definition and measurement of generalisation

This thesis studies methods that improve the generalisation of neural networks using unlabelled data. We define generalisation as the ability to make accurate predictions on future unseen data. The generalisation is measured as test accuracy. However, there might be other types of generalisation that are worth studying. Guo et al. [41] observed that modern neural networks are poorly *calibrated* even though they have high test accuracy. A model is poorly calibrated if its output probabilities do not reflect the ground truth correctness likelihood. This can be a problem if the output probability is used as a measure of the confidence of the model on a prediction. If a model is poorly calibrated, we cannot trust its output probabilities even though the model has a high test accuracy. Guo et al. [41] claim that although modern neural networks have high test accuracies, they are actually overfitting the training data in terms of output probabilities. In other words, even if a model can generalise well if we measure generalisation using the accuracy of predictions, but at the same time this model might generalise poorly if we measure generalisation with the accuracy of output probabilities. Therefore, a potential research direction that is worth exploring is using unlabelled data to improve the calibration of neural networks.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5:4308, 2014.

[4] Eric B Baum and Kenneth Lang. Query learning can work poorly when a human oracle is used. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 335–340, 1992.

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

[6] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learn-

ing with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.

[7] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 5049–5059, 2019.

[8] Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

[9] Erdem Bıyık, Kenneth Wang, Nima Anari, and Dorsa Sadigh. Batch active learning using determinantal point processes. *arXiv preprint arXiv:1906.07975*, 2019.

[10] Michael C Burl and Esther Wang. Active learning for directed exploration of complex systems. In *Proceedings of the 26th International Conference on Machine Learning*, pages 89–96, 2009.

[11] Colin Campbell, Nello Cristianini, Alex Smola, et al. Query learning with large margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 111–118, 2000.

[12] Olivier Chapelle. Active learning for parzen window classifier. In *AISTATS*, volume 5, pages 49–56. Citeseer, 2005.

[13] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-supervised learning*. Cambridge: The MIT Press, 2006.

[14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[15] François Chollet et al. Keras. `https://keras.io`, 2015.

[16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[17] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[18] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[19] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 746–751. AAAI Press, 2005.

[20] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the International Conference on Machine Learning*, pages 150–157. Morgan Kaufmann, 1995.

[21] Sanjoy Dasgupta, Daniel J Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *Advances in Neural Information Processing Systems*, pages 353–360, 2008.

[22] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30, 2006.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[24] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[25] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[26] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[27] 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.

[28] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[29] Sandra Ebert, Mario Fritz, and Bernt Schiele. Ralf: A reinforced active learning formulation for object class recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633. IEEE, 2012.

[30] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[31] Yifan Fu, Xingquan Zhu, and Bin Li. A survey on instance selection for active learning. *Knowledge and Information Systems*, 35(2):249–283, 2013.

[32] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013.

[33] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the International Joint Conference on Neural Networks*, pages 189–194. IEEE, 2000.

[34] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[36] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[37] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[38] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[39] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[40] Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[41] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. *arXiv preprint arXiv:1706.04599*, 2017.

[42] Isabelle Guyon. Design of experiments of the NIPS 2003 variable selection benchmark, 2003.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[44] 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.

[45] Geoffrey Hinton. Neural networks for machine learning. Coursera, video lectures, 2012.

[46] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[48] Alex Holub, Pietro Perona, and Michael C Burl. Entropy-based active learning for object recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[49] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. In *Advances in Neural Information Processing Systems*, pages 892–900, 2010.

[50] Dino Ienco, Albert Bifet, Indrė Žliobaitė, and Bernhard Pfahringer. Clustering based active learning for evolving data streams. In *International Conference on Discovery Science*, pages 79–93. Springer, 2013.

[51] Dino Ienco, Indrė Žliobaitė, and Bernhard Pfahringer. High density-focused uncertainty sampling for active learning over evolving stream data. In *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 133–148, 2014.

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

[53] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307, 1988.

[54] Christoph Käding, Erik Rodner, Alexander Freytag, and Joachim Denzler. Active and continuous exploration with deep neural networks and expected model output changes. *arXiv preprint arXiv:1612.06129*, 2016.

[55] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[56] Ross D King, Kenneth E Whelan, Ffion M Jones, Philip GK Reiser, Christopher H Bryant, Stephen H Muggleton, Douglas B Kell, and Stephen G Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, 2004.

[57] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[58] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *The Second International Conference on Learning Representations*, 2014.

[59] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[60] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 2017.

[61] Daniel Kottke, Georg Krempl, Dominik Lang, Johannes Teschner, and Myra Spiliopoulou. Multi-class probabilistic active learning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 586–594, 2016.

[62] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. In *International Conference on Machine Learning*, pages 1863–1871, 2014.

[63] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[64] Bartosz Krawczyk, Bernhard Pfahringer, and Michał Woźniak. Combining active learning with concept drift detection for data stream mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2239–2244. IEEE, 2018.

[65] Georg Krempl, Daniel Kottke, and Myra Spiliopoulou. Probabilistic active learning: Towards combining versatility, optimality and efficiency. In *International Conference on Discovery Science*, pages 168–179. Springer, 2014.

[66] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[68] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

[69] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *The Fifth International Conference on Learning Representations*, 2017.

[70] 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.

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

[72] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[73] 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*, 2013.

[74] David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1994.

[75] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR94*, pages 3–12. Springer, 1994.

[76] Mingkun Li and Ishwar K Sethi. Confidence-based active learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1251–1261, 2006.

[77] Yanchao Li, Yongli Wang, Dong-Jun Yu, Ning Ye, Peng Hu, and Ruxin Zhao. Ascent: Active supervision for semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(5):868–882, 2019.

[78] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[79] Andrew K McCallum and Kamal Nigam. Employing em in pool-based active learning for text classification. In *Proceedings of 15th International Conference on Machine Learning, Madison, US*, pages 350–358, 1998.

[80] Andrew Kachites McCallumzy and Kamal Nigamy. Employing em and pool-based active learning for text classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer, 1998.

[81] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pages 234–243. Springer, 1978.

[82] D Mishkin and J Matas. All you need is a good in it. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[83] Tom M Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[84] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. In *The Fifth International Conference on Learning Representations*, 2017.

[85] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. In *The Fourth International Conference on Learning Representations*, 2016.

[86] Mor Shpigel Nacson, Jason Lee, Suriya Gunasekar, Pedro Henrique Pamplona Savarese, Nathan Srebro, and Daniel Soudry. Convergence of gradient descent on separable data. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3420–3428. PMLR, 2019.

[87] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

[88] Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems*, pages 3235–3246, 2018.

[89] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[90] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[91] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[92] MarcAurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann L Cun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2007.

[93] 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.

[94] N Roy and A McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the 18th International Conference on Machine Learning*, pages 441–448. Morgan Kaufmann, 2001.

[95] Igor Rudan, Cynthia Boschi-Pinto, Zrinka Biloglav, Kim Mulholland, and Harry Campbell. Epidemiology and etiology of childhood pneumonia. *Bulletin of the World Health Organization*, 86:408–416B, 2008.

[96] Sebastian Ruder and Barbara Plank. Strong baselines for neural semi-supervised learning under domain shift. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1044–1054, 2018.

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

[98] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. Fast and accurate recurrent neural network acoustic models for speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[99] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[100] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[101] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[102] Burr Settles. *Curious machines: Active learning with structured instances.* PhD thesis, University of Wisconsin–Madison, 2008.

[103] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[104] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079, 2008.

[105] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In *Advances in Neural Information Processing Systems*, pages 1289–1296, 2008.

[106] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 287–294, 1992.

[107] Claude E Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[108] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[109] Samuel L Smith and Quoc V Le. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017.

[110] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

[111] 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.

[112] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

[113] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[114] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[115] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[116] 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.

[117] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of

example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*, 2018.

[118] Kagan Tumer and Joydeep Ghosh. Estimating the bayes error rate through classifier combining. In *Proceedings of 13th international conference on pattern recognition*, volume 2, pages 695–699. IEEE, 1996.

[119] Kagan Tumer and Joydeep Ghosh. Bayes error rate estimation using classifier ensembles. *International Journal of Smart Engineering System Design*, 5(2):95–109, 2003.

[120] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[121] Julie S Weber and Martha E Pollack. Entropy-driven online active learning for interactive calendar management. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, pages 141–150, 2007.

[122] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning*, pages 1954–1963, 2015.

[123] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[124] Tengyu Xu, Yi Zhou, Kaiyi Ji, and Yingbin Liang. When will gradient methods converge to max-margin classifier under relu models? *arXiv preprint arXiv:1806.04339*, 2018.

[125] Yadollah Yaghoobzadeh, Remi Tachet, Timothy J Hazen, and Alessandro Sordoni. Robust natural language inference models with example forgetting. *arXiv preprint arXiv:1911.03861*, 2019.

[126] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated recurrent neural networks. *arXiv preprint arXiv:1508.03790*, 2015.

[127] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[128] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Appendix

A Additional results for Experiments in Section 4.5

Table A.1 reports additional results for experiments described in Section 4.5, measured using AUC-ROC, F1-Score and Cohen's Kappa.

Table A.1: Comparison of the model initialised using the supervised pretraining method (Pretrained) against the model initialised with *he_normal* (Base).

Data	**Method**	**AUC-ROC**	**F1-Score**	**Cohen's Kappa**
BanknoteID	**Base**	9.97E-01 ±7.04E-04	9.94E-01 ±3.30E-03	9.87E-01 ±6.61E-03
	Pretrained	9.96E-01 ±1.36E-04	9.86E-01 ±3.21E-03	9.71E-01 ±6.42E-03
BanknoteID_small	Base	9.06E-01 ±2.51E-03	8.51E-01 ±1.17E-03	7.02E-01 ±2.40E-03
	Pretrained	9.09E-01 ±2.16E-03	8.57E-01 ±2.54E-03	7.12E-01 ±5.77E-03
MAGIC	Base	9.00E-01 ±1.44E-03	8.36E-01 ±3.40E-03	6.74E-01 ±4.40E-03
	Pretrained	9.05E-01 ±1.07E-03	8.41E-01 ±1.53E-03	6.81E-01 ±3.28E-03
MAGIC_small	Base	8.78E-01 ±1.14E-03	7.96E-01 ±2.03E-03	5.91E-01 ±4.07E-03
	Pretrained	8.80E-01 ±5.24E-03	7.99E-01 ±5.26E-03	5.99E-01 ±1.05E-02
Waveform	Base	9.89E-01 ±3.39E-04	9.41E-01 ±1.50E-03	8.81E-01 ±3.00E-03
	Pretrained	9.90E-01 ±4.07E-04	9.44E-01 ±7.97E-04	8.88E-01 ±1.59E-03
Waveform_small	Base	9.82E-01 ±6.30E-04	9.25E-01 ±3.96E-03	8.49E-01 ±7.91E-03
	Pretrained	9.84E-01 ±1.47E-03	9.33E-01 ±1.49E-03	8.67E-01 ±2.98E-03
Waveform_noise	**Base**	9.90E-01 ±5.53E-04	9.47E-01 ±3.26E-03	8.93E-01 ±6.53E-03
	Pretrained	9.89E-01 ±4.78E-04	9.44E-01 ±4.19E-03	8.88E-01 ±8.38E-03
Waveform_noise_small	Base	9.84E-01 ±1.23E-03	9.26E-01 ±4.36E-03	8.53E-01 ±8.71E-03
	Pretrained	9.89E-01 ±2.56E-04	9.40E-01 ±1.69E-03	8.79E-01 ±3.37E-03

B Hyperparameters for Experiments in Section 5.4

All the hyperparameters used in the experiments in Section 5.4 are reported here. All hyperparameters were chosen based on the validation results. When tuning hyperparameters we set the data seed to 0 in all cases. Data seed only affects the split of labelled training data and unlabelled training data, it does not change the validation set or the test set.

B.1 Hyperparameters for Pseudo-Label

Table B.1 and Table B.2 show the hyperparameters used for Pseudo-Label in experiments of Section 5.4.1. Table B.3 and Table B.4 show the hyperparameters used for Pseudo-Label in experiments of Section 5.4.2.

Table B.1: Hyperparameters used for Pseudo-Label when unlabelled data is clean.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	1.000	5.000	1.000
lr	0.100	0.100	0.001
γ	0.990	0.900	0.900

Table B.2: Hyperparameters used for Pseudo-Label when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	1.000	5.000	1.000
lr	0.100	0.100	0.001
γ	0.990	0.900	0.900

Table B.3: Hyperparameters used for Pseudo-Label with weights calculated from raw images when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	1.000	0.500	1.000
lr	0.100	0.100	0.001
γ	0.990	0.900	0.900
β	10.000	10.000	20.000

Table B.4: Hyperparameters used for Pseudo-Label with weights calculated from encodings when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	1.000	0.500	1.000
lr	0.100	0.100	0.001
γ	0.990	0.900	0.900
β	10.000	5.000	5.000

B.2 Hyperparameters for Mean Teacher

Table B.5 and Table B.6 show the hyperparameters used for Mean Teacher in experiments of Section 5.4.1. Table B.7 and Table B.8 show the hyperparameters used for Mean Teacher in experiments of Section 5.4.2.

Table B.5: Hyperparameters used for Mean Teacher when unlabelled data is clean.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	5.000	50.000	100.000
T	10.000	10.000	10.000
lr	0.300	0.100	0.001
γ	0.999	0.990	0.900
η	0.900	0.900	0.995

Table B.6: Hyperparameters used for Mean Teacher when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	5.000	50.000	50.000
T	10.000	10.000	10.000
lr	0.300	0.100	0.001
γ	0.999	0.990	0.900
η	0.900	0.900	0.995

Table B.7: Hyperparameters used for Mean Teacher with weights calculated from raw images when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	5.000	1.000	100.000
T	10.000	10.000	10.000
lr	0.300	0.100	0.001
γ	0.999	0.990	0.900
η	0.900	0.900	0.999
β	10.000	10.000	2.000

Table B.8: Hyperparameters used for Mean Teacher with weights calculated from encodings when unlabelled data is dirty.

Hyperparameter	**MNIST**	**Fashion-MNIST**	**CIFAR-10**
a	50.000	50.000	100.000
T	10.000	10.000	10.000
lr	0.300	0.100	0.001
γ	0.999	0.990	0.900
η	0.900	0.900	0.999
β	10.000	3.000	2.000

C Hyperparameters for Experiments in Section 6.6

The hyperparameters used in the experiments in Section 6.6 are reported here for reproducibility. The number of queried examples is denoted as *num_queried*, and the number of initial labelled examples is referred to as *num_init*. The value "-" in the tables indicates that this setting was not used in our experiments.

C.1 MNIST

For MNIST dataset, we set learning rate *lr* to 0.1. The total *epochs* used for all experiments is 500. Table C.1 reports the *skip* value used in all experiments.

Table C.1: The hyperparameter *skip* used for experiments on MNIST. The *lr* used is 0.1, the total *epochs* is 500.

num_init	**num_queried**				
	100	**500**	**1000**	**5000**	**10000**
100	300	200	100	100	0
500	150	-	-	-	-
1000	100	-	-	-	-
5000	0	-	-	-	-
10000	0	0	0	0	0

Table C.2: The total *epochs* used for experiments on Fashion-MNIST. The *lr* used is 0.3.

num_init	**num_queried**				
	100	**500**	**1000**	**5000**	**10000**
100	500	500	500	500	500
500	500	-	-	-	-
1000	400	-	-	-	-
5000	300	-	-	-	-
10000	300	300	300	300	300

C.2 Fashion-MNIST

For Fashion-MNIST dataset, the learning rate *lr* used is 0.3 for all experiments. Table C.2 reports the total *epochs* used in the experiments, and Table C.3 reports the *skip* value used.

Table C.3: The hyperparameter *skip* used for experiments on Fashion-MNIST. The *lr* used is 0.3.

num_init	**num_queried**				
	100	**500**	**1000**	**5000**	**10000**
100	400	200	200	200	200
500	100	-	-	-	-
1000	300	-	-	-	-
5000	100	-	-	-	-
10000	0	0	0	0	0

C.3 CIFAR-10

For CIFAR-10 dataset, the learning rate *lr* used is 0.05 for all experiments. Table C.4 reports the total *epochs* used in the experiments, and Table C.5 reports the *skip* value used.

Table C.4: The total *epochs* used for experiments on CIFAR-10. The *lr* used is 0.05.

num_init	**num_queried**				
	100	**500**	**1000**	**5000**	**10000**
100	800	800	800	800	800
500	-	500	-	-	-
1000	-	400	-	-	-
5000	-	300	-	-	-
10000	300	300	300	300	300

Table C.5: The hyperparameter *skip* used for experiments on CIFAR-10. The *lr* used is 0.05.

num_init	**num_queried**				
	100	**500**	**1000**	**5000**	**10000**
100	600	500	600	500	500
500	-	300	-	-	-
1000	-	300	-	-	-
5000	-	200	-	-	-
10000	50	150	100	0	0

D Experimental Results for Section 6.7

Below are all the experimental results reported in Section 6.7, where *num_queried* is the number of queried examples and *num_init* is the number of initial labelled examples. The tables report the mean test accuracies and the standard deviations in parentheses. The value “-” indicates that this setting was not used in our experiments.

D.1 MNIST

Table D.1: Experimental results on MNIST without using active learning (*no_AL*).

num_init	**test accuracy**
100	81.38% (1.80%)
500	91.95% (0.61%)
1000	94.37% (0.40%)
5000	97.70% (0.15%)
10000	98.36% (0.12%)

Table D.2: Experimental results on MNIST using *uniform*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	86.98% (1.06%)	92.55% (0.51%)	94.66% (0.36%)	97.69% (0.14%)	98.37% (0.10%)
500	92.61% (0.59%)	-	-	-	-
1000	94.69% (0.36%)	-	-	-	-
5000	97.70% (0.14%)	-	-	-	-
10000	98.37% (0.11%)	98.41% (0.11%)	98.45% (0.12%)	98.66% (0.10%)	98.81% (0.09%)

Table D.3: Experimental results on MNIST using *kcenter_greedy*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	88.35% (1.05%)	94.20% (0.41%)	96.12% (0.27%)	98.42% (0.10%)	98.83% (0.07%)
500	93.18% (0.51%)	-	-	-	-
1000	95.10% (0.33%)	-	-	-	-
5000	97.80% (0.12%)	-	-	-	-
10000	98.42% (0.11%)	98.50% (0.10%)	98.61% (0.10%)	98.88% (0.07%)	98.99% (0.06%)

Table D.4: Experimental results on MNIST using *graph_density*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	78.70% (2.32%)	78.51% (1.91%)	80.52% (1.62%)	94.35% (0.47%)	98.32% (0.07%)
500	91.74% (0.69%)	-	-	-	-
1000	94.38% (0.36%)	-	-	-	-
5000	97.72% (0.12%)	-	-	-	-
10000	98.38% (0.11%)	98.36% (0.11%)	98.37% (0.10%)	98.48% (0.11%)	98.78% (0.07%)

Table D.5: Experimental results on MNIST using *margin*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	90.01% (0.92%)	94.64% (0.47%)	96.18% (0.32%)	98.27% (0.16%)	98.76% (0.12%)
500	94.37% (0.49%)	-	-	-	-
1000	95.75% (0.30%)	-	-	-	-
5000	97.94% (0.12%)	-	-	-	-
10000	98.49% (0.11%)	98.73% (0.08%)	98.88% (0.08%)	99.13% (0.06%)	99.17% (0.06%)

Table D.6: Experimental results on MNIST using *proposed*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	90.14% (0.78%)	94.76% (0.42%)	95.96% (0.49%)	98.04% (0.17%)	98.47% (0.20%)
500	94.29% (0.43%)	-	-	-	-
1000	95.70% (0.31%)	-	-	-	-
5000	97.95% (0.13%)	-	-	-	-
10000	98.49% (0.09%)	98.75% (0.08%)	98.87% (0.08%)	99.08% (0.07%)	99.07% (0.06%)

D.2 Fashion-MNIST

Table D.7: Experimental results on Fashion-MNIST without using active learning (*no_AL*).

num_init	**test accuracy**
100	67.52% (2.26%)
500	77.31% (1.05%)
1000	80.65% (0.70%)
5000	86.02% (0.35%)
10000	87.70% (0.34%)

Table D.8: Experimental results on Fashion-MNIST using *uniform*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	71.85% (1.51%)	78.40% (0.90%)	81.17% (0.57%)	86.02% (0.43%)	87.70% (0.41%)
500	78.27% (1.01%)	-	-	-	-
1000	81.17% (0.69%)	-	-	-	-
5000	86.02% (0.38%)	-	-	-	-
10000	87.72% (0.34%)	87.82% (0.48%)	87.83% (0.76%)	88.62% (0.32%)	89.25% (0.34%)

Table D.9: Experimental results on Fashion-MNIST using *kcenter_greedy*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	71.55% (1.30%)	76.11% (0.93%)	79.25% (0.92%)	85.43% (0.46%)	87.68% (0.52%)
500	78.40% (0.93%)	-	-	-	-
1000	81.25% (0.65%)	-	-	-	-
5000	86.10% (0.35%)	-	-	-	-
10000	87.78% (0.48%)	87.99% (0.50%)	88.09% (0.34%)	89.02% (0.34%)	89.56% (0.32%)

Table D.10: Experimental results on Fashion-MNIST using *graph_density*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	68.92% (1.60%)	71.59% (1.43%)	73.90% (1.17%)	84.59% (0.47%)	87.99% (0.28%)
500	78.08% (0.80%)	-	-	-	-
1000	81.09% (0.70%)	-	-	-	-
5000	85.99% (0.55%)	-	-	-	-
10000	87.70% (0.49%)	87.81% (0.38%)	87.81% (0.34%)	88.36% (0.62%)	89.26% (0.28%)

Table D.11: Experimental results on Fashion-MNIST using *margin*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	73.13% (1.43%)	80.05% (0.94%)	82.21% (0.76%)	86.65% (0.56%)	88.16% (0.49%)
500	79.28% (0.71%)	-	-	-	-
1000	81.71% (0.58%)	-	-	-	-
5000	86.16% (0.46%)	-	-	-	-
10000	87.81% (0.37%)	87.87% (1.46%)	88.36% (0.33%)	89.87% (0.25%)	90.67% (0.24%)

Table D.12: Experimental results on Fashion-MNIST using *proposed*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	73.58% (1.52%)	79.26% (1.37%)	81.76% (1.15%)	86.31% (0.49%)	87.95% (0.31%)
500	78.99% (0.81%)	-	-	-	-
1000	81.60% (0.71%)	-	-	-	-
5000	86.17% (0.36%)	-	-	-	-
10000	87.83% (0.35%)	88.11% (0.35%)	88.52% (0.30%)	89.63% (1.40%)	90.65% (0.27%)

D.3 CIFAR-10

Table D.13: Experimental results on CIFAR-10 without using active learning (*no_AL*).

num_init	**test accuracy**
100	18.35% (2.59%)
500	28.24% (0.94%)
1000	32.84% (1.16%)
5000	44.77% (1.09%)
10000	50.73% (0.87%)

Table D.14: Experimental results on CIFAR-10 using *uniform*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	21.48% (1.87%)	28.94% (1.38%)	33.01% (1.25%)	44.57% (1.04%)	50.68% (1.05%)
500	-	31.57% (2.40%)	-	-	-
1000	-	35.60% (1.42%)	-	-	-
5000	-	45.72% (0.88%)	-	-	-
10000	51.17% (0.65%)	51.07% (0.92%)	51.81% (0.84%)	54.62% (0.94%)	57.73% (0.82%)

Table D.15: Experimental results on CIFAR-10 using *kcenter_greedy*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	19.72% (2.10%)	25.72% (1.30%)	28.81% (0.97%)	40.50% (0.88%)	47.04% (1.14%)
500	-	30.71% (1.46%)	-	-	-
1000	-	33.71% (1.89%)	-	-	-
5000	-	45.28% (1.12%)	-	-	-
10000	50.95% (1.10%)	51.28% (0.83%)	51.72% (0.92%)	53.77% (1.19%)	56.92% (0.98%)

Table D.16: Experimental results on CIFAR-10 using *graph_density*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	21.25% (2.12%)	25.08% (1.99%)	28.49% (1.46%)	44.05% (1.29%)	50.61% (0.75%)
500	-	28.59% (4.81%)	-	-	-
1000	-	34.51% (1.66%)	-	-	-
5000	-	44.86% (1.27%)	-	-	-
10000	50.81% (0.78%)	51.04% (0.92%)	51.42% (0.97%)	54.65% (0.80%)	57.60% (0.88%)

Table D.17: Experimental results on CIFAR-10 using *margin*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	22.00% (1.27%)	28.47% (1.55%)	31.96% (5.47%)	44.45% (1.20%)	50.25% (0.77%)
500	-	32.51% (1.52%)	-	-	-
1000	-	35.50% (2.15%)	-	-	-
5000	-	45.56% (1.38%)	-	-	-
10000	50.79% (1.14%)	50.73% (1.33%)	51.76% (1.01%)	55.44% (0.95%)	58.64% (0.91%)

Table D.18: Experimental results on CIFAR-10 using *proposed*.

num_init	**num_queried**				
	100	500	1000	5000	10000
100	22.29% (1.41%)	28.86% (1.68%)	32.82% (1.35%)	43.52% (1.61%)	50.14% (1.41%)
500	-	33.23% (1.46%)	-	-	-
1000	-	35.28% (1.39%)	-	-	-
5000	-	45.78% (1.17%)	-	-	-
10000	51.17% (0.79%)	51.70% (0.83%)	51.95% (0.71%)	55.72% (1.33%)	59.19% (1.39%)