

# **An Attempt to Reduce the Number of Training Samples for Convolutional Neural Networks**

*Yuwен Heng*

Master of Science  
Data Science, Technology, and Innovation  
School of Informatics  
University of Edinburgh  
2020

# Abstract

Training deep neural networks can be resources-consuming. The budget required is increasing with the size of the dataset. During the past ten years, many achievements are dedicated to accelerating the convergence speed with heuristic or theoretical training procedures. However, we still need the whole dataset to train the network and paying for a large dataset may not pay back well if we can use a smaller subset to achieve an acceptable performance. In order to reduce the number of training samples needed, we first adapted and evaluated three methods, Patterns by Ordered Projections (POP), Enhanced Global Density-based Instance Selection (EGDIS), and Curriculum Learning (CL), to reduce the size of two image datasets, CIFAR10 and CIFAR100, for the classification task. Based on the analysis, we present our main contributions: improved CL and evaluated its two variations, the Weighted Curriculum Learning (WCL) and the Boundary based Weighted Curriculum Learning (BWCL). The WCL outperforms POP and EGDIS in terms of both classification accuracy and time complexity. Also, WCL and BWCL achieve comparable performance compared with CL while keeping a portion of hard examples. Besides, we proposed a trade-off framework for WCL to select a subset of samples according to the acceptable relative accuracy and the original datasets.

## Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr Yang Cao, for offering me the opportunity to work with him on such an attracting and challenging project. His encouragement and valuable guidance helped me tackle the obstacles in my research path.

Furthermore, special thanks go to Professor Bob Fisher, Dr Pavlos Andreadis at the University of Edinburgh and Dr Jiacheng Ni at IBM for sharing me their knowledge about computer vision and deep learning. The programming skills and coursework experience that I learnt from them helped me to organise the experiments well.

Finally, I would like to send my love to my fiancee Danni Li for her accompany during the past three years. I wouldn't have the chance to study full-time without her full support.

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background Research</b>	<b>3</b>
2.1	Classification and Feature Extraction with CNNs . . . . .	3
2.2	Training Set Arrangement . . . . .	5
2.2.1	Current Hypothesis Methods . . . . .	5
2.2.2	Target Hypothesis Methods . . . . .	7
2.3	Instance Selection Algorithm . . . . .	8
2.3.1	Patterns by Ordered Projections . . . . .	9
2.3.2	Enhanced Global Density Based Instance Selection . . . . .	10
2.4	Trade-off Framework . . . . .	11
<b>3</b>	<b>Adapted Data Reduction Methods</b>	<b>13</b>
3.1	Datasets and Image Feature Extraction . . . . .	14
3.2	Difficulty Tuneable Algorithms . . . . .	15
3.2.1	Weighted Curriculum Learning . . . . .	15
3.2.2	Boundary Based Weighted Curriculum Learning . . . . .	16
3.3	Evaluation Designs . . . . .	17
<b>4</b>	<b>Data Reduction Evaluations</b>	<b>18</b>
4.1	Experiment 1: Feature Extraction . . . . .	18
4.2	Experiment 2: Intrinsic Behaviour . . . . .	21
4.3	Experiment 3: Logistic Regression . . . . .	25
4.4	Experiment 4: Data Reduction for CNN . . . . .	29
<b>5</b>	<b>Trade-off Framework</b>	<b>32</b>
5.1	Subset Selection Framework . . . . .	32

<b>6 Conclusion and Future Work</b>	<b>35</b>
<b>Bibliography</b>	<b>36</b>
<b>A Training History</b>	<b>41</b>
A.1 Feature Extraction for CIFAR20 and CIFAR40 . . . . .	41
A.2 Logistic Regression Original History . . . . .	42
A.3 CNN Original History . . . . .	42

# Chapter 1

## Introduction

With the development of Convolutional Neural Networks (CNNs), many computer vision challenges have proven to achieve high performances on image datasets. Krizhevsky et al. [24] designed an eight-layer CNN and outperformed the record in the ImageNet Large Scale Visual Recognition Competition (ILSVRC2012) by about 9.4 per cent. Three years later, the very deep CNN by He et al. [16, 17] surpassed human-level performance on the same dataset. It is possible to train CNNs with hundreds of layers due to the increment of dataset size, robust initialisation methods, development of GPU training frameworks, advanced regularisation skills, and network architectures such as highway connection [17, 18]. These achievements made deep CNNs the dominant choice for the image classification task. While deep architectures achieved high accuracies, the large-scale datasets also caused CNNs both time-consuming and eager for computing and storage resources.

In order to overcome these drawbacks, recent research developed many training data arrangement procedures to accelerate the CNN convergence speed. These approaches typically fall into one of the two categories: select mini-batch samples non-uniformly [36, 26, 20, 8] or rank the order by which samples are fed into CNNs during training [4, 15]. Both methods need to evaluate sample classification scores. We have two options on when to evaluate the classification scores: at each training step or calculate once before the training process starts. We use the term **Current Hypothesis Method** and **Target Hypothesis Method** to refer to these situations [15]. However, we still need to train the whole training set or evaluate them during training.

There are methods designed to reduce the number of training samples which have structured features called **instance selection** algorithms [30, 2, 7, 33]. The typical approach is to select samples that can maintain the decision boundary of machine learning

algorithms such as k-nearest neighbour [28]. However, to the best of our knowledge, researchers have not developed an efficient pipeline to make them work with image datasets and CNNs [38, 3, 5]. One reason is that images are not structural data. The other reason is that these selection algorithms are not optimised for CNNs. Hence, we need to build a pipeline for these algorithms to be compatible with CNNs.

The aim of this project is to establish an efficient pipeline to reduce the number of training samples needed for CNNs. We take the image classification task as an example in particular. To do so, we will adapt typical instance selection algorithms and the target hypothesis methods to reduce the size of training sets for CNNs. Before CNNs became popular, researchers tend to reduce the image dimensionality by transforming the images into feature vectors then classify these features with machine learning algorithms such as SVM [32, 11]. Since these extracted features can be considered as structural features, and the pre-calculated classification scores can reflect the sample importances for CNNs, it is possible to achieve a better reduction performance, in terms of classification accuracy and algorithm compute time.

In specific, we will extract the image feature vectors as a pre-processing step. The existing instance selection algorithms are extended with the awareness of sample classification scores. Since there is no enough experience to guide us configure the algorithms for CNNs, we first explore the behaviours by visualising the selected samples and train the subsets with the logistic regression method. After tuning the hyper-parameters and selecting the same amount of subsets on the feature vectors, we train CNNs from scratch with corresponding images and report the relative classification accuracies. We then dive into the most suitable algorithm and build the trade-off framework, which can guide researchers to balance the relative accuracy and the number of samples selected.

This dissertation is structured as follows: Chapter 2 describes the necessary background to understand image feature extraction, CNN dataset arrangement techniques, and the instance selection algorithms. We then cover the experiments to evaluate instance selection algorithms in Chapter 3. We explain the datasets and feature extraction procedures used in the experiments, detail the instance selection algorithms proposed for CNNs, and clarify the evaluation designs. The results of our experiments are reported in Chapter 4. In Chapter 5, we extend the adapted instance selection algorithms to select subsets for a given relative accuracy. Finally, in Chapter 6, we finish this thesis with a conclusion and a discussion about future work.

# Chapter 2

## Background Research

In this chapter, we begin with presenting the necessary background to understand the CNNs for classification and feature extraction task, training data arrangement procedures, and instance selection methods, as well as other ideas required to understand our research. We start with the typical structure of CNNs and the gradient descent training method. We then discuss the advanced training set arrangement methods that can speed up the training procedure and outline their deficiencies for our purpose. Next, we review the instance selection literature and present a CNN instance selection pipeline - use the network pre-trained on ImageNet to extract low-dimensional features and run the instance selection methods on extracted features. Furthermore, we cover the existing trade-off framework BlinkML [31] in the context of maximum-likelihood estimation machine learning algorithms and explain why it is not suitable for the deep neural network. Finally, we present TAPAS [19], which is an accuracy predictor for the deep neural network without training and has several properties that make it useful to build our trade-off framework.

### 2.1 Classification and Feature Extraction with CNNs

Classification is a kind of machine learning task which learns the mapping between visual inputs and output labels from a set of well-labelled training samples. The visual data can be images, videos or even 3D models [37]. The output scores after a softmax operation can be considered as the probability for a given image belonging to each class. We use the symbol  $P(c|x, \theta)$  to represent the probability that sample  $x$  belongs to true class  $c$ . The scores of true classes also reflect the difficulties for CNNs to classify the samples correctly. Higher scores indicate the samples are easier to classify than

lower score samples. CNNs are particular tools that can solve this task. They are a set of chained operations with trainable parameters. These parameters define the actual input-out mapping. For this reason, we use the symbol  $f(x|\theta)$  to represent the output score of true class predicted by the CNNs, which takes the input  $x$  with a particular parameter set  $\theta$ .

Figure 2.1 gives a basic CNN structure which is optimised to classify images as cats or dogs. It contains two convolutional (Conv) layers, one max-pooling layer and one fully connected (FC) layer. The last FC layer is a multi-class logistic regression model which maps the outputs of the max-pooling layer to the classification scores. From this perspective, we can divide the CNN structure into two parts: feature extraction part and logistic regression part. The feature extraction part performs as a black box which ideally transforms the input images to points in a lower-dimensional, linearly separable space.

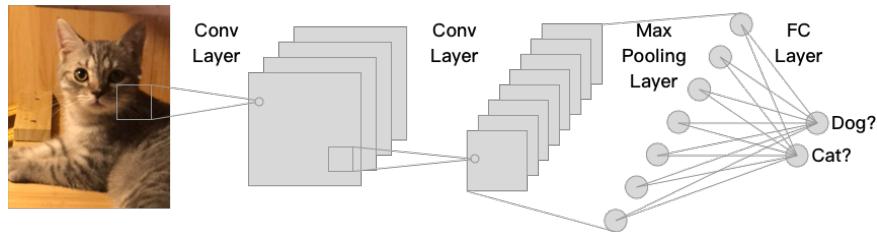


Figure 2.1: A basic CNN structure to classify images between cats and dogs. The outputs of the penultimate FC layer are the extracted lower-dimensional features of the input images. These features should be linearly separable to achieve a high classification accuracy.

We use the symbol  $y$  to represent the ground truth of the input sample  $x$ . The equation  $L(f(x|\theta), y)$  represents the loss function which measures the difference between the predicted output and the ground truth label. Then the training process is to find the parameter set  $\theta^*$ , which minimise the average loss of the whole training set as follows:

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(f(x_i|\theta), y_i) \quad (2.1)$$

where the symbol  $N$  stands for the number of samples in the training set. This equation turns the training process into an optimisation problem. Different from machine learning algorithms like logistic regression and SVM, the equation 2.1 is non-convex thus cannot be solved analytically [13, p. 304]. Several techniques have been developed to solve this problem with the requirement that the loss function  $L(.,.)$  is differ-

entiable. The basic one is called stochastic gradient descent (SGD) which updates the parameters with the partial derivatives of a randomly selected sample. At each step, the new parameter is calculated with:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial L(f(x|\theta), y)}{\partial \theta_t} \quad (2.2)$$

and  $\eta$  is the step size. A simple variant of SGD is mini-batch gradient descent which divides the training set into disjoint subsets and averages the gradients within the subset before updating the parameters:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{M} \sum_{i=1}^M \frac{\partial L(f(x_i|\theta), y_i)}{\partial \theta_t} \quad (2.3)$$

where  $M$  is the batch size of the subset. For CNNs, batch size  $M$  is often smaller than the training set size  $N$  because it takes too much memory to fit in the whole dataset. Usually, we use the value 128 or 256 as the batch size.

The feature extraction method mentioned above is to train a network first then take the penultimate FC layer outputs. In practice, this is not efficient nor effective. If we have an extensive training set, it takes us too many resources to train them well [9]. If we have a small training set, it may be hard to acquire high-quality features. An alternative method is to extract the features with a pre-trained network. Typically we use the weights trained on ImageNet [34] because this dataset is large enough and the pre-trained network can extract good enough features for other datasets [22].

## 2.2 Training Set Arrangement

Since the mini-batch gradient method trains the network with a subset of samples at each step, how to select the samples becomes a problem in the deep learning literature. Instead of uniform sampling, many researchers proposed to select the samples with sample weights based on different criteria. In this section, we plan to introduce the current hypothesis methods and target hypothesis methods. Current hypothesis methods measure the samples based on the parameter set  $\theta_t$  at step  $t$  while target hypothesis methods are based on the final optimal parameter set  $\theta^*$ .

### 2.2.1 Current Hypothesis Methods

Different authors have proposed a variety of current hypothesis methods. Specifically, in self-paced learning [25, 26, 29], active bias learning [8], and hard example mining

[36, 27], the batch selection process is based on the classification scores  $f(x|\theta, y)$ . For importance sampling methods, the process is based on the gradient norm for each sample,  $|\frac{\partial L(f(x_i|\theta), y_i)}{\partial x_i}|$ . We finish this section by briefly explaining the theories of these approaches.

### 2.2.1.1 Difficulty Based Methods

Self-paced learning tends to select easy samples which have a high classification score by injecting a pace function into the optimisation target function 2.1:

$$\theta^* = \arg \min_{\theta, v} \sum_{i=1}^N v_i L(f(x_i|\theta), y_i) + \lambda \sum_{i=1}^N v_i \quad (2.4)$$

where  $v$  is the sample weight calculated by the pace function. The pace function can be either a simple step function [25] or a more complicated dynamic function which changes while training  $t$  [26] as long as it can assign weight 0 to samples. By minimising the target function 2.4, this method would zero out hard examples which have higher loss values, thus keeps only the easy samples. With self-paced learning, the trained network can be more robust to outliers [29].

A potential problem of self-paced learning is that it would gradually increase the loss of hard examples [25]. As a consequence, the trained network may not achieve the desired accuracy. The possible solution is to use the active bias learning method, which is designed to select the uncertain samples whose classification scores fluctuate near the decision threshold. Chang et al. proposed and evaluated many self-paced methods, and the representative one is SGD Sampled by Threshold Closeness (SGD-STC) [8]. It records the historical average classification probability  $\bar{P}$  for each sample. The sample weights are calculated with an equation that is proportional to  $(1 - \bar{P}) \times \bar{P}$  whose maximum point is at  $\bar{P} = 0.5$ . However, the problem is that we need extra space and computation to maintain historical scores.

Hard example mining is yet another heuristic method aims at maximising the convergence speed by extending the self-paced learning method [36]. The algorithm proposed by [27] ranks the samples based on the latest computed classification score in descending order. At early training stages, the algorithm chooses easy samples just like self-paced learning. After a thorough training process, the algorithm tends to select hard examples which have low classification scores. In this way, the trained classifier may be able to achieve higher accuracy than self-paced learning. The downside is that training with hard examples can affect the decision boundary. As a consequence, the network may forget learned features previously and reduce the accuracy.

### 2.2.1.2 Importance Based Methods

Although experiments in the cited resources above have proved that difficulty based methods can surely speed up the training process and may achieve even higher accuracy, the lack of mathematical prove could lower the interests of researchers. On the contrary, importance based methods raise from the profound mathematical demonstration [41] and are more reliable. Despite the elaborate derivation, the most important conclusion is that the optimal weight distribution is proportional to the per-sample gradient norm.

The challenge is that computing the per sample gradient norm  $|\frac{\partial L(f(x_i|\theta), y_i)}{\partial x_i}|$  is intractable. In the past few years, many researchers have adapted their approximate methods to speed up the process. The most convincing one is proposed by Katharopoulos et al. which derives an upper bound of the gradient norm [21],

$$\left| \frac{\partial L(f(x_i|\theta), y_i)}{\partial x_i} \right| \leq |h(x_i)| \quad (2.5)$$

that  $h(x_i)$  is the upper bound function depends on the last layer pre-activation outputs. With this equation, we can compute the largest sample gradient after a single forward propagation.

The benefit of current hypothesis methods is that the sample weights vary with training step. Thus the chosen samples at each step can reflect the current capacity of the network. However, because evaluating the whole training set is time-consuming, we often select a subset uniformly first and then select the samples within the subset. As a result, we can only get a sub-optimal choice which is no better than the theory performance.

### 2.2.2 Target Hypothesis Methods

Compared with the current hypothesis methods, target hypothesis methods arrange the training set based on the possible final classification scores of the network thus the weights of the samples are pre-defined. They will not change during the training process [4]. For this reason, the target hypothesis method is more suitable to reduce the size of the datasets. To our knowledge, Curriculum Learning (CL) is the only method with these properties, as stated by Hacohen et al. in 2019 [15].

Similar to hard example mining, CL trains the network with easy samples first. Rather than switching to difficult samples, CL adds difficult samples into the training set. Eventually, the subset would contain all the training samples. The classification

scores are measured with a pre-trained network or with a linear classifier such as SVM [15]. The major concern is that CL weights may not reflect the sample classification difficulties of the chosen CNN. Considering the reality that all methods described above are sub-optimal in practice, we choose to accept the drawbacks in this project.

The Python style<sup>1</sup> pseudo-code of CL is shown in Algorithm 1. In this dissertation, we assume the type of input feature vector is NumPy array.

---

**Algorithm 1:** CL

---

**Data:** image feature vectors  $M$

**Input:** number of samples to select  $m$ , classification score for each sample  
 $scores$ , number of classes  $n$

**Output:** selected sample index by CL

```

1 selected_idx_list = [] a;
2 foreach class label  $L$  do
3   | scores = all sample scores with label  $L$  ;
4   | idx_list = sort_by_value(scores) ;
5   | selected_idx_list.append(idx_list[: floor(m/n)]) ;
6 end
7 return selected_idx_list ;
```

---

<sup>a</sup>All variables ended with the postfix *\_list* are Python list objects.

## 2.3 Instance Selection Algorithm

Most instance selection methods are designed to reduce the size of the structured dataset for machine learning algorithms like SVM and logistic regression. The assumption is that we can recover the decision boundary with fewer samples. According to the thorough review [30], the instance selection methods can be divided into two categories: wrapper and filter. Wrapper methods select the subset samples based on the classification results. K-NN [2] is a common choice to evaluate classification quality. Misclassified samples will be selected because they can contribute to the K-NN accuracy. Filter methods select samples without repetitive evaluation. They tend to select samples near the boundary between different classes with the assumption that these samples can guide the classifiers to recover the position of the original decision boundary [33, 28]. Although the wrapper methods could achieve higher accuracy because they are classifier dependent, they tend to cost too much time because they need

---

<sup>1</sup>We use Python objects and their functions such as `.append()`.

to evaluate the accuracy multiple times. From this perspective, filter methods are more suitable for our experiments. In this section, we take two typical filter algorithms, POP and EGDIS, to introduce their mechanisms.

### 2.3.1 Patterns by Ordered Projections

The POP algorithm [33] is designed to select boundary samples by removing inner samples which are far from the class contour. Rather than calculating the sample positions in the high dimensional feature space, Riquelme et al. simplified this process by projecting the samples onto each feature dimension. To be precise, we decide if a sample is inner or not in each feature space. We use the term **pure inner samples** to refer to samples which are inner in all feature spaces.

Algorithm 2 describes the POP pseudo-code. we use the variable *weakness* to count the number of times that a sample is inner. In each feature dimension, the function *sort\_by\_value* first sort the samples in descending order. Since our extracted features are continuous values, we need an extra hyper-parameter *equal tolerance*(*et*) to decide when two feature values are equal. Then the function *resort\_by\_label* scans the samples and record the start label of consecutive samples with the same value and detects the label of the first sample with a different value. Then the function sorts the scanned samples by moving samples with the same labels as the two recorded to the start of the list and the end of the list. After that, we scan all samples one more time to detect label changes and mark all samples as inner except the two at the beginning and the end of the scanned list. Figure 2.2 depicts a typical work flow of the function *resort\_by\_label*.

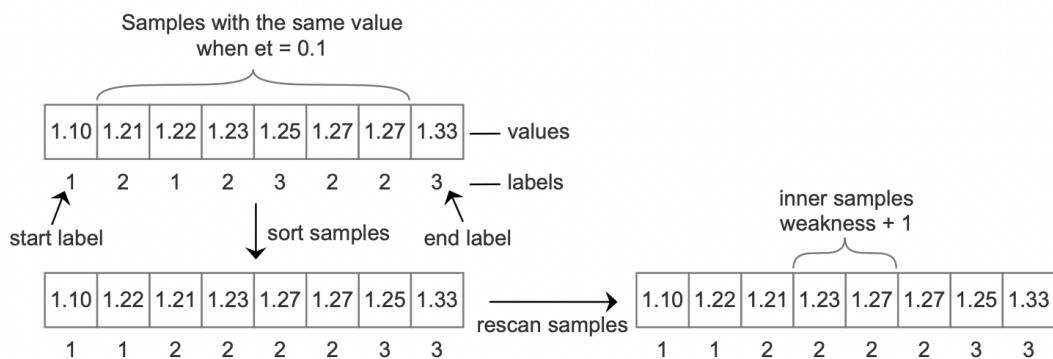


Figure 2.2: The typical workflow of POP.

---

**Algorithm 2:** POP for continuous features

---

**Data:** feature vectors  $M$

**Input:** weakness threshold  $wt$ , equal tolerance  $et$

**Output:** selected sample index by POP

```

1  weakness = np.zeros(len(M)) ;
2  foreach feature dimension  $F_j = M[:, j]$  do
3      idx_list = sort_by_value( $F_j$ ) ;
4      idx_list = resort_by_label( $F_j$ , et, idx_list) ;
5      foreach idx in idx_list do
6          if  $M[idx, j]$  is inner then
7              |   weakness[idx] += 1;
8          end
9      end
10 end
11 return np.argwhere(weakness < wt) ;

```

---

### 2.3.2 Enhanced Global Density Based Instance Selection

Similar to POP, EGDIS [28] aims at selecting boundary samples as well. Instead of removing inner samples, EGDIS selects the boundary samples, which are close to other classes. It also selects samples at the densest area to capture some inner samples. The pseudo-code is shown in Algorithm 3. In order to find these samples, the function *kneighbours* calculates the distances between the  $k$  neighbours. Then for each sample, we check how many neighbours are from other classes and save the value to variable *irrelevancescore*. If the score is greater or equal to the integer part of  $k/2$ , we record this sample to the *boundary\_idx* list. If not, function *density* calculates the global density values:

$$\text{density}(x_i) = -\frac{1}{N} \sum_{j \neq i} \text{distance}(x_i, x_j) \quad (2.6)$$

for the sample and its  $k$  neighbours. We add the sample to the *densest\_idx* list if its density value is the highest compared with its  $k$  neighbours. According to the original paper, EGDIS performs better with global density function in terms of reduction rate<sup>2</sup>. However, the compute time increases with the number of samples. We will explore possible solutions in Chapter 3.

---

<sup>2</sup>Reduction rate is the percentage of samples selected

---

**Algorithm 3:** EGDIS

---

**Data:** feature vectors  $M$

**Input:** number of neighbourhoods  $k$

**Output:** selected sample index by EGDIS

```

1 boundary_idx = [] ;
2 densest_idx = [] ;
3 neighbour_distance_list, neighbour_index_list = kneighbours( $M, k$ ) ;
4 for  $i$  in range( $\text{len}(\text{neighbour\_index\_list})$ ) do
5   neighbour_index = neighbour_index_list[ $i, :]$  ;
6   irrelevance_score = irrelevance(neighbour_index) ;
7   if irrelevance_score is greater or equal to  $\text{floor}(k/2)$  then
8     | boundary_idx.append(k) ;
9   else
10    | if density( $M[i]$ ) is larger than density( $M[\text{neighbour\_index}]$ ) then
11      |   | densest_idx.append(i)
12    | end
13  | end
14 end
15 return np.union1d(boundary_idx, densest_idx) ;

```

---

## 2.4 Trade-off Framework

The main drawback of the instance selection algorithms is that we cannot control how much data to select nor the minimum accuracy<sup>3</sup> that we can accept. Although there is one trade-off framework published by Park et al. [31], it can only work with machine learning algorithms which can be optimised by the maximum likelihood method. Therefore, to work with CNNs, we need to build a new trade-off framework.

In order to build the framework, the primary challenge is to find the relationship between required relative accuracy and the number of training samples needed. However, this is not straightforward because the optimisation of CNNs for classification tasks can be considered as a non-convex problem in most cases [14, p. 114]. It is hard to get the final accuracy without a long time training. Some second-order method could find the zero gradient point, such as Newton's method [40]. However, they are

---

<sup>3</sup>Here we assume that reduce the number of samples would affect the classification accuracy, which is true from our CNN experiments.

still prone to local minimum points and not scale well to large CNNs [14, p. 310]. For the reasons mentioned above, we tend to investigate experimental methods rather than analytical methods.

In 2019, Istrate et al. published a paper aiming at predicting the accuracy without training the networks [19]. They built a Lifelong Database of Experiments (LDE) which stores a huge amount of training experiments on many CNN structures and datasets. When a new dataset is given, the framework TAPAS first train the data with a small probe CNN [35] for 5 epochs. The accuracy is recorded as the Dataset Characterisation Number (DCN). TAPAS then fetches experiments with similar DCN from the LDE. With these history data, a regression model is trained which takes the network structure and DCN as inputs to predict the classification accuracies. Although it is not realistic for us to collect enough data for such a framework, TAPAS provides a working example that it is possible to build the trade-off framework based on experiment histories.

# Chapter 3

## Adapted Data Reduction Methods

In this chapter, we begin by presenting the experimental datasets, CIFAR10 and CIFAR100 along with the image feature extraction process. Next we adapt three methods overviewed in Chapter 2 to reduce the size of image dataset, called the Patterns by Ordered Projections (POP) [33], Enhanced Global Density-based Instance Selection (EGDIS) [28], and Curriculum Learning (CL) [15]. Then we propose our edited data reduction method, called Weighted Curriculum Learning (WCL), based on CL scores and Boundary Based Curriculum Learning (BBCL), based on the EGDIS selected boundary instances. After that, our work is focused on the comprehensive evaluation of the methods. We illustrate the data reduction geometry patterns with three generated datasets, blobs, moons and circles. We then describe the model fitting procedure of the logistic regression algorithm. Finally, we extend the reduction pipeline to deep learning method with the particular network, DenseNet121. Figure 3.1 gives the pipeline overview of this project.

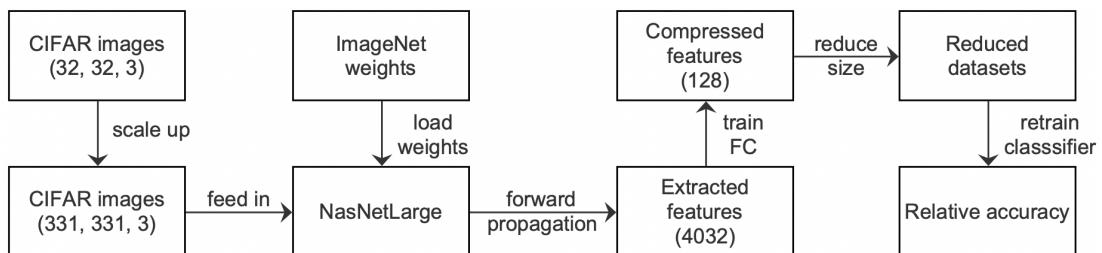


Figure 3.1: Overview of the data reduction pipeline.

### 3.1 Datasets and Image Feature Extraction

We choose to use CIFAR10 and CIFAR100 [23] as our experimental datasets which contain 6,000 and 600 tiny images of size  $32 \times 32$  per class respectively. The advantage of CIFAR is that they are large in the number of images and small in the size of images. With CIFAR datasets, we could train the network faster thus explore the reduction rate for a wider scale within the required timetable. Another advantage is that CIFAR datasets can reflect the performance of data reduction algorithms for both simple dataset and hard dataset in term of classification accuracy. According to Kornblith et al. [22], the test set results indicate that CIFAR10 is very easy to classify and CIFAR100 is as difficult as other high resolution datasets such as the Describable Textures Dataset (DTD) [10], Food-101 [6]. These features could gain us thorough and representative evaluation results with limited compute resources.

After scaling up the image size to 331 and transforming the images into range 0 to 1 by dividing 255, we therefore performed the feature extraction task. The goal was to provide structural data for the data reduction algorithms to work with. We did this job with pre-trained NasNetLarge [42] because Kornblith et al. [22] have evaluated the quality of extracted features and the quality of extracted features is good enough. Their experiments show that the classification scores with simple logistic regression are very close to the state-of-art classification scores with CNN. In order to simplify the implementation, we chose to use the Keras implemented NasNetLarge network from TensorFlow Hub, which is designed to get feature vectors from images [1]. However, the original shape of the feature vector is 4032 and it would take longer time to run the reduction algorithms. To speed up the reduction process, we trained another network with two FC layers. The depth of the first FC layer is 128 and we took the outputs as compressed feature vectors. The test accuracy is also reported as the baseline performance. Figure 3.2 represents the network structure. We also trained a batch normalisation layer after the 128-D FC layer to limit the feature range. This ensured that the Euclidean distance between two vectors wouldn't be dominated by dimensions with wider range.

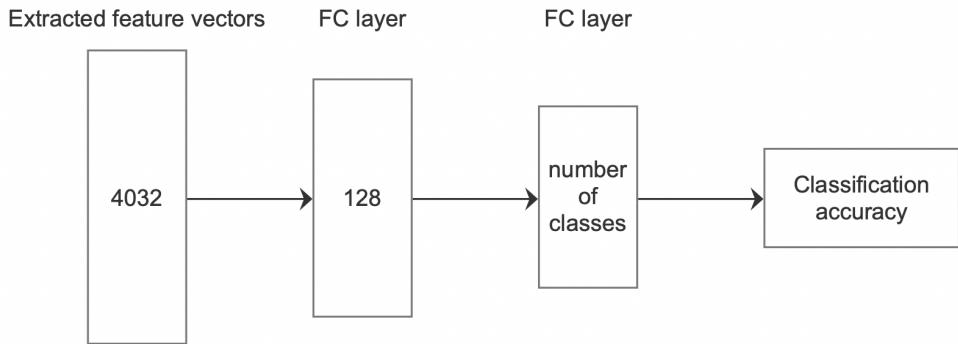


Figure 3.2: Network structure to compress the extracted feature vectors.

## 3.2 Difficulty Tuneable Algorithms

Before presenting the evaluation plans, it should be noted that the algorithms described in 2.2.1.2 are not perfect. First of all, POP and EGDIS are not deep learning based algorithms so the CNN may not work well with selected samples. Also, although CL is target hypothesis based algorithm, by keeping easy samples only may limit the highest performance that the network could achieve. Therefore, our first contribution is to enhance CL with the ability to contain a tuneable proportion of relatively difficult samples. We proposed two variations:

1. Weighted Curriculum Learning, which select samples according to classification scores.
2. Boundary Enhanced Weighted Curriculum Learning, which selects a proportion of the EGDIS boundary samples first then fill in the subset with easier samples.

We followed the requirement in [15] to select balanced subsets.

### 3.2.1 Weighted Curriculum Learning

Instead of selecting the top N samples after sorting the samples in ascending order based on classification scores, we normalised the scores as the survival probability. By dividing the sample score to the sum of all scores, the sum of all normalised scores will be 1 so that we can treat them as probability. In this way, sample with higher classification score would have higher probability to be selected. Therefore, not only the easy samples are selected, some hard examples are also selected. Because we only select the subset once, this behaviour should be able to achieve higher accuracy if

the dataset is harder to classify and the network is powerful to learn from these hard examples. However, if the network is not capable of handling these hard examples, then the test accuracy may decay.

---

**Algorithm 4:** WCL

---

**Data:** compressed 128-D feature vectors  $M$

**Input:** number of samples to select  $m$ , classification score for each sample  
 $scores$ , number of classes  $n$

**Output:** selected sample index by WCL

```

1 selected_idx_list = [] ;
2 foreach class label  $L$  do
3   scores = all sample scores with label  $L$  ;
4   scores = scores / sum(scores) ;
5   idx_list = choose floor( $m/n$ ) samples based on scores ;
6   selected_idx_list.append(idx_list)) ;
7 end
8 return selected_idx_list ;

```

---

### 3.2.2 Boundary Based Weighted Curriculum Learning

While WCL can reflect the difficulty of the datasets, it cannot guarantee to select enough hard examples for the network to mine the pattern. Therefore, we proposed the Boundary Based Weighted Curriculum Learning method to tune the amount of difficult samples in the selected subsets.

---

**Algorithm 5:** BWCL

---

**Data:** compressed 128-D feature vectors  $M$

**Input:** number of samples to select  $m$ , classification score for each sample

$scores$ , number of classes  $n$ , EGDIS boundary sample index list

$egdis\_boundary\_index$ , percent of boundary to select  $p$

**Output:** selected sample index by BWCL

```

1 selected_idx_list = [] ;
2 foreach class label  $L$  do
3   scores = all non-EGDIS sample scores with label  $L$  ;
4   egdis_boundary_index_L = all EGDIS boundary sample index with label  $L$ 
      ;
5   egdis_idx = choose floor( $m/n \times p$ ) samples from egdis_boundary_index_L ;
6   selected_idx_list.append(egdis_idx)) ;
7   scores = scores / sum(scores) ;
8   idx_list = choose floor( $m/n \times (1-p)$ ) samples based on scores ;
9   selected_idx_list.append(idx_list)) ;
10 end
11 return selected_idx_list ;

```

---

### 3.3 Evaluation Designs

We have the following few experiments to evaluate the performance of the chosen three and the two proposed algorithms:

1. We extracted the feature vectors for CIFAR10 and CIFAR100. We visualised the extracted features with t-SNE.
2. We use CIFAR10 to explore the intrinsic behaviour of these methods. We reported the extraction time for both CIFAR10 and CIFAR100.
3. We use logistic regression to test all five methods. We also use the experiment results to decide how to select the CIFAR subsets.
4. We used these four datasets to evaluate them CNN, DenseNet121.

# Chapter 4

## Data Reduction Evaluations

### 4.1 Experiment 1: Feature Extraction

The python version CIFAR10 and CIFAR100 datasets are downloaded from the official website with pre-defined train and test sets. The train set were split into train and validation sets with the ratio 8:2 randomly. We got 40,000 train samples, 10,000 validation samples and 10,000 test samples for both datasets. In order to speed up the experiments, we started by extracting the image feature vectors with a single forward propagation before training the FC layers.

All trainable parameters of the two FC layers were initialised by the Xavier uniform method [12]. The training process was monitored by the validation accuracy. After each epoch, we reported the validation accuracy and kept a record of the model parameters which achieved the best validation accuracy so far. We scheduled two training stages to approach the optimal classification accuracy. First we performed 500 training epochs with step size 0.01 then decreased it to 0.001 for another 500 epochs. After each training stage, the recorded parameters were loaded to prevent overfitting. The final held-out test set accuracy for CIFAR10 is 0.9258 and for CIFAR100 is 0.7444. Finally we compressed the extracted 4032-D feature vectors with the 128-D FC layer.

We further took a closer look at the compressed features by projecting them down to two dimensional vectors with t-SNE [39] and maintains the relative distance between samples. The plots are shown in Figure 4.1. Different colours represent different classes of samples. First of all, the network transformed the images into blobs where ideally samples from the same class should be closer to each other. This is true especially for CIFAR10. The quality of these compact blobs reflects the classification accuracy. Strays far from the cluster would be misclassified by the logistic regression

layer thus lower the classification accuracy. Furthermore, the boundaries between different classes are not always clear. Blobs are close to each other near the boundary thus the classification accuracy is sensitive to slight variations of the boundary. The effect is more obvious if there are more samples for each class. This partly explains the rapid vibration of the CIFAR10 validation accuracy curve shown in Figure 4.2(a). In fact, we can widen the distance between adjacent blobs by fine-tuning the pre-trained feature extraction network and achieve a higher classification score [22]. However, since we want to minimise the pre-processing time required, we omitted this step and stayed with the sub-optimal features.

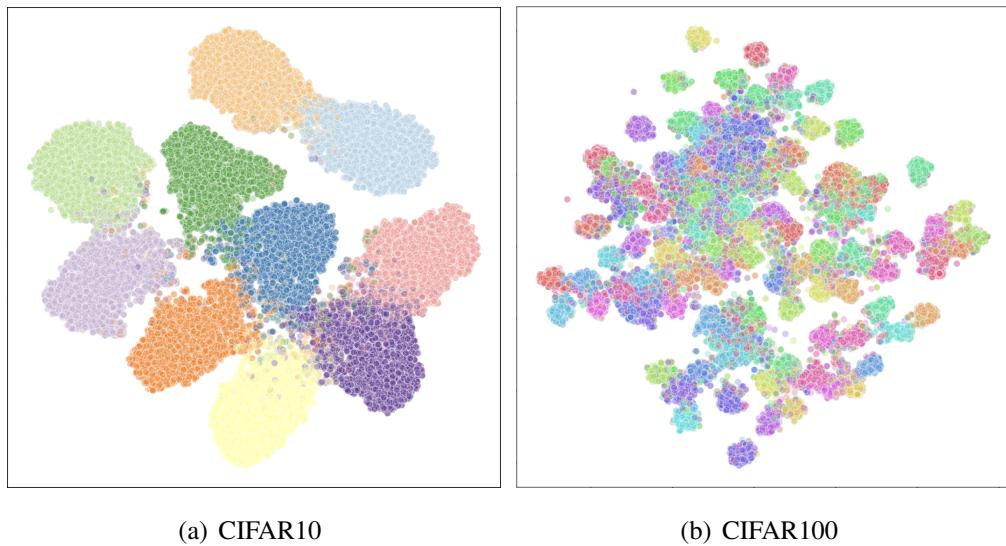


Figure 4.1: Visualisation of extracted features with the t-SNE algorithm.

In addition, we compared the training process of the two datasets in Figure 4.2. Across the two training stages, CIFAR10 converges faster than CIFAR100 but it starts to overfit earlier. The green line indicates that we could reduce the number of epochs in the first training stage by half and leave the mining job at the second stage.

Finally, we also visualised the classification distributions in Figure 4.3. As observed from the quality of blobs plotted in Figure 4.1, CIFAR10 samples tend to achieve higher classification scores around 0.9 while CIFAR100 samples are spread across the range below score 0.9. Samples from CIFAR100 are harder to classify correctly with lower scores. Based on the sample selection procedures described in Chapter 2, we may need more samples for CIFAR100 to achieve a similar relative accuracy compared with CIFAR10. This suggests that data reduction algorithms may be less efficient for datasets with more classes and lower classification accuracy. We will

discuss the algorithm performance further in Section 4.3 and 4.4.

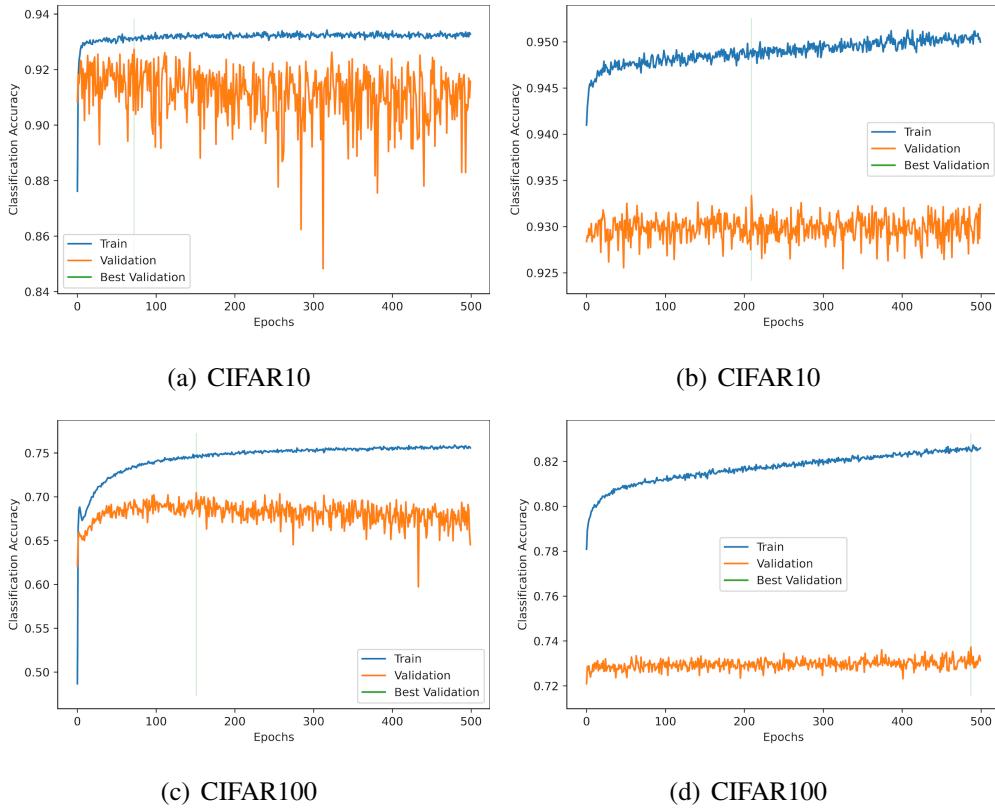


Figure 4.2: The training history of CIFAR10 and CIFAR100. The left column is the first 500 epochs and the right column is the second 500 columns. The green line represents the best validation epoch.

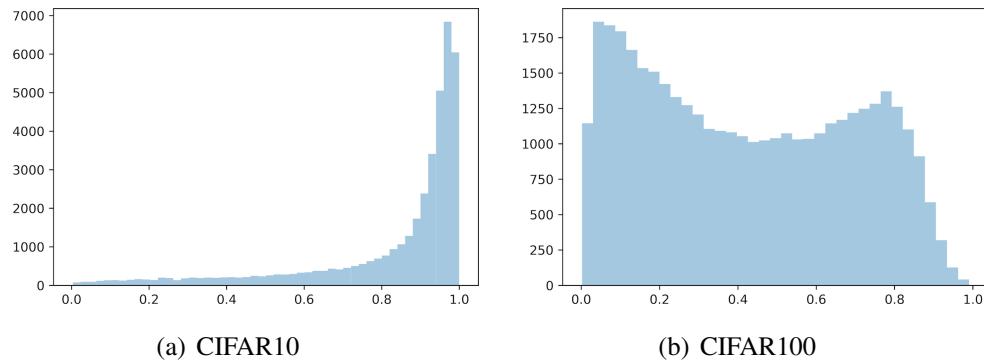


Figure 4.3: Classification score distributions

## 4.2 Experiment 2: Intrinsic Behaviour

Obtaining the discriminative t-SNE plots, we assumed the FC layers to have learnt high quality compressed features. In this experiment, we tried to get an understanding of the intrinsic behaviours of these data reduction algorithms by visualising the selected CIFAR10 samples with red points. We managed to explain the behaviours with the answers to two questions:

1. What are the geometrical distributions of the selected samples?
2. What are the classification score distributions of the selected samples?

For each algorithm, the selection behaviours were further explored by tuning the hyper-parameters. We chose to use CIFAR10 for the reason that the quality of t-SNE projected blobs are more discriminative than CIFAR100 thus we can understand the behaviours easily.

First of all, we present the POP selected vsamples by adjusting the equal tolerance of the *resort\_by\_label* function described in Algorithm 2. For each tolerance choice, we removed only pure inner samples whose weakness values are 128. Figure 4.4 depicts that with proper threshold setting such as  $et = 1$ , we can select most samples along the blob contours as desired. Also, from Figure 4.4(a) and 4.4(b), we observe a trend to select samples closer to the boundary between two adjacent blobs rather than cover the whole contours. This behaviour can guide linear classifiers to build the proper decision boundaries. Another advantage of POP is the fast compute speed. It took us 33.27 seconds on average to get the result. However, POP was designed to process integer features. It is difficult to choose the suitable tolerance value and keep just enough samples. Our experiments indicate that the number of samples far from the boundary decreases much faster than expected by increasing the tolerance value. One possible reason is that we normalised the compressed features so the value range is too compact. This makes POP hard to use for image datasets.

The second algorithm we explored is EGDIS by tuning the hyper-parameter  $k$  of the *knn* classifier. Figure 4.5 presents how the selection pattern varies with  $k$  values 3, 5, and 7. Our first discovery is that EGDIS tends to select all misclassified samples together with samples surrounding them. Compared with Figure A.1, only a few samples still exist with different colours. Also, by increasing the value of  $k$ , fewer inner samples and boundary are selected. To explain these results, we need to know how EGDIS works with hyper-parameter  $k$ . In short, higher  $k$  values require boundary

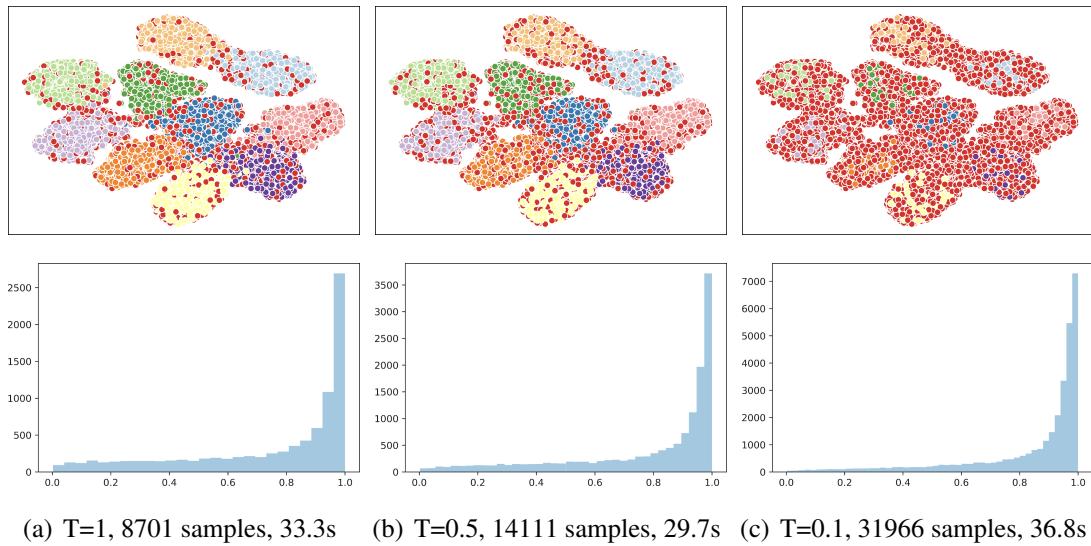


Figure 4.4: POP tune threshold

samples to have more neighbours from different classes. Based on this mechanism, we can infer that most misplaced samples are alone and surrounded by correctly classified samples. For these isolated samples, they would be considered as boundary samples and have enough number of neighbours to be selected. For samples near the boundary, fewer samples will be qualified and only those closely contacted with other blobs would be selected. For dense samples near isolated misplaced points, some of them may be categorised as boundary samples and wouldn't be selected anymore.

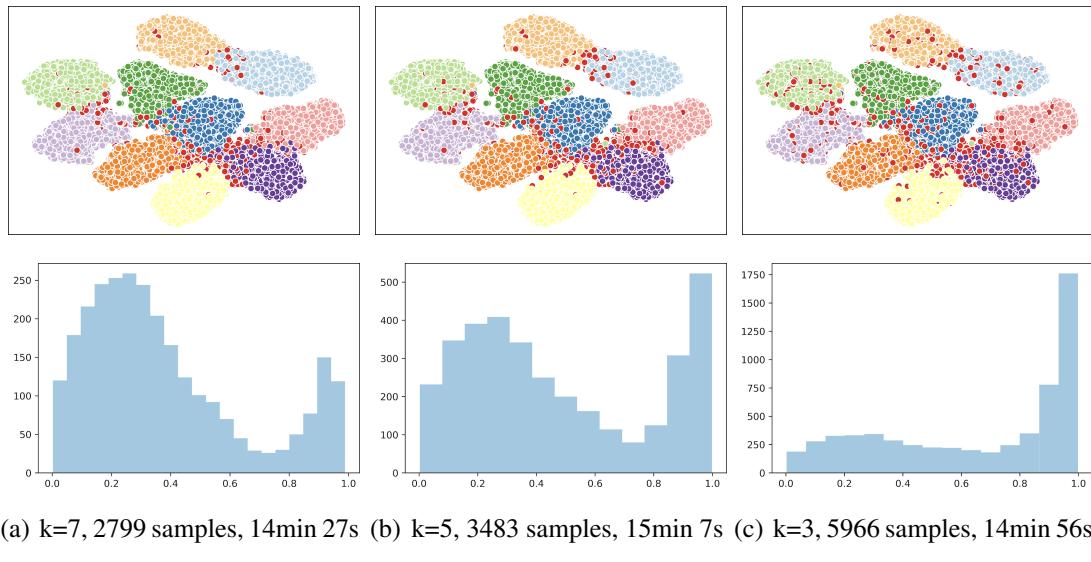


Figure 4.5: EGDIS tune kNN

The drawback of EGDIS is the global density calculation which computes the dis-

tance between all training samples. In our experiments, it took us on average 14 minutes 50 seconds to get all the selected samples and we only need about 12 seconds to select the boundary samples. Figure 4.6 shows the score distribution of EDGIS selected boundary samples with  $k = 3$ . Compared with the second column of Figure 4.5(c), the main difference is the lack of high score samples. Therefore, if we choose to select higher score samples from the score list, we can combine them with EGDIS selected boundary samples and get a similar EGDIS score distribution within about 20 seconds. We will mention this topic in the next few paragraphs.

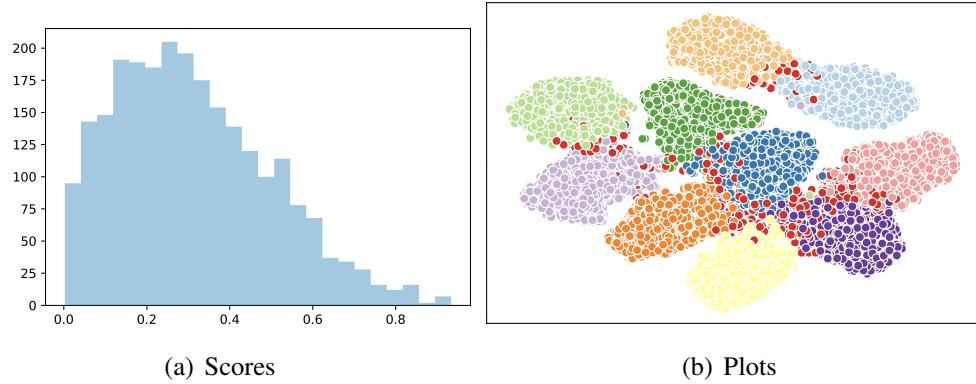


Figure 4.6: EGDIS boundary score distribution

Next, we visualised the selection preference of CL in Figure 4.7. We selected different percentages of samples and the selection pattern is very obvious. We can see that CL tends to select only high score samples, and these samples are lying in the opposite direction as the EGDIS boundary samples. This is what we expected because samples far from the decision boundary would have high scores if the classifier. From the score distributions, we can see that all selected samples are easy to classify for deep learning. This reminds us to combine these high score samples with EGDIS boundary samples. We can construct a limit of the blobs. However, the downside is that no inner samples are selected and the limit borders are not complete. We can not capture the whole shape of the blobs and the results are more like two isolated blobs for each class. With linear regression model, the boundary cannot be recovered precisely thus we may get lower accuracy.

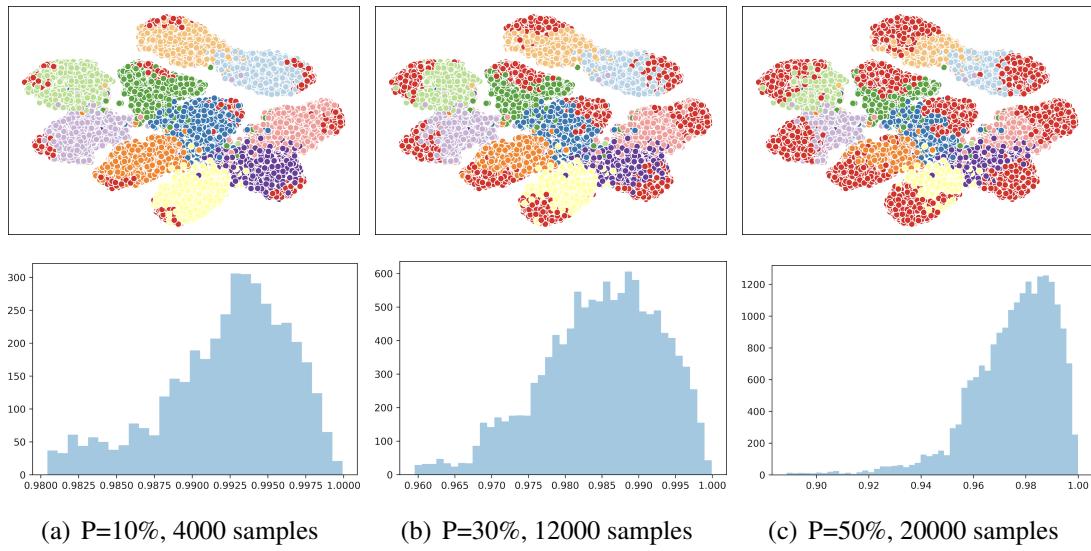


Figure 4.7: CL tune selection percentage

Rather than selecting only the top score samples, we proposed WCL to randomly select samples within each class based on the sample score. The results are shown in the first column of Figure 4.8. Although most samples are selected from the high score region, there are some sample selected from the lower score regions. The problem is that less samples near the EGDIS boundary are selected. The boundary between close blobs are blurry thus we cannot guarantee the performance.

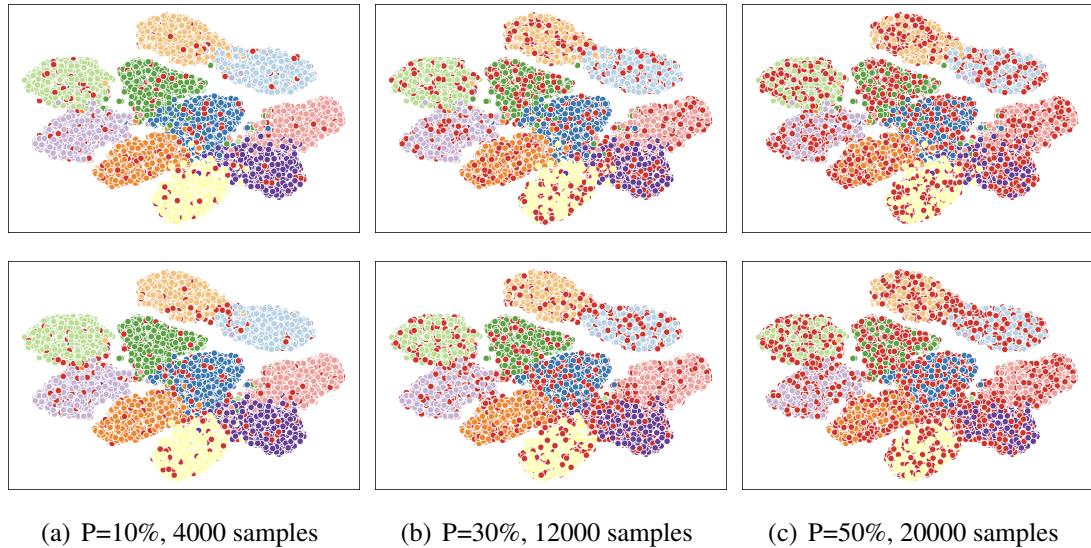


Figure 4.8: WCL tune selection percentage. The first row we use weighted sampling method, the second row we use EGDIS based method.

Therefore, we combined both WCL and EGDIS selected boundary samples and

proposed the method BWCL. Compared with EGDIS, our BWCL can both get the similar score distribution as well as recover the shape of blobs. However, due to randomness during selection, we cannot guarantee to select the same datasets each time. This makes the classification accuracy unstable for machine learning models. We plotted the score distributions in Figure 4.9. We found that the score distribution of WCL is similar with the score distribution of the whole dataset in Figure 4.3 while by selecting similar amount of samples, BWCL is similar with EGDIS in Figure 4.5(c).

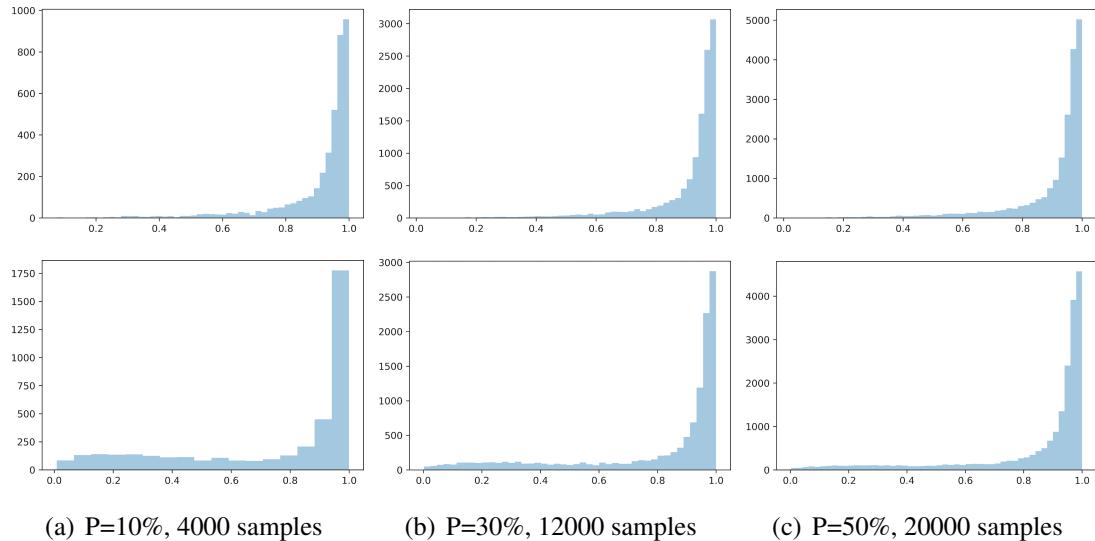


Figure 4.9: WCL tune selection percentage. The first row we use weighted sampling method, the second row we use EGDIS based method.

### 4.3 Experiment 3: Logistic Regression

For more comprehensive evaluation results and inspired by [19], we manually synthesised two CIFAR100 subsets and fill in the classification gap between CIFAR10 and CIFAR100. Between 10 classes to 90 classes with gap 10, we choose to build 9 subsets with by selecting the samples from required number of classes. For each required classes, we randomly do the job 5 times and trained them to convergence with logistic regression model. The boxplot is shown in Figure 4.13. We choose 40 classes and 20 classes as the number of our extra synthesised datasets, with LR test accuracy: 0.78925 and 0.853. They have 8,028 samples and 16042 samples. Then we repeated the feature compression process described before to fine-tuning the selected subsets and compress the extracted features. The final test accuracy is 0.8065 and 0.8745 respectively. In our reported accuracy below, we use CIFAR20 and CIFAR40 to refer to these synthesised

datasets.

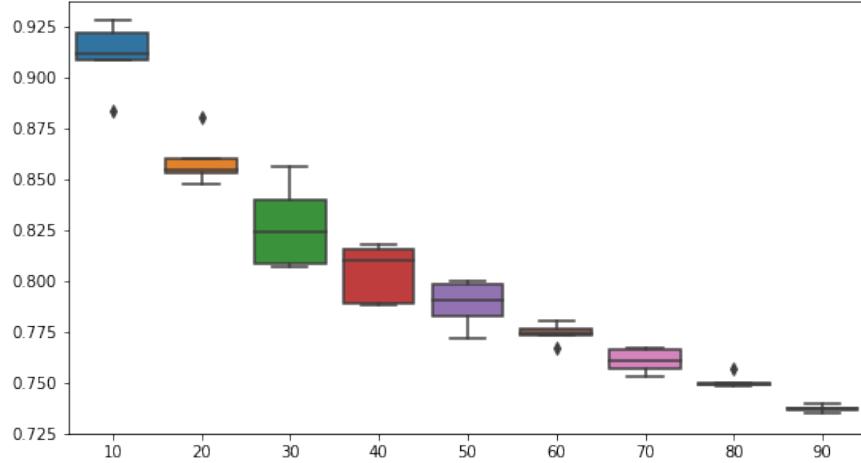


Figure 4.10: The test set accuracies of CIFAR100 subsets. Horizontal axis is the number of classes selected. Vertical axis is the accuracy score. For each selection, we randomly choose the classes for five times and report the accuracy with logistic regression model.

Our first discovery is that it is hard to select right amount of subsets with POP because the number of samples with weakness 128 is very low for all datasets except CIFAR10. Even for threshold 1, the number of pure inner is still low. Therefore, it is not good to choose samples with weakness  $\geq 128$ . From our experiment, we found that the reduction rate for EGDIS is good enough. Therefore, we make POP, CL, WCL and BWCL to select the same amount of samples as EGDIS. For POP, we ranked samples based on weakness and select from low to high. Therefore, we can have a fair comparison of their classification performance. The POP weakness distributions are shown in Figure 4.11.

For the five algorithms, we performed the selection process 12 times and reported the average accuracy with logistic regression model. The relative accuracy is recorded in Table 4.1. We can see that the retention rate is increasing with the classification difficulty of the datasets. POP and EGDIS perform better with simpler dataset such as CIFAR10. For EGDIS, it even achieved an accuracy increase. We think this is because with selected boundary and dense samples only, the overfitting problem is reduced. However, the performances of WCL and BWCL are more stable than EGDIS. Their relative accuracy decreases slowly just like POP. Among all three classification score based algorithms, BWCL is more capable of dealing with harder datasets. We believe the reason is that BWCL can maintain the EGDIS selected boundary and the self-

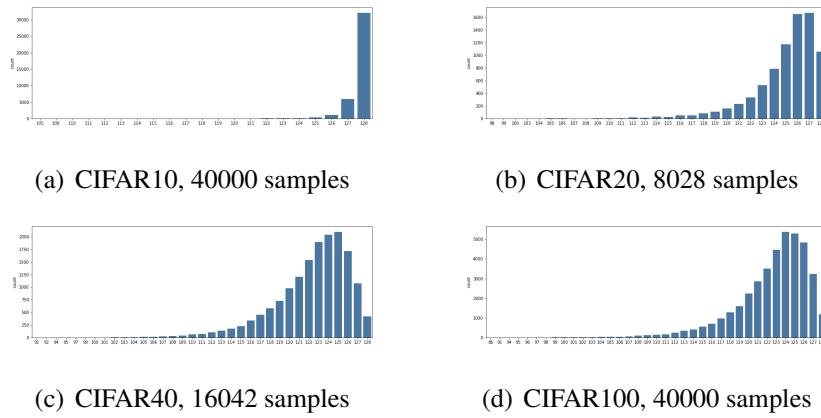


Figure 4.11: POP with 4 datasets.

adaptive selection manner of WCL.

Datasets	Retention Rate	POP	EGDIS	CL	WCL	BWCL
CIFAR10	14.915%	100.00%	<b>100.04%</b>	99.74%	99.96%	99.91%
CIFAR20	16.67%	<b>99.94%</b>	99.01%	99.57%	99.31%	99.43%
CIFAR40	21.09%	<b>99.64%</b>	98.92%	99.09%	99.46%	99.63%
CIFAR100	31.48%	99.38%	97.78%	98.78%	99.46%	<b>99.49%</b>

Table 4.1: Logistic Regression test set relative accuracy by averaging 12 runs

We should notice that the average relative accuracy of BWCL is not the best value. We tuned the maximum proportion of boundary samples from 0.1 to 0.4 with step 0.1 and trained them 3 times each. In Figure 4.12(b), we reported the relative accuracy in plot. It is clear that for simpler datasets, we should choose less hard examples. For harder datasets, we should choose more boundary samples.

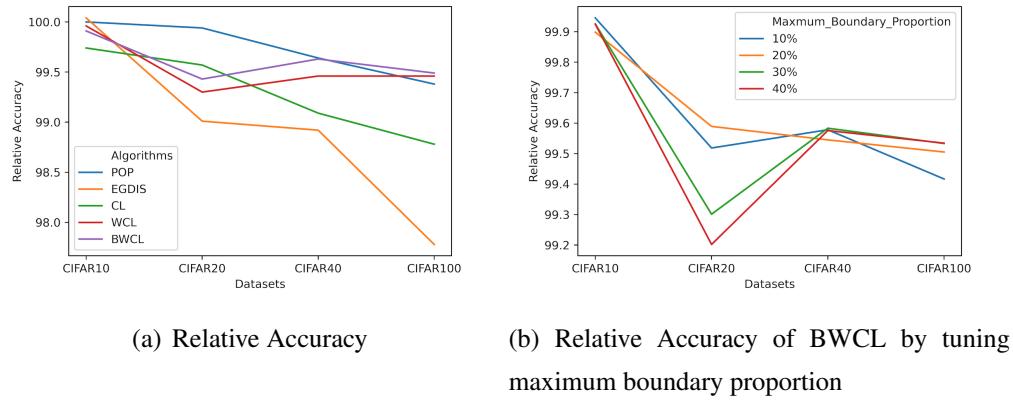


Figure 4.12: Relative Accuracy of data reduction algorithms

We took a further analysis for BWCL, by varying the proportion of samples selected relative to the number of samples selected by EGDIS. We fixed the maximum boundary proportion to 0.4. The relative accuracy is reported in Figure 4.13. We found that the quality of the extracted features are so good that the test set samples are classified easily. This means that the results from logistic regression may not transfer to CNN experiments well but this provide us a good start.

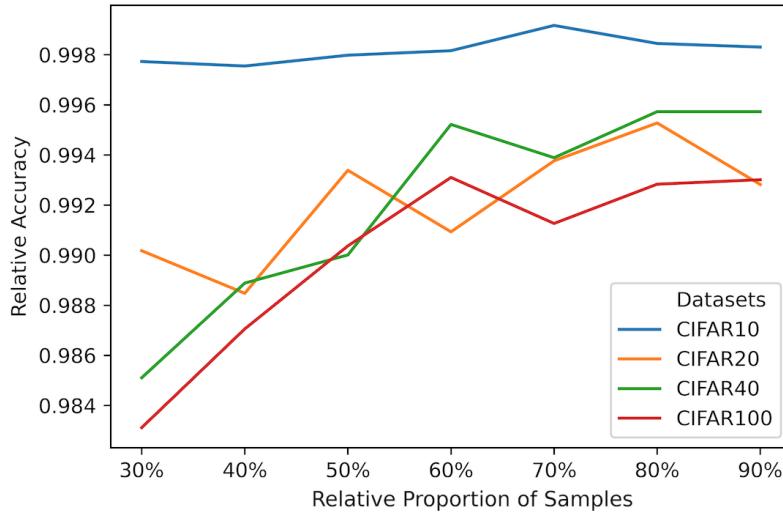


Figure 4.13: The BWCL test set accuracies. Averaged with 3 individual runs. The threshold is 10%. Horizontal axis is the percentage of samples selected, in term of EDGIS selected samples. Vertical axis is the accuracy score.

## 4.4 Experiment 4: Data Reduction for CNN

We trained the network densenet101, three times, with learning rate 0.1 (150 epochs), 0.01 (100 epochs) and 0.001 (100 epochs). First we trained the network with the same selection configuration as logistic regression and reported the classification accuracy in Table 4.2.

Datasets	Retention Rate	POP	EGDIS	CL	WCL	BWCL
CIFAR10	14.915%	86.80%	85.18%	86.71%	<b>88.36%</b>	86.75%
CIFAR20	16.67%	59.93%	61.43%	<b>70.96%</b>	69.34%	64.67%
CIFAR40	21.09%	63.99%	63.89%	<b>75.77%</b>	71.36%	70.65%
CIFAR100	31.48%	75.24%	73.74%	82.72%	<b>83.23%</b>	81.69%

Table 4.2: CNN test set relative accuracy

First of all, we found that POP and EGDIS selected samples are not suitable for neural network. The network cannot extract good features from these images thus the relative accuracy is much lower than expected. Second, classification score based algorithms can achieve relatively higher accuracy. In particular, for datasets with less per-class samples, CL is better. For datasets with more per-class samples, WCL is better. However, BWCL performs worse than expected. We believe that this is because we evaluated the classification scores with NasNetLarge, who has better extraction power than DenseNet. We just contains too much hard examples so the network cannot handle the training samples well. We proposed another hypothesis that if the network could learn these hard samples well, then it should be able to achieve higher classification accuracy based on the hard-example mining method described before.

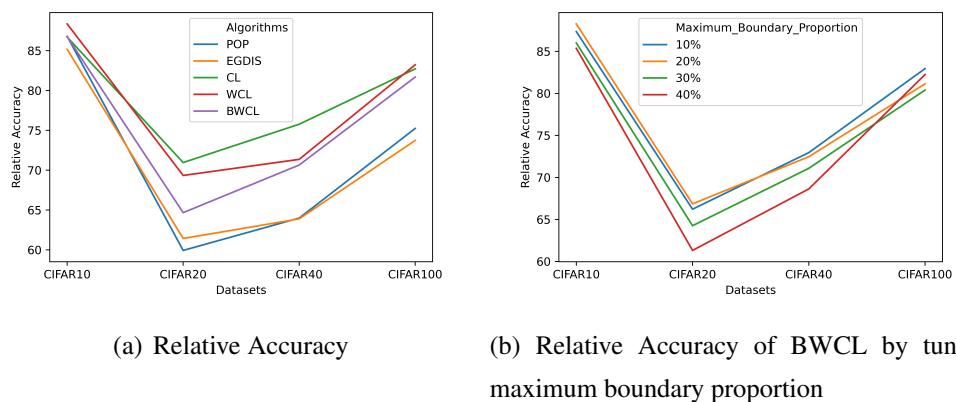


Figure 4.14: Relative Accuracy of data reduction algorithms

We took a closer look at the training curve of the first training stage to analysis the convergence speed. For all datasets, CL selected samples can help the network to converge faster.

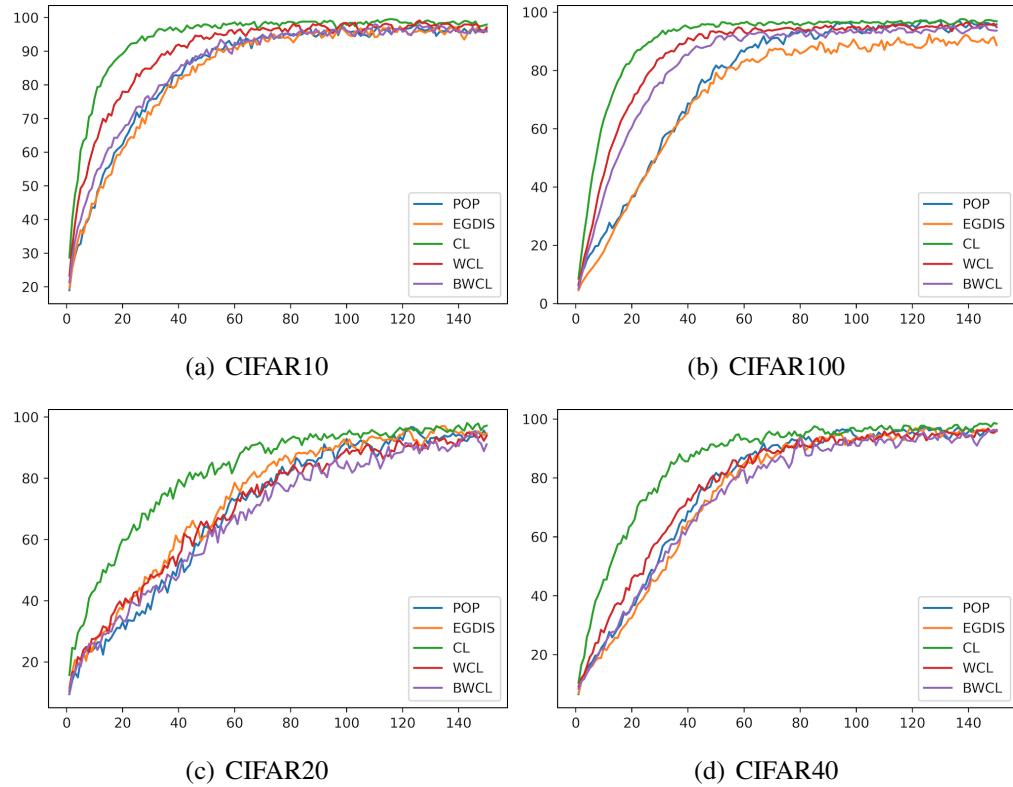


Figure 4.15: The training trend of four datasets by selecting the same amount of samples as EGDIS, of the first 150 epochs

We took a closer look at the relative accuracy achieved by adjusting the proportion of samples selected by these algorithms. It seems that WCL is better than CL and BWCL when dealing with easier datasets like CIFAR10. However, we still have one more problem remains: how to set the right amount of samples to select as a start? We noticed that for all our four datasets, by selecting 50% of the whole samples, we can have a relative accuracy higher than 90%. A heuristic guide is to take 50% of samples and consider it as 90% relative accuracy. Then use linear line to decide the amount of samples to select. To make this more reliable, we took another experiment, by analysing the relationship between validation accuracy history and final relative accuracy. We reported the detail in the next Chapter.

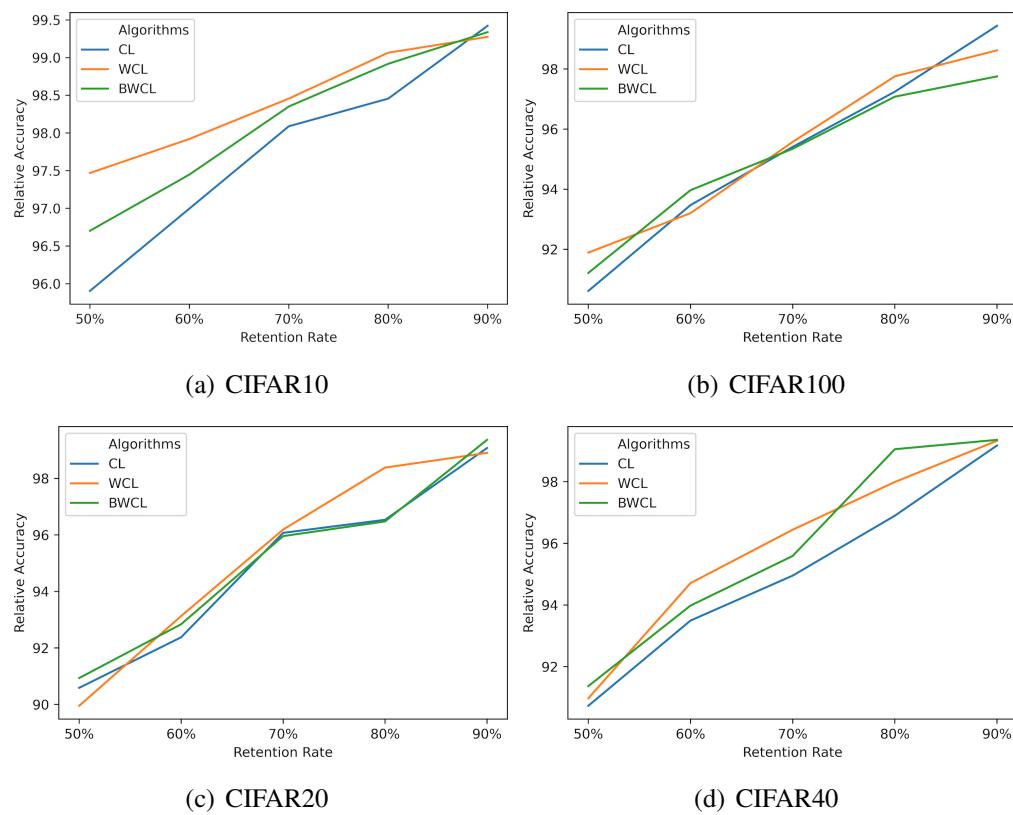


Figure 4.16: The relative accuracy to compare the performance of three algorithms with different retention rate.

# Chapter 5

# Trade-off Framework

According to the trainless accuracy predictor published by IBM [35], the validation accuracy for the first few epochs of a smaller network are linearly related to the final test accuracy. If their experiment is solid, then we could expect it works for relative accuracy as well, for a particular dataset. Also, we want to know if this works for other network, like DenseNet121 as well. We plotted the results in Figure 5.3 with our training history data to analysis this hypothesis. We didn't use CIFAR20 and CIFAR40 history because we want to focus on large datasets.

The result is not exactly linear in our case. The shape is more like part of a arc instead. However, if we take a closer look at the data which achieves a relative accuracy larger than 0.9, then the curve is more close to a linear line, especially for CIFAR10.

## 5.1 Subset Selection Framework

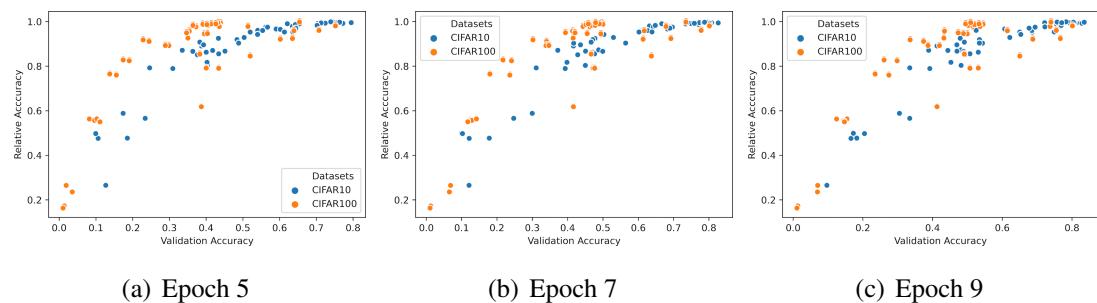


Figure 5.1: The linear relationship between the validation accuracy and the final relative accuracy achieved

We also explored the logit transformation method recommended by [22], who claims that we can have a better linear shape. In Figure 5.3, we can see that this is true. However, this is still not good enough because we also want to link it to the hyper-parameter.

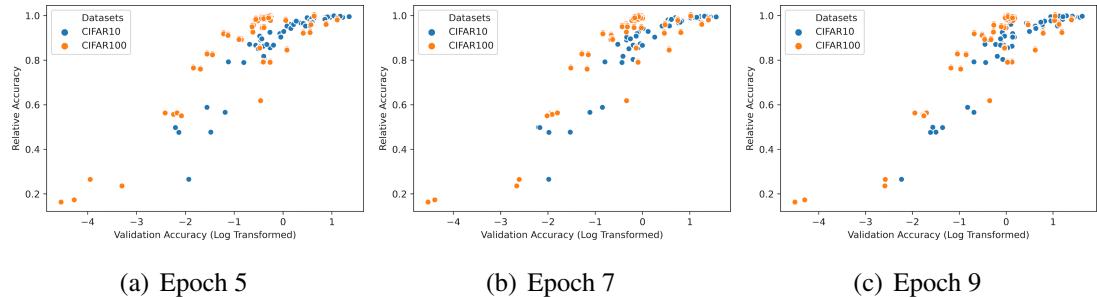


Figure 5.2: The linear relationship between the validation accuracy and the final relative accuracy achieved

Therefore, we trained a simple logistic regression model which takes the first 10 epochs' validation accuracy and the first 5 epochs' whole dataset validation accuracy as the input. By selecting only samples with accuracy larger than 90%, we got a good result shown in Figure 5.3.

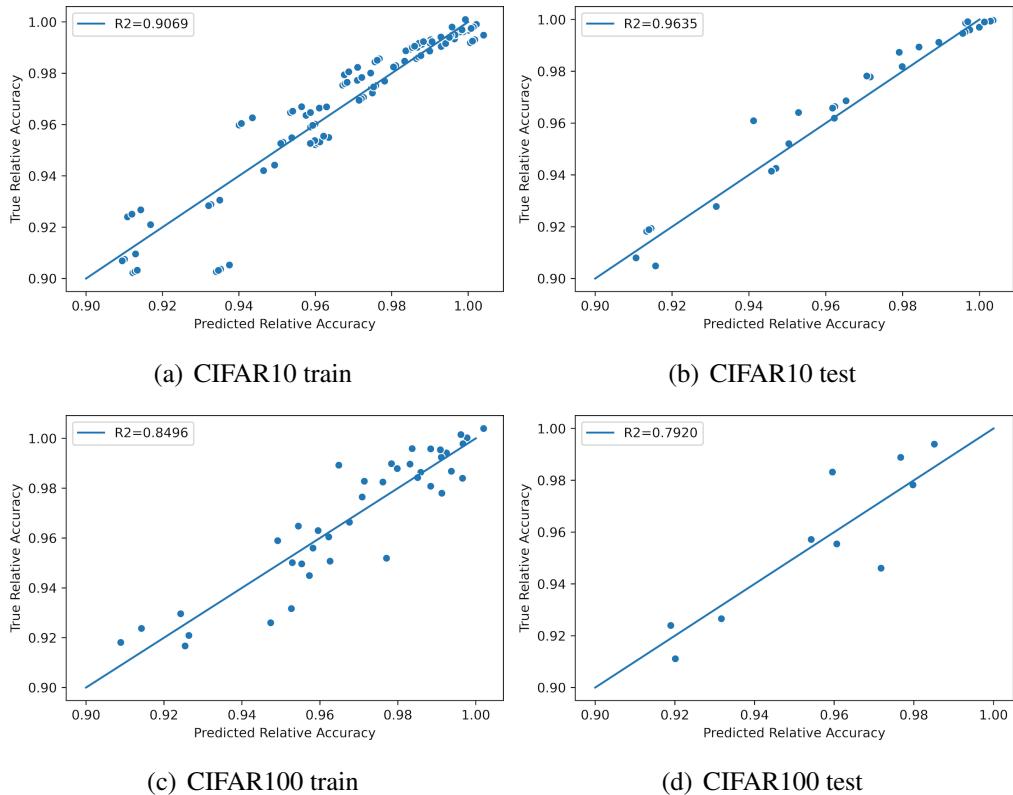


Figure 5.3: The linear model for CIFAR10 and CIFAR100 to predict the relative accuracy

Then we train with subsets selected from CIFAR10 and CIFAR100 for only 100

# **Chapter 6**

## **Conclusion and Future Work**

Moreover, according to the Google Data Labelling Service, it costs 35\$ to label 50,000 images for classification tasks. The price goes to 870\$ for segmentation tasks. It would cost a lot to build a large datasets with millions of images. If we could extend the pipeline to predict the amount of data needed to achieve a accuracy level, we may save the sponsor a huge amount of money.

# Bibliography

- [1] TensorFlow Hub. NasNetLarge.
- [2] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, jan 1991.
- [3] Saleh Albelwi and Ausif Mahmood. Analysis of instance selection algorithms on large datasets with Deep Convolutional Neural Networks. In *2016 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2016*. Institute of Electrical and Electronics Engineers Inc., jun 2016.
- [4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ACM International Conference Proceeding Series*, volume 382, pages 1–8, New York, New York, USA, 2009. ACM Press.
- [5] Vighnesh Birodkar, Hossein Mobahi, and Samy Bengio. Semantic Redundancies in Image-Classification Datasets: The 10% You Don’t Need. jan 2019.
- [6] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 - Mining discriminative components with random forests. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8694 LNCS, pages 446–461. Springer Verlag, 2014.
- [7] Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms, 2002.
- [8] Haw Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 1003–1013, 2017.

- [9] Yushi Chen, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi. Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. *IEEE Transactions on Geoscience and Remote Sensing*, 54(10):6232–6251, oct 2016.
- [10] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3606–3613. IEEE Computer Society, sep 2014.
- [11] Gabriella Csurka, Christopher R Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual Categorization with Bags of Keypoints. In *Workshop on statistical learning in computer vision, ECCV*, pages 1(1–22): 1–2, 2004.
- [12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Journal of Machine Learning Research*, volume 9, pages 249–256, 2010.
- [13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016.
- [14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [15] Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, pages 4483–4496, 2019.
- [16] 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*, volume 2015 Inter, pages 1026–1034, 2015.
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778, 2016.
- [18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. In *Proceedings - 30th IEEE Con-*

- ference on Computer Vision and Pattern Recognition, CVPR 2017, volume 2017-Janua, pages 2261–2269. Institute of Electrical and Electronics Engineers Inc., aug 2016.
- [19] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C. I. Malossi. TAPAS: Train-Less Accuracy Predictor for Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:3927–3934, jun 2019.
  - [20] Angelos Katharopoulos and François Fleuret. Biased Importance Sampling for Deep Neural Network Training. may 2017.
  - [21] Angelos Katharopoulos and François Fleuret. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. *35th International Conference on Machine Learning, ICML 2018*, 6:3936–3949, mar 2018.
  - [22] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2019-June, pages 2656–2666, may 2019.
  - [23] Alex Krizhevsky. Learning multiple layers of features from tiny images. Tech. rep., CIFAR-10 (Canadian Institute for Advanced Research), 2009.
  - [24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
  - [25] M. Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, pages 1189–1197, 2010.
  - [26] Hao Li and Maoguo Gong. Self-paced Convolutional Neural Networks. Technical report, 2017.
  - [27] Ilya Loshchilov and Frank Hutter. Online Batch Selection for Faster Training of Neural Networks. nov 2015.

- [28] Mohamed Malhat, Mohamed El Menshawy, Hamdy Mousa, and Ashraf El Sisi. A new approach for instance selection: Algorithms, evaluation, and comparisons. *Expert Systems with Applications*, 149:113297, jul 2020.
- [29] Deyu Meng, Qian Zhao, and Lu Jiang. What Objective Does Self-paced Learning Indeed Optimize? 2015.
- [30] J Arturo Olvera-López, J. Ariel Carrasco-Ochoa, J. Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods, 2010.
- [31] Yongjoo Park, Jingyi Qing, Xiaoyang Shen, and Barzan Mozafari. BlinkML: Efficient maximum likelihood estimation with probabilistic guarantees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1135–1152, New York, New York, USA, jun 2019. Association for Computing Machinery.
- [32] Vlad Popovici and Jean Philippe Thiran. Face Detection Using an SVM Trained in Eigenfaces Space. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2688:190–198, 2003.
- [33] José C. Riquelme, Jesús S. Aguilar-Ruiz, and Miguel Toro. Finding representative patterns with ordered projections. *Pattern Recognition*, 36(4):1009–1018, apr 2003.
- [34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, dec 2015.
- [35] Florian Scheidegger, Roxana Istrate, Giovanni Mariani, Luca Benini, Costas Bekas, and Cristiano Malossi. Efficient Image Dataset Classification Difficulty Estimation for Predicting Deep-Learning Accuracy. mar 2018.
- [36] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 761–769, 2016.

- [37] Wei Song, Lingfeng Zhang, Yifei Tian, Simon Fong, Jinming Liu, and Amanda Gozho. CNN-based 3D object classification using Hough space of LiDAR point clouds. *Human-centric Computing and Information Sciences*, 10(1):1–14, dec 2020.
- [38] Xunhu Sun and Philip K. Chan. An analysis of instance selection for neural networks to improve training speed. In *Proceedings - 2014 13th International Conference on Machine Learning and Applications, ICMLA 2014*, pages 288–293. Institute of Electrical and Electronics Engineers Inc., feb 2014.
- [39] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.
- [40] Peng Xu, Fred Roosta, and Michael W Mahoney. Second-order optimization for non-convex machine learning: An empirical study. In *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020*, pages 199–207, 2020.
- [41] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 1, pages 1–9, 2015.
- [42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8697–8710. IEEE Computer Society, jul 2018.

# Appendix A

## Training History

### A.1 Feature Extraction for CIFAR20 and CIFAR40

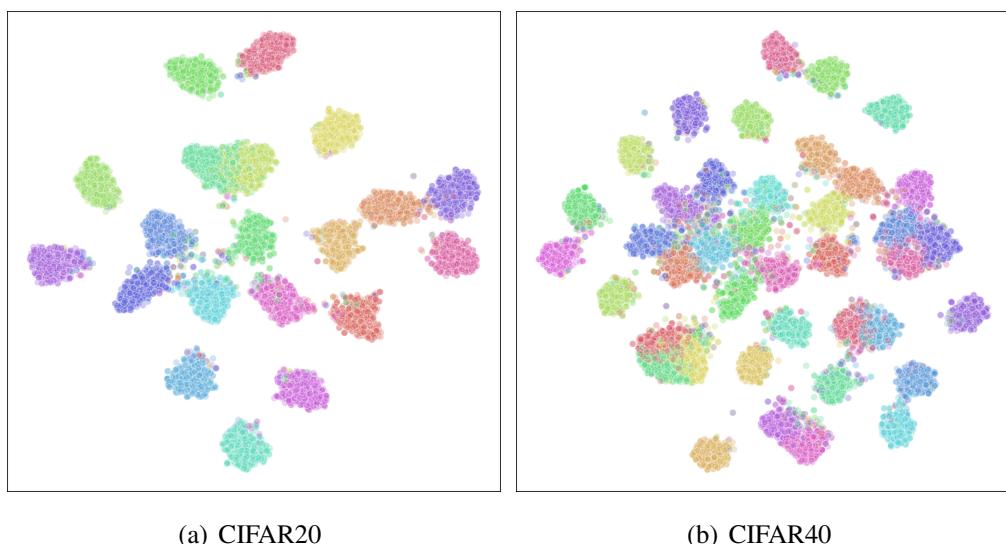


Figure A.1: Extracted features visualisation with t-SNE algorithm

## A.2 Logistic Regression Original History

Datasets	Retention Rate	Whole	POP	EGDIS	CL	WCL	BWCL
CIFAR10	14.915%	0.9258	0.9258	<b>0.9262</b>	0.9234	0.9254	0.9250
CIFAR20	16.67%	0.8825	<b>0.8820</b>	0.8738	0.8787	0.8764	0.8775
CIFAR40	21.09%	0.8159	<b>0.8130</b>	0.8071	0.8085	0.8115	0.8129
CIFAR100	31.48%	0.7444	0.7398	0.7279	0.7353	0.7404	<b>0.7406</b>

Table A.1: Logistic Regression test set accuracy by averaging 12 runs

Datasets	Retention Rate	10%	20%	30%	40%
CIFAR10	14.915%	<b>0.9253</b>	0.9249	0.9246	0.9251
CIFAR20	16.67%	0.8783	<b>0.8789</b>	0.8758	0.8744
CIFAR40	21.09%	0.8125	0.8122	<b>0.8134</b>	0.8132
CIFAR100	31.48%	0.7401	0.7407	<b>0.7410</b>	<b>0.7410</b>

Table A.2: Logistic Regression BWCL test set accuracy by averaging 12 runs

## A.3 CNN Original History

Datasets	Retention Rate	Whole	POP	EGDIS	CL	WCL	BWCL
CIFAR10	14.915%	0.9522	0.8265	0.8111	0.8257	<b>0.8414</b>	0.8260
CIFAR20	16.67%	0.8660	0.5190	0.5320	<b>0.6145</b>	0.6005	0.5600
CIFAR40	21.09%	0.8233	0.5268	0.5260	<b>0.6238</b>	0.5875	0.5816
CIFAR100	31.48%	0.7842	0.5900	0.5783	0.6487	<b>0.6527</b>	0.6406

Table A.3: CNN test set accuracy

Datasets	Retention Rate	10%	20%	30%	40%
CIFAR10	14.915%	0.8318	<b>0.8405</b>	0.8189	0.8128
CIFAR20	16.67%	0.5735	<b>0.5790</b>	0.5565	0.5310
CIFAR40	21.09%	<b>0.5953</b>	0.5913	0.5800	0.5600
CIFAR100	31.48%	<b>0.6505</b>	0.6363	0.6305	0.6450

Table A.4: BWCL test set accuracy