Incremental Learning With Sample Generation From Pretrained Networks

by

Rishabh Patil

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Sethuraman Panchanathan, Co-Chair
Hemanth Venkateshwara, Co-Chair
Troy McDaniel

ARIZONA STATE UNIVERSITY
May 2020
ABSTRACT

In the last decade deep learning based models have revolutionized machine learning and computer vision applications. However, these models are data-hungry and training them is a time-consuming process. In addition, when deep neural networks are updated to augment their prediction space with new data, they run into the problem of catastrophic forgetting, where the model forgets previously learned knowledge as it overfits to the newly available data. Incremental learning algorithms enable deep neural networks to prevent catastrophic forgetting by retaining knowledge of previously observed data while also learning from newly available data.

This thesis presents three models for incremental learning; (i) Design of an algorithm for generative incremental learning using a pre-trained deep neural network classifier; (ii) Development of a hashing based clustering algorithm for efficient incremental learning; (iii) Design of a student-teacher coupled neural network to distill knowledge for incremental learning. The proposed algorithms were evaluated using popular vision datasets for classification tasks. The thesis concludes with a discussion about the feasibility of using these techniques to transfer information between networks and also for incremental learning applications.
DEDICATION

Dedicated to my friends and family, who have always supported me in every venture.
ACKNOWLEDGEMENTS

Sincere thanks to my supervisory committee for their valuable guidance and to all the professors and students I have learned so much from over these 4 semesters.
# TABLE OF CONTENTS

<p>| LIST OF TABLES | vi |
| LIST OF FIGURES | vii |
| <strong>CHAPTER</strong> | |
| 1 INTRODUCTION | 1 |
| 1.1 Goals and Motivations | 2 |
| 1.2 Contributions | 2 |
| 1.3 Thesis Outline | 3 |
| 2 OVERVIEW | 5 |
| 2.1 Incremental Learning | 5 |
| 2.1.1 Data Dependent Methods | 7 |
| 2.1.2 Generation Based Methods | 8 |
| 2.1.3 Data Independent Methods | 9 |
| 3 GENERATION FROM PRETRAINED MODELS | 10 |
| 3.1 Generative Methods | 10 |
| 3.2 Proposed Method | 11 |
| 3.3 Training | 11 |
| 3.3.1 Random Initialization | 12 |
| 3.3.2 Loss Functions | 12 |
| 3.3.3 Dropout | 13 |
| 3.4 Experimental Analysis | 15 |
| 3.4.1 Visualizations | 16 |
| 3.4.2 Results | 21 |</p>
<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5 Conclusions and Future Scope</td>
<td>22</td>
</tr>
<tr>
<td>4 INCREMENTAL LEARNING IN DEEP CLUSTERING NETWORKS</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Deep Clustering and Hashing Networks</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Introduction</td>
<td>23</td>
</tr>
<tr>
<td>4.2.1 Training Process</td>
<td>25</td>
</tr>
<tr>
<td>4.2.2 Experiment Setup</td>
<td>28</td>
</tr>
<tr>
<td>4.2.3 Results</td>
<td>28</td>
</tr>
<tr>
<td>4.3 Conclusions and Summary</td>
<td>32</td>
</tr>
<tr>
<td>5 KNOWLEDGE TRANSFER USING GENERATIVE METHOD</td>
<td>33</td>
</tr>
<tr>
<td>5.1 Proposed Method</td>
<td>33</td>
</tr>
<tr>
<td>5.2 Process</td>
<td>34</td>
</tr>
<tr>
<td>5.2.1 Sample Generation from Teacher</td>
<td>34</td>
</tr>
<tr>
<td>5.2.2 Sample Generation from Student</td>
<td>35</td>
</tr>
<tr>
<td>5.2.3 Teacher Querying and Student Update</td>
<td>35</td>
</tr>
<tr>
<td>5.2.4 Co-generation by Teacher and Student Networks</td>
<td>35</td>
</tr>
<tr>
<td>5.3 Experimental Analysis</td>
<td>36</td>
</tr>
<tr>
<td>5.3.1 Results</td>
<td>37</td>
</tr>
<tr>
<td>5.4 Conclusions and Summary</td>
<td>40</td>
</tr>
<tr>
<td>6 CONCLUSION</td>
<td>42</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>45</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>4.1</td>
<td>Accuracies of Deep Clustering Network on CIFAR and CIFAR100</td>
</tr>
<tr>
<td>5.1</td>
<td>Accuracies During Cyclic Training of Student Network</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.1</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>19</td>
</tr>
<tr>
<td>3.6</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>39</td>
</tr>
</tbody>
</table>
This thesis discusses the methods in which we can transfer and preserve knowledge to incrementally train a neural network. The problem of Incremental Learning can be understood as the need to update a neural network with some new knowledge we may want to incorporate. Let’s look at a simple example, Suppose we want to build a massive classifier capable of identifying all kinds of animals we have on earth. We have several images for each type, so it’s simply a matter of training resources and time, before we have a well-performing deep neural network. However, it is very well possible that a few years down the line, we come across a new species found only on a small island somewhere in the Pacific Ocean. As of now, our model is simply not capable of identifying this being. The simplest way of solving this would be to take a new neural network, increase the number of classes by one and train it from scratch while including this discovery. But that would mean a whole lot of computation again and the previous model being completely thrown away.

An efficient solution would retain the information learned by the trained model and update in such a way that it works well for all the types of animals, old and new; while also saving on a whole lot of computation.

Incremental Learning algorithms usually start with a base number of classes and add a few classes at a time. In most approaches, the model trained on base classes is updated to meet the new requirements. The main challenge here is to prevent “catastrophic forgetting” of the older classes, which is a very common problem when neural networks are retrained on new tasks. At the same time, it is important that the network is not biased and gives good performance for both old and new classes.
This thesis explores new methods for training and retaining knowledge for lifelong learning.

1.1 Goals and Motivations

The goal of this thesis is to propose incremental training and knowledge distillation methods for image classification problems in computer vision.

The motivations are highlighted below.

1. Explainability: Despite the immense progress in the area of deep learning, we cannot explain the outputs of a neural network. Although we can traceback the activations and outputs for every single neuron, yet its not possible to say that some parameters are “correct” or ”wrong”. One of the experiments in this thesis tries to work around this complexity and try to bring to light what a neural network expects to see in an input.

2. Catastrophic Forgetting: It is a long-known issue with neural nets, that when we take a pretrained neural network and optimize for a different task, the performance of neural network drops drastically for the old task. This challenge if surmounted will open up a plethora of possibilities only one of which happens to be Incremental Learning.

1.2 Contributions

The contributions of the thesis are as follows.

1. A new sample generation method is proposed which attempts to utilize the information stored in a pretrained network to generate samples for specific targets.
2. Building on the generation method, a new knowledge distillation algorithm is proposed. Samples generated from both the teacher and student models are used to bring these models to a consensus.

3. Another continual learning approach inspired by deep hashing, attempts to cluster samples and exploit the cluster sphere similarity to incrementally train models.

1.3 Thesis Outline

The thesis is structured in the following manner.

**Chapter 2** provides an overview of Incremental Learning. It describes the problem in detail, the use cases where its applicable in real life, usual constraints associated with it. It also touches upon closely related areas like knowledge distillation and generative methods. It includes a literature review of recent research and an analysis of the commonalities in these techniques.

**Chapter 3** describes in detail a sample generation method that only uses the parameters from a previously trained neural network. Unlike GANs it does not rely on any kind of external feedback from another network. The chapter explores different methods of parameter estimation and how different hyperparameters impact the quality of samples generated. It includes images generated for some popular datasets while using state of the art architectures.

**Chapter 4** explores an alternate way of classifying images similar to deep hashing. These models attempt to bring samples into an encoding space where similar points are grouped and dissimilar samples are pushed away from each other. This setup is very promising for incremental learning, because we have access to seemingly infinite points with a cluster sphere which can be used to generate new samples. This chapter
explores the performance of clustering techniques and whether this is feasible.

**Chapter 5** introduces a recursal based knowledge distillation algorithm which builds upon the sample generation introduced in the previous chapter. This chapter briefly describes the usual setup with a teacher and a student network. The different proposed methods use samples generated by both the teacher as well as the student. One approach makes use of a label correction loss based on the incorrect labels assigned by the student network. Another approach introduces co-generation of samples by teacher and student networks, and these samples are reused to update the student’s parameters.

**Chapter 6** concludes this document by summarizing the results of experiments and the contributions of this thesis.
Chapter 2

OVERVIEW

This thesis attempts to tackle problems in Incremental Learning and Knowledge Transfer. The methods proposed and experiments undertaken build upon previous research and proven techniques. This chapter contains a literature survey of previous publications and analyses different types of approaches and how they align with my motivations and line of thought.

2.1 Incremental Learning

Incremental Learning studies the problem of learning from an ever increasing amount of data, with the purpose of gradually incorporating new tasks with strong performance while maintaining knowledge of older tasks. Incremental Learning is also known as continual learning, lifelong learning or sequential learning. The basic principle is that the proposed learning algorithm should be capable of learning from data in a sequential manner. And the most important obstacle for incremental learning is catastrophic forgetting which is the interference caused by training on new data which diminishes the knowledge of previously learned data. This thesis primary focuses on using Incremental Learning for Classification tasks, this is because classification is one of the most extensively studied areas of deep learning, elegant techniques have been researched which are simple to use and result in very good performance.

This is a along standing challenge in deep learning, with very little success towards building unforgetting neural networks. As a result researchers attempt to break down the problem into small parts by making certain assumptions and constraints.

The incremental learning task setting deals with series of incoming tasks, \( t_1, t_2, \)
Each task $t_i$ consists of $X(i)$, a set of data samples and $Y(i)$, set of class labels for these samples. The aim is to minimize the sum of calculated losses for all previously seen tasks:

$$L(M(X(i); \theta), Y_i)$$ (2.1)

Where n refers to the number of total tasks seen so far, M is the network output for task t and \( \theta \) refers to the model parameters.

$$\min_{\theta_n} = \frac{1}{n} \sum_{t=1}^{n} L(M(X(i); \theta), Y_i)$$ (2.2)

This equation summarizes the task of incremental learning i.e. The goal is to find the optimal $\theta_n$ that minimizes the average loss across all n tasks.

Catastrophic interference has been a challenge ever since the beginning of deep learning research. There have been multiple studies about this, Robins (1995) studies the potential rehearsal mechanisms to prevent catastrophic forgetting. There have been other approaches like reducing the feature overlap by French (1992) and Sloman and Rumelhart (1992). More recently Srivastava et al. (2014) studied the effect of dropout regularization on model over-fitting and representation overlap. Bengio et al. (2013) also studies the effect of dropout while training on two separate tasks.

Comparatively incremental learning is more recent area of research, but it has brought a new light on this problem. Continual learning problems today have a larger dataset with more complexity. Some researchers choose to store a part of the training data set, to simplify the problem a bit. Here I distinguish previous work based on reliance on original training data:

* Data Dependent Methods
• Generation Based Methods

• Data Independent Methods

Although this thesis focuses on Generation based methods, its important to compare the incremental training step itself and also compare the trade-off caused between performance and reliance on old data. Further subsections delve deeper into each of these categories, different experiments and their results.

2.1.1 Data Dependent Methods

Some approaches to solve incremental learning choose to store a small portion of the original data-set also called as episodic memory. This memory is used to limit forgetting of their respective classes during training of classes. Inclusion of newer tasks means that this memory needs to be updated with some new samples.

Rebuffi et al. (2017) do this by limiting the number of samples stored in memory, they introduce an priority algorithm to pick the top-n samples which best represent the chosen class. In their approach, for a memory space of $P$ samples, each class is allotted $P/C$ where $C$ is the number of total classes, this prevents any data imbalance or bias between old classes. A fixed memory method also means that when a new task is added, some samples from the memory need to be discarded, the priority algorithm is reused when this happens.

Chaudhry et al. (2019) explore a method where they store a small episodic memory, but their training process updates this memory after every model update. This means that while training for a new task, each iteration can add new samples to the memory, which in turn affects the next iteration. Their results suggest that this brings a faster rehearsal of older data. The authors use a random sampling method to update the memory.
Isele and Cosgun (2018) introduce memory rehearsal approach for deep reinforcement learning. They improve upon the standard FIFO buffer approach by introducing selective replay by favouring specific experiences like reward, surprise, matching training distribution and coverage of samples. Their results show that when matching training distribution is preferred the performance on older data is least affected.

2.1.2 Generation Based Methods

One of the earliest suggested methods of countering catastrophic forgetting was to generate artificial samples from the original data distribution. This is often referred to as pseudo-rehearsal or pseudo recursion. These samples are then added to the mini-batch along with new task data. This results in preventing old class knowledge loss.

Recent work by Shin et al. (2017) makes use of GANs (Goodfellow et al. (2014)) to train a generator along with a classifier. At the arrival of a new task, this generator is used to generate sufficient data samples to accompany the new task data. Its important to note that the new task also needs to be incorporated into the generator. This results in exceptional performance on all tasks, historical and recent. Its important to keep in mind the computation involved with training and evaluating from the generator at each step.

Atkinson et al. (2018) (and Atkinson et al. (2019)) focus on generating only crucial features for pseudo-rehearsal. It builds on the method generation method mentioned earlier by introducing another discriminator which discriminates between intermediate activations. This forces the generator to prioritize the features which are seen in real data samples. The researchers of this paper experiment with these techniques for a reinforcement learning problem of playing ATARI games, and notice considerable improvement.
2.1.3 Data Independent Methods

One of the most fundamental contributions in incremental learning was Learning without Forgetting (LwF) by Li and Hoiem (2016). LwF built on a lot of older approaches like fine-tuning, joint-training, feature extraction, network expansion to build a simple and elegant technique which minimized the performance loss on older tasks. Compared to other technique LwF has lower performance on newer tasks, but it forms a very good baseline for comparison.

Kirkpatrick et al. (2016) introduced a novel idea of preventing changes in weights which are vital for previous tasks. They do this by using a loss term which minimizes the difference in weights from the the previous state, this is introduced as a quadratic term, and hence behaves as a spring anchoring to the old parameters, which performed well on old tasks. This results in a more efficient usage of all parameters and prevents forgetting of older tasks.

Some methods train a new network for the combined old + new tasks, instead of updating the current one. This prevents the network from being biased towards earlier tasks. Chen et al. (2016) makes use of knowledge transferring techniques to transfer old task information from the older network to the new one. The newer network is modified to be wider or deeper to incorporate more information or complex features coming from the new task data. Encoder based Lifelong learning (ELL) by Triki et al. (2017) also has a similar approach, except the network architecture is not modified. Activations at a central layer are used in a code loss to ensure that extracted features remain as similar as possible between the old and new network. This need not be permanently stored, hence this method is not storage dependent (apart from the memory required to store the model parameters.)
Chapter 3

GENERATION FROM PRETRAINED MODELS

As research in deep learning moves forward, we see novel ideas and application being developed. Ever since Generative Adversarial Networks were introduced, a completely new direction of research has opened up, some very exceptional papers like CycleGAN have come about. Scientists have found ways to incorporate GANs to help improve previous models. In this chapter, I propose a method of generating samples using a pretrained classifier.

3.1 Generative Methods

Generative Techniques focus on designing deep learning techniques with the goal of creating unique data samples which belong to a target data distribution. There are many well performing algorithms in NLP, Speech Synthesis and Computer Vision space. One of the most popular methods being the Generative Adversarial Networks paper by Ian Goodfellow et al. GANs have two specialised neural networks which need to be trained together to achieve the ability of generating exceptional images. But unlike GANs, in this chapter we look at a process which only depends on a pretrained classifier and the whatever information has been learned by its parameters. This method is inspired from previous work in style transfer, which shows how gradients on input samples can be used to control style and content of an image and gradient visualizations.

The following sections outline the generation algorithm, the training process and visualize the results from a few datasets.
3.2 Proposed Method

We focus on using the information stored within a trained classifier to help generate samples for its target classes. To ensure variation, we start with a image initialized with uniform noise. At this point it is very likely that the classifier will have very low confidence classifying it into any class. This image can be considered to be a point in a very high dimensional space (784 for MNIST), we can also plot cross entropy classification loss in this high dimensional space. The goal is to update the image such that it achieves a high classification score for a target class. This can be solved similar to a general gradient descent problem. Its differs slightly from classifier training since all the weights and biases are frozen and the parameters to be optimized are the image pixel themselves.

![Figure 3.1: Generation from Pretrained Network](image)

3.3 Training

The training procedure for the generative method proposed above is described in this section. It includes analysis of input sample optimization, loss functions,
hyperparameters tuning.

3.3.1 Random Initialization

Our goal is to create samples which belong to the target data distribution. We prepare the initial sample using uniform noise, with the same value bounds that the train dataset was normalized to. This is to make sure that activations generated are as close as possible to a genuine image from the original data.

On the other end of the network we need a target that the generated image should classify as. For the sake of simplicity we used hard labels, i.e. the expected confidence for one class is '1' and the rest is '0'. Also, when generating multiple samples in a batch, the number of samples is kept equal. Once the initial image and its target are ready, we can move to the next step.

3.3.2 Loss Functions

During the training of a classifier the goal is to predict the input images as best as possible, which leads to choosing categorical crossentropy metric which is shown to be a excellent classification loss. But in this problem, the goal is to create images which get best classified into the target category. Hence, there are a number of options that can be tried out. The simplest loss could be Root Mean Squared Error (RMSE) since our primary aim is to bring gen prediction to be as close as possible to the target label. When using soft labels, binary crossentropy would be a strict measure to ensure that the soft labels for non-target class are also weighted equally. In the experiments conducted here we stick to crossentropy to keep it as similar to the intial training setup as possible.
3.3.3 Dropout

We often use dropout in deep learning models to ensure that models don’t overfit and the learning is split properly across all the neurons. Eventually after training is complete, dropout is no longer used i.e. the entire network is used to make predictions. During the generation process I decided to keep dropout, since it will have two benefits. Firstly, using dropout is another way of ensuring variations in the generated samples. Secondly and more importantly, after using dropout, we get a different effective neural network during every forward pass, this can been seen as a way of using multiple neural networks all proficient in the same task, to generate samples for a target class. Although this may not perform exactly as well as using actual different neural networks, doing so will be going against the primary motivation of finding an efficient generation process.

3.3.4 Hyperparameter Tuning

Using different hyperparameters might affect the speed and/or quality of generation. Some important parameters that should studied are:

- Learning Rate: Apart from the usual risk of gradient explosions, there is also a significant chance of random samples being spawned near suboptimal local minima, higher learning rates might help jump out of these and generate better samples.

- Optimizer: Stocastic Gradient Descent is the safest option to generate samples. But it may take way too long to bring loss down to a expected value and keeping in mind the goal of discovering a efficient method it might be beneficial to try using momentum, adam, adagrad optimizers.
Algorithm 1 Generation from Pretrained Classifier

**Input:** model $M$, number of classes $C$, iterations $e$, input dimension $p$

**Output:** Generated Samples $I$, Targets $T$.

1: For a batch size say $b$, we generate $b/C$ samples per class.
2: Initialize image matrix $(I)$ of shape $(b, p, p)$ with uniform noise.
3: Prepare targets for this batch $T$ of shape $(b, C)$. Where each target is a one hot vector.
4: for $i = 1$ to $e$ do
5:    Compute model output, $out = M(I)$.
6:    Calculate loss $L = \text{loss}\_\text{function}(T, out)$.
7:    Compute gradients over the inputs, $dL/dI$
8:    Update $I$ with the chosen learning rate and optimizer
9:    If change in loss is less than some tolerance $t$, break loop.
10: return $I, M(I)$.

3.3.5 Generation Process

The generation process is described in a step wise manner in Algorithm 4. It is important to note some things here:

- **Early Stopping:** Some random initialization may descent faster to a local minima than other ones. In which case it might be better to avoid more iterations and over detailing the image.

- **Soft Outputs:** Similar to classification tasks, the output label of a sample is very unlikely to be a perfect one hot vector. The confidence for one label might be significantly higher than the others, yet if this data is to be reused anywhere it should be accurate, hence the algorithm returns the final model output $M(I)$.
as opposed to returning $T$.

- **Regularization**: Adding some L2 decay to the input image might help with gradient descent since the problem is dealing with a function in very high dimensional space. Yet too much regularization might result in the image moving away from the mean of the data distribution expect or worse might completely diminish each sample.

### 3.4 Experimental Analysis

To test the proposed generation method, I chose some popular datasets that have been studied well in classification tasks. Starting with MNIST(LeCun and Cortes (2010)) and FashionMNIST(Xiao et al. (2017)) which are simple datasets and it should be quite straightforward to test variations in the generation technique. Next we can move onto complex datasets like CIFAR10(Krizhevsky et al. (????)) and if it performs well for these finally, Imagenet.

It is important to pick state of the art network architectures, because with a very strong classifier, it is safer to make comments about the success and pitfalls of the generation technique. We start with Lenet(Lecun et al. (1998)) architecture for MNIST and FashionMNIST since it is know to perform better than 90% accuracy after just a few iterations over training data. For CIFAR10, we need to use a deeper architecture like the Resnext (Xie et al. (2016)) architectures which show 70%+ accuracy.

For the above datasets and their architectures, the experiments need to test different hyperparameters like:

- **Learning Rate**
- **Batch Size**
- **Weight Decay**
• Different Optimizers

• Dropout

• Change in loss metrics over epochs

3.4.1 Visualizations

The included figures visualize the result of generation for MNIST and CIFAR10 datasets. They show the impact of different learning rates, batch sizes, optimizers and use of temperature softmax.

Figure 3.2: Generated Samples for MNIST Dataset Using Lenet Architecture. Hyperparameters Used: ‘sgd’ Optimizer, 1 Lr, 1000 Iterations
Figure 3.3: Generated Samples for CIFAR10 Dataset Using Resnext101 Architecture. Hyperparameters Used: 'adagrad' Optimizer, 0.1 Lr, 10 Iterations
Figure 3.4: The above figure shows images generated with temperature softmax. There is little change in the image generated after 1000 epochs in 3.2 and image generated after 10000 in (a). Using temperature helps generate more realistic images and significantly speeds up generation.
Figure 3.5: The above figure shows images generated with different batch sizes. With larger batches the images generation takes significantly longer and the quality of images created is significantly lower.
Figure 3.6: The above figure shows images generated with different optimization methods. Use of momentum significantly speeds up gradient descent, some experiments yielded better results after fewer iterations and lower learning rates.
3.4.2 Results

The categorical cross entropy loss for the input sample keeps on decreasing continuously throughout the generation process. If the sample is updated for long enough the loss descends to a negligible value (\(< 10^5\)). This generated sample classifies perfectly into its target class i.e. with confidence \(> 0.99\). Some samples descent faster than other, but this process worked for all the experiments conducted.

The samples generated for MNIST look very similar to the original images from the train dataset, and it is quite easy tell the different classes apart. Hence this sample generation process works extremely well for MNIST dataset. The samples generated for FashionMNIST are also visually similar to the original dataset, but they lack the fine details. Yet it is possible to tell the classes apart for a human being. Next I tried this process with CIFAR10 and SVHN. The generated samples classify perfectly, but they dont look anything like the original dataset. To a human viewer the samples look like images with random noise. Hence, the visual quality of generated samples drops significantly as we move to more complicated datsets.

When using SGD optimizer the generation process takes a significant number of epochs (1000 iterations) for the samples to get classified with high confidence. This is a high amount of compute and the underlying motivation is facilitate incremental learning I focused on improving the efficiency of the algorithm. Using momentum and more advanced optimizers makes a huge difference. Using regularization also helps speed up the process a bit, although the amount of decay needs to adjusted properly.

Hinton et al. (2014) introduced temperature softmax which was key to their knowledge transfer approach. I thought using soft labels might help the gradient descent approach here. Using temperature softmax improved the quality of the images generated as seen in the figures.
3.5 Conclusions and Future Scope

As shown in 3.6c the generation technique works exceptionally well for MNIST. With the right hyperparameter choices the quality of generated samples can be improved significantly. Images generated for MNIST dataset are quite realistic and it’s easy to make out the classes they belong to.

The generation method fails to create real looking images for CIFAR10, yet its important to note that they classify almost perfectly into their respective classes. This property of the generated samples might still make them viable for use in incremental learning and knowledge distillation. The next chapters will go into further depth on this topic.

I believe there is scope for further experiments with different types of architectures, and studying the impact of dropout. Dropout in deeper neural networks should yield a lot of variation in generated samples. Also, it may be possible to try incorporating a variational loss to ensure that the generated dataset belongs to the same data distribution as the original training data.
4.1 Deep Clustering and Hashing Networks

Another upcoming area of research in neural networks is Deep Clustering. One of the earliest methods proposed in machine learning was K-means Lloyd (1982), which was a unsupervised clustering method used to group similar data samples together. Recently researchers are trying to use deep learning to generate identical embeddings for similar samples. Also, there have been many publications related to Deep Hashing, which introduce quantization losses to ensure that generate embeddings are binary and can be used in hash-tables to make memory recall easier.

In this chapter, we look at incremental training with deep clustering and hashing networks. It looks at how clustering methods are different from general classifiers and how this is important from the point of view of incremental learning. I look at some implementations of deep hashing and propose a method of using such networks as classifiers. I also look at adding new classes to such networks without having to add any new parameters, unlike continual approaches with conventional classifier architectures.

4.2 Introduction

Xia et al. (2014) introduced a deep hashing technique where a similarity matrix $S$ of all samples is decomposed into a form $H^T H$, where $H$ is the hash for a specific sample. These hashes are now treated as label, and a deep neural network is trained to generate these hashes. Although this method ensures that similar samples get
identical hashes, yet it may be challenging to train the neural network to match these hashes. This was one of the first works in hashing using neural networks. Since, there has been research working completely with neural networks and experimenting with different losses. Most techniques make use of a \( \text{tanh} \) activation after the final layer to ensure each bit is normalized in the 1 to -1 range. To ensure that each bit as close to 1 or -1 as possible, a quantization loss is included usually a root mean square loss of the difference between bit magnitude and 1.

Zhu et al. (2016) introduced a pairwise crossentropy loss for similarity preserving learning. Hashnet (Cao et al. (2017)) builds on this and presents a weighted maximum likelihood estimation to optimize the hashcodes. Cao et al. (2018) presents another pairwise loss this time based on a cauchy distribution, which penalizes similar image pairs with higher hamming distance that the cluster radius specified.

Incremental Learning research has largely focused on conventional classifiers which use softmax activation and a crossentropy loss during training. Because of this setup, when such networks are tasked to solve incremental learning, they need to be modified to be able to be trained on any new classes. Deep hashing networks are different in the sense that their output layer can have a fixed shape based on the predetermined bit size and yet support varying number of clusters or groups of samples. One important point to be noted is that these methods were developed with the goal of image retrieval and hence the number of samples per group were considerably smaller than found in classification tasks. I attempt to turn these hashing networks into supervised clustering networks by building on the previous work done by Venkateswara et al. (2017) and Kang et al. (2019).
4.2.1 Training Process

Let’s say, given a dataset \( X = \{x_i\}_{i=1}^N \) and corresponding label \( Y = Y_{i=1}^N \). We create a pairwise similarity \( s_{ij} \) between \( x_i \) and \( x_j \). The similarity \( s_{ij} \) is equal to 1, if \( y_i \) is the same as \( y_j \) else \( s_{ij} \) is equal to 0. We also define the hash-codes \( h_i \) of length K-bits for each of these samples. Just like \( s_{ij} \) we also define \( q_{ij} \) which is the similarity between the hashes \( h_i \) and \( h_j \).

**Algorithm 2** Training of Supervised Deep Clustering Model using Similarity based Loss

**Input:** dataset \( D \), number of classes \( C \), bit size \( B \)

**Output:** Trained Clustering Model \( C \)

1: Initiate a neural network architecture with \( B \) neurons in the final layer.
2: Add a tanh activation after the final layer.
3: for \( i = 1 \) to \( e \) do
4: For current batch with \( b \) samples, generate similarity matrix \( S \) of size \( b \times b \). \( S_{ij} \) is set to 1 if \( C_i = C_j \), else it is 0.
5: Do a forward pass through the model \( C \) obtaining hashes \( H_i \) of size \( B \) each.
6: Calculate Similarity Based loss using \( H \) and \( S \)
7: Do gradient updates using this loss.
8: return \( C \)

4.2.1.1 Loss Functions

Most deep hashing research tries to improve the performance by experimenting with different loss functions. There are two aspects to the loss:

- Quantization Loss: This loss is included with the goal of ensuring that output
bits have a magnitude as close to 1 as possible.

\[ Q = \sum_{i=1}^{n} \| \text{sgn}(z_i) - z_i \|_2 \]  

(4.1)

- Similarity Based Loss: This loss focuses on separating different images and bring similar images together in the hash-space. Kang et al. (2019) formulates the problem to be:

\[ L = \sum_{s_{ij}} w_{ij} (s_{ij} \log \frac{1}{q_{ij}} + (1 - s_{ij}) \log \frac{1}{(1 - q_{ij})}) \]  

(4.2)

where, \( w_{ij} \) is used to appropriately weight similar pairs against dissimilar pairs, since the number of similar pairs will be significantly lesser (if all classes have equal samples.)

4.2.1.2 Output Layer

To be able to output hashcodes we need to be able to generate binary bits. For the sake of convenience we use \([-1, 1]\) instead. For this purpose a hashing neural network uses a tanh activation in the final layer.

4.2.1.3 Loss Based on Cluster Centres

One of the crucial challenges involved here is that each class will contain a large number of samples and since each training step will be constrained by the batch size, it may take a large number of epochs for all samples to be brought together or worse it may never happen. To tackle this, I calculate the locations of intermediate cluster centers after each epoch and add a loss based on distance of each sample from its own cluster center. Although the centres will keep changing after each mini batch, this should be help speed up clustering. Let \( C_k \) be the cluster center for class \( k \) with
The cluster center after \( e \) iterations would be,

\[
C_k^e = \frac{1}{n_c} \sum_{i=1}^{n_c} h_{c_i}^e
\]  

(4.3)

The centre based loss during the next iteration would look like,

\[
L_C = \sum_{i=1}^{n_c} \left\| C_k^e - h_{c_i}^{e+1} \right\|_2
\]  

(4.4)

**Algorithm 3** Training of Supervised Deep Clustering Model using Similarity based Loss and Centre Based Loss

**Input:** dataset \( D \), number of classes \( C \), bit size \( B \)

**Output:** Trained Clustering Model \( C \)

1: Initiate Clustering Model \( C \)

2: Using all samples from train set, do a forward pass through the network.

3: Calculate \( C \), where \( C_i \) is cluster centre for class \( i \).

4: Update model parameters using both Similarity Loss and Centre based loss calculated using \( H \) and \( C \).

5: After each iteration over the entire train dataset, update the centres.

6: return \( C \)

**4.2.1.4 Accuracy**

Since this is a clustering model, there is a need for a determined method of obtaining the label for a image. Most Deep Hashing research focusing on the MAP (Mean Average Precision) since the main goal has been using hashing for memory recall. Since our motivation is incremental learning of classifiers, here, for every input image \( x_i \) with a model output hash of \( h_i \), I pick the center \( C_k \) which is closed to the hash (using euclidean distance). The image is then predicted to be of class \( k \).
This metric can be modified to using a minimum hamming radius. During classification the samples should be within the hamming sphere to be classified as that class. But for initial experiments I use the closest center.

4.2.2 Experiment Setup

MNIST is a fairly uncomplicated dataset, most crossentropy based networks already have near perfect performance on it. For this experiment I chose the CIFAR datasets, I start with the CIFAR10 and resnext101 neural network which has been shown to have above 70% accuracy on it. I choose a comfortable bit size of 128, to make sure the network is not forced to condense too much information into a small number of bits. Next I will train it with CIFAR100 (resnext100 again) to study performance on a dataset with larger class size.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Train Acc.</th>
<th>Test Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR10</td>
<td>0.76</td>
<td>0.69</td>
</tr>
<tr>
<td>CIFAR100</td>
<td>0.98</td>
<td>0.45</td>
</tr>
</tbody>
</table>

4.2.3 Results

The supervised clustering model takes a long time to be trained for both the CIFAR10 and CIFAR100 datasets. During training of the CIFAR10 dataset the loss seems to settle after about 200 epochs, training beyond this point does not result in any significant drop in loss. The CIFAR100 model shows progress for longer and the change in loss trails off after 350 epochs.

Figure 4.2 and Figure 4.3 show the loss plotted against epochs for both the resnext networks. Table 4.1 shows the performance of the deep clustering network on CI-
FAR10 and CIFAR100 datasets.

When using the Centre Loss along with the Similarity Based Loss the training process is sped up by a considerable amount. But the end result remains the same. Since the loss does descent to a minimum value after training for a long time, this is an interesting result.

To investigate the cause of this I plotted the similarity and dissimilarity parts of the Similarity Based Loss. Based on the loss graphs, it appears that the model does a good job of bringing similar images together but it does a very poor job of separating dissimilar pairs of images.

Another way of verifying the clustering result is to pass the generated hashes to PCA or TSNE algorithm which can help us visualize the clusters in lower dimensions. I only do this for CIFAR10 dataset, since differentiating 100 classes with their own colors is very difficult.

Looking at the figures we can see that the clustering model does bring groups of similar samples together, but quite a lot of these groups are very close to other groups which are of completely different classes. This would not be a problem if the clusters were small enough in radius, but since the diameter of these clusters is comparable to the distance from nearby clusters there is a lot of misclassification. Also interestingly the model does not form one big cluster for all samples from one class, it forms several smaller groups for samples for the same classes, which is another interesting result.
Figure 4.1: Result of TSNE on generated hashes, similar samples are grouped together, but the blobs of dissimilar clusters are too close together and hence the classification accuracy is very low.
Figure 4.2: Similarity Loss Observed During Training of Resnext101 for CIFAR10 dataset

Figure 4.3: Similarity Loss Observed During Training of Resnext101 for CIFAR100 dataset
4.3 Conclusions and Summary

Based on the pattern of change in loss and accuracies shown by the supervised clustering networks, it appears as though the models are overfitting on the train dataset. Even then, the CIFAR10 trained resnext perform quite poorly on the dataset as compared to other state of the art methods. As shown in Chapter 3, trained neural networks that shows such high error rate do not generate realistic samples and it will be very difficult to train a student network or transfer knowledge. Despite this, if this challenge of training the teacher network well using a supervised clustering approach is overcome, it may be possible to explore the possibility of generating a variety of samples within a class’ cluster sphere, which can then be used to train student networks.
Knowledge distillation is activity of transferring information from a teacher network to a student network. The teacher network is already trained on a task and the student is to be trained on it or on another task related to it. Section 2 studies different research papers in this area. One of most groundbreaking contributions was by Hinton et al. (2014) and Yosinski et al. (2014) who showed how using temperature softmax can be used to generate a softer label distribution for the transfer dataset.

A sample generation method was introduced in Chapter (3). In this chapter, with the use of this method we propose a new method of knowledge distillation.

5.1 Proposed Method

The samples generated from the method proposed in Chapter 2 are a good representation of what a trained neural network expects to see with respect to a specific class. These samples could be used to train a new classifier. Another interesting aspect of this is that the samples generated by the student neural network can be used to query the teacher network to confirm if the student is learning correctly. A cyclic process in which the student learns the correct labels for samples created by the teacher and samples created by itself should result in good test data performance by the student network. The next section outlines the training process and each step involved.
5.2 Process

One of the earliest methods in incremental learning were the use of pseudo-recursal to prevent catastrophic forgetting. This process builds on this idea by using generative techniques while also using a querying loss to correct the student network if it is learning incorrectly.

Algorithm 2 lists the steps during cyclic training. The subsections below go into further detail about the generation process and how the student network is trained.

**Algorithm 4** Cyclic Training of Student Network

**Input:** Teacher $T$, number of classes $C$, image size $s$

**Output:** Trained Student $S$.

1. Initiate a student network $S$ which should classify into $C$ classes.
2. Generate $D_T$ samples from Teacher network, their teacher labels being $L_T$
3. Train Student network using $D_T$ and $L_T$
4. Generate $D_S$ samples from Student network.
5. Query Teacher Network for $D_S$ and get labels $L_{TS}$
6. Train Student network using $D_S$ and $L_{TS}$
7. Repeat Step 2 through 6 till student network meets target test accuracy.

### 5.2.1 Sample Generation from Teacher

The student needs to be initialized to be able to query the teacher properly. To do so, I generate an initial dataset from the teacher network using the method described in Algorithm 4. It is important to generate a healthy number of samples and make sure the student does not overfit on these. Since there can be many permutations of parameters can results in similar predictions, the state of the student network after this initial training may not be optimal state for classification on the test dataset.
The student model needs to be corrected and retrained multiple to ensure that the knowledge from the teacher has been properly transferred to the student.

### 5.2.2 Sample Generation from Student

The generation process in 3 proposes a method to generate samples which is what a network expects to see for a certain target class. The previous step used samples generated by the teacher to train the student, yet it is quite possible that the resultant student parameters are not optimal for the purpose of the goal task. To ensure this we can again generate more samples, this time from the student network itself. These samples can be verified by passing these samples through the teacher network and comparing the teacher labels with the student labels.

### 5.2.3 Teacher Querying and Student Update

In case the teacher labels are not similar to the student labels, it is imperative to correct these mistakes and update the student network with the student generated samples and their teacher verified labels. While the previous step help learn goal task information, this step should prevent overfitting and chances of misclassification. These steps may have to be continued in a for loop till a consensus is reached between the student network and the teacher network.

### 5.2.4 Co-generation by Teacher and Student Networks

Another way to better transfer information from teacher to the student network might be to find samples that the student classifies poorly on. This is can be done by making a small modification to the generation process. Both the teacher and the student network can make updated to the input samples together, the teacher attempts to reduce the classification loss, while the student is trying to increase the
classification loss for the same target label. This should result in generating samples which are correctly classified by the teacher network but are wrongly classified by the student network. The student can now be updated using these generated samples along with the correct teacher labels.

5.3 Experimental Analysis

Building upon the results of the generation method described in Chapter 3, the generated images can now be used to train a student network. For this purpose the number of samples generated should be significantly more. Once the samples are generated there are three important experiments which need to be conducted:
• Training on Teacher Generated Samples as described in 5.2.1

• Cyclic Training with independently generated samples from teacher and student networks as described in 5.2.3

• Cyclic Training with samples generated by teacher and student networks together as described in 5.2.4

5.3.1 Results

5.3.1.1 Teacher Generated Samples

We conduct this experiment on the MNIST dataset using the lenet network architecture. The experiment generated 1000 samples for each class of the initial data distribution, and then attempts to train on this data. The loss used for training is categorical crossentropy. As expected the newly trained student network fails to achieve the performance of the teacher network which was about 97% and achieves around 80% classification accuracy. Although this confirms that the generated samples are capable of transferring information there is definitely room for improvement.

The same experiment is conducted with CIFAR10 dataset and resnext101 architecture, with significantly worse results. The cifar10 student network achieves a maximum test accuracy of 12% as compared to the 69% accuracy of the teacher network. This accuracy does not increase with increase in number of samples generated or longer training of the network (higher number of iterations). This means that the samples being generated for CIFAR10 are of inferior quality and transfer very little information.
5.3.1.2 Cyclic Training on independent samples

As seen in the previous experiment, training solely on the teacher generated samples does not result in a well trained student network. Some kind of supplementary training is required to increase its accuracy. When this experiment was conducted on MNIST, using the cyclic training on both teacher generated and student generated samples, over each cycle it gradually improves the performance of the network. The performance increases quite haphazardly and quite often drops off significantly after training on the student generated samples. Eventually the student network achieves 96% accuracy on MNIST test data, compared to the teacher’s 97% this is an excellent result.

When this cyclic training method is tried with CIFAR10 data and resnext101
Figure 5.3: Test Accuracies noted during all cycles of Student Lenet Network trained on MNIST

architecture, the student net

5.3.1.3 Cyclic Training on co-generated samples

In the previous experiment the teacher and student generated samples independently. In this experiment they work together to create samples which are classified poorly by the student network. When run on MNIST dataset the results are immediately different, the accuracy increase quite regularly in this run although it is still a slower ascend. The generation loss does have a risk of exploding due to the reverse gradients from the student network, but well adjusted hyperparameters avoid this and results in a 96% accuracy on the MNIST test dataset.
Table 5.1: Accuracies During Cyclic Training

<table>
<thead>
<tr>
<th>Cycles on MNIST</th>
<th>Acc. after $T_s$ train</th>
<th>Acc. after $S_s$ train</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8286</td>
<td>0.6292</td>
</tr>
<tr>
<td>2</td>
<td>0.8958</td>
<td>0.7247</td>
</tr>
<tr>
<td>3</td>
<td>0.9167</td>
<td>0.7703</td>
</tr>
<tr>
<td>4</td>
<td>0.9327</td>
<td>0.8032</td>
</tr>
<tr>
<td>5</td>
<td>0.9339</td>
<td>0.8426</td>
</tr>
<tr>
<td>6</td>
<td>0.9436</td>
<td>0.7658</td>
</tr>
<tr>
<td>7</td>
<td>0.9457</td>
<td>0.8441</td>
</tr>
<tr>
<td>8</td>
<td>0.9493</td>
<td>0.8001</td>
</tr>
<tr>
<td>9</td>
<td>0.9506</td>
<td>0.8037</td>
</tr>
<tr>
<td>10</td>
<td>0.9548</td>
<td>0.7545</td>
</tr>
<tr>
<td>11</td>
<td>0.9551</td>
<td>0.7556</td>
</tr>
<tr>
<td>12</td>
<td>0.961</td>
<td>0.7465</td>
</tr>
<tr>
<td>13</td>
<td>0.9633</td>
<td>0.7964</td>
</tr>
<tr>
<td>14</td>
<td>0.9595</td>
<td>0.7968</td>
</tr>
<tr>
<td>15</td>
<td>0.9575</td>
<td>0.7749</td>
</tr>
<tr>
<td>16</td>
<td>0.9616</td>
<td>0.8307</td>
</tr>
<tr>
<td>17</td>
<td>0.963</td>
<td>0.849</td>
</tr>
<tr>
<td>18</td>
<td>0.9577</td>
<td>0.8144</td>
</tr>
<tr>
<td>19</td>
<td>0.9648</td>
<td>0.8107</td>
</tr>
<tr>
<td>20</td>
<td>0.9631</td>
<td>0.8079</td>
</tr>
</tbody>
</table>

5.4 Conclusions and Summary

The above results confirm that the generation method can be used for knowledge transfer between two networks. It is important to note that samples generated from the teacher network alone are not sufficient to train the student network. There
is a need to correct the student network in some manner to ensure best possible performance. There might be other ways of doing this apart from the ones explored in this chapter like, introducing a activation similarity loss between the teacher network activations and student network activations. This method seems quite promising and there is future scope in implementing it for different tasks other than classification and also with different datasets and network architectures.
In this thesis I have put forward new approaches to solve the problem of Incremental Learning. My proposals focus on resolving catastrophic forgetting which has been a long standing challenge in not just continual learning but deep learning research in general. Well trained classifiers compress the information learned from the train dataset. Hence, the catastrophic forgetting problem can be reshaped to be a challenge of decoding and maintain this information in a trained network. My proposal focuses on regenerating samples using backpropagation, very similar to conventional training methods. During incremental training process the model can rehearse on these generated samples to ensure retention of information for old task data.

Chapter 3 proposed a sample generation technique, which performs exceptionally well with simpler datasets like MNIST, FashionMNIST and generates samples which can be easily discerned by the human eye. This method does not do as well on more sophisticated datasets like CIFAR10, SVHN, the generated samples are not visually similar to the images from the original train data. But since all the generated samples classify perfectly into their target classes (the ones they were generated for), it is important to verify their quality. To test whether these samples well represent the original task data distribution can be verified by training a new neural network using this data.

Chapter 4 covers a different area of deep learning. Motivated from Deep Hashing Networks, I intended to make the most of the properties of a clustering problem. As a result of clustering we are left with groups of samples very close together in an embedding space. Since generation from classification networks lacks in producing
variations in the samples being created, the similarity between all points within a
cluster sphere could be used to overcome this challenge. It should be possible to pick
any point within this cluster sphere and generate a sample for that specific embedding.
This will allow a large variety of samples to be generated for the same class. Since
very little work has been done to build deep clustering networks for classification I
proposed loss functions to train such a neural network. The similarity based loss
attempts to bring identical samples together in the embedding space and separate
dissimilar samples. I also propose a Center Based Loss to speed up the process, here
I compute the cluster means before each iteration and calculate the distance of each
sample from its cluster centre and attempt to reduce this. Unfortunately I faced
several challenges during this experiment. The trained model is able to learn from
similarity in between samples and bring them together in embedding space, but fails
to separate samples from different classes. Perhaps as a result of this, or due to other
failures of this technique the end trained model has a very low classification accuracy.

Chapter 5 builds on this generation method and tests the completeness of the
generated samples. It also puts forth two novel cyclic learning techniques which utilize
samples generated from both the teacher and the student network. One method uses
samples generated by teacher and student network separately, queries the teacher
network for these samples and then trains the student network on these samples
and labels. The other method proposes a joint update method where the teacher
and the student network together generate a sample that is classified correctly by
the teacher network but incorrectly by the student network. These methods both
achieves near perfect transfer learning performance on MNIST dataset, the student
network ending up being within \( \pm 1\% \) of the teacher network’s train accuracy. The
cyclic training algorithm is fairly simple and can incorporate different approaches
which correct sample labels with the help of the teacher network.
Both the sample generation method and the cyclic learning approach show promising results on at least one dataset. These methods are based on existing tried and tested techniques and form a strong base for future research. The sample generation technique makes a meaningful contribution towards neural network output explanation and has potential to improved and used with more sophisticated datasets. The cyclic learning algorithm confirms that the samples contain meaningful information and this algorithm can be further modified to be used with different types of neural networks. My experiments shed a new light on catastrophic forgetting and how it can be overcome using rehearsal based methods, while depending on nothing except for a trained network.


