Automatic Curriculum Learning for Deep Models Using Active Learning

Ian McWilliam

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2018

Abstract

Acknowledgements

Many thanks

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ian McWilliam*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

1.1.1 Supverised Learning

1.1.2 Curriculum Learning

1.1.3 Active Learning

1.2 Contribution

1.3 Document Structure

The subsequent chapters of this thesis will be organised as follows:

- Background - Here we will introduce in more detail the topics introduced above, giving the reader the background required to understand and appreciate the rest of work. We will also develop some of the nomenclature that will be used in various other sections.

- Related Work - In this chaper we set our contributions in the context of related studies, by dicsussing various other works which have tested approaches for automating curriculum discovery and improving learning performance with curriculum methods.

- Bootstrapped Active Curricula - In this chapter we introduce the bootstrapped active curriculum approach, explaining the curriculum construction, the methods used to test it, as well as the results of the experiments and subsequent discussion.

- Dynamic Active Curricula - Motivated by the previous chapter, we test methods for dynamically constructing and implementing learning curricula throughout training, again laying out the curriculum construction approach, the methods used to test its efficacy and a discssion of the results of the experiments.
- Conclusion and Further Work - Finally, we summarise the findings of the thesis and suggest directions in which future work can build on the experiments shown in this paper.

In the next section we will introduce in more detail the topics introduced above, giving the reader the background required to understand and appreciate the rest of work. We will then set our contribution in the context of related studies that have also tested methods for automating curriculum discovery and improving learning performance with curriculum methods.The

Supervised learning is the area of machine learning in which algorithms learns the relationship between a set of input features and corresponding 'ground truth' labels, the ultimate goal being to construct a predictive model of the relationships between the inputs and the labels in order to predict the labels of future, unseen input samples. Deep learning models perform this task by building hierarchical representations of the input features throughout a multitude of layers, often using feature maps such as convolutions or recurrent layers to construct complex representations of the inputs. When training a deep model a standard methodology is *gradient descent*, which calculates the gradient of a chosen error function with respect to the free parameters of the model so as to tune the parameters in a way that will minise this error function on the training set of input-label pairs. A popular variant of the gradient descent algorithm is *mini-batch stochastic gradient descent*, which uniformly samples mini-batches of a preset size from the available training data, performing a gradient descent update on each batch until the all training samples have been selected, then repeating until the network converges to a solution. Sampling uniformly from the training data ensures that the mini-batch gradient is an unbiased estimation of the gradient over the whole training set, however the estimation can exhibit high variance. In this paper we analyse approaches for augmenting mini-batch stochastic gradient descent (SGD), using methods inspired by two areas of study; *active learning* and *curriculum learning*.

Active learning is generally used when there is a prohibitive cost to obtaining labels for supervised learning; in such cases it is desirable to know which samples will lead to the greatest best improvement in algorithm performance, selected from a set of unlabeled candidate samples. As such, there is a rich literature in active learning detailing how to choose the most informative samples, in particular using *acquisition functions* to select which sample(s) to label and use for training. While active learning is usually employed to reduce labeling costs and speed up learning, curriculum learning explores the hypothesis that the overall accuracy of the network can be improved by presenting the training data to the algorithm in a meaningful order. Inspired by the way in which humans and animals learn, REF BENGIO suggest learning can be improved by emphasising easier concepts earlier on in training before introducing difficult samples, or by emphasising more difficult training samples later in training. In their paper REF BENGIO for example, the authors use a the 'Geometric Shapes' dataset, consisting of images of geometric shapes of different complexities, to show that by initially training on 'easy', regular shapes, test classification accuracy is improved.

The substantial challenge with curriculum learning however is that in many domains however it is challenging to identify a clear delineation between 'easy' and 'hard' samples through which to implement curriculum learning; in this paper we propose that the methodologies developed for the active learning approach are well suited to estimating the difficulty of training samples, allowing the automatic construction of learning curricula that will improve the training of deep networks on a wide range of tasks. Specifically, the approach set out in this paper modifies the SGD algorithm by, instead of sampling with uniform probability, sampling training examples proportionally to some measure of 'difficulty', as derived from an active learning style acquisition function metric. We test our methods on three image classification datasets; MNIST, CIFAR 10 and the GeoShapes dataset (a geometric shapes classification dataset with an established curriculum baseline), exploring a range of active learning metrics as well as several curriculum construction methods. Our results show consistent performance against a uniform sampling baseline, with significant reductions in test set error, robust to different network architectures, datasets and curriculum methodologies. The output of this work is a set of flexible methods for improving deep models in a wide range of tasks, as well as an investigation of how using the difficulty and uncertainty of training samples affect learning performance.

In the next section we will introduce in more detail active and curriculum learning, exploring the link between the two approaches and the sometimes contradictory hypotheses they pose. We will then discuss related work where the authors implement similar methods for improving algorithm performance through biasing learning towards certain training samples throughout training. We will then lay out the experimental methodologies and datasets used in the paper before presenting and analysing the results of the tests and concluding with a discussion and suggestions for further work.

Chapter 2

Background

2.1 Supervised Learning

Machine learning is the field of study concerned with building algorithmic systems that can automatically infer patterns and relationships in data. While machine learning has several main subfields, for example reinforcement [38] and unsupervised learning [17], the most commonly studied area of the field is arguably *supervised* learning. Supervised learning is characterised by learning a function that maps inputs to outputs (or 'labels'), using a *training set* of example input-output pairs, (supervised learning has previously been referred to as "learning from examples" [6]). The training set, which we will denote by $\mathcal{T}$ in this report, consists of a number of inputs-output pairs: $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), ...(x_N, y_N)\}$, where N is the number of examples in the training set. We will refer to an input-output pair from the training set as a 'sample' or an 'example' interchangeably. Concretely, the goal of supervised learning is to find a function $f : X \to Y$, where X denotes the input space and Y the output space, so as to minimize the *generalization error* of the function. The generalization error is defined as the expected error of the function averaged over future input-output samples REF MURPHY, where the error is defined using a loss function $L : Y \times Y \to \mathbb{R}$, such that $L(f(x_i), y_i)$ gives the error of the function on the input-output pair $\{x_i, y_y\}$. Generalization error can therefore be defined as $\mathbb{E}_{X \times Y}(L(f(x), y)$, however, as finding a closed form soltuion for the generalization error is generally intractable, it is usually approximated using the empricial error on a held out *test set* of samples that were not used when fitting the function. Using a test set of M input-output examples, we therefore approximate the generalization error as $\frac{1}{M}\sum_{i=1}^{M} L(f(x_i), y(i))$. A supervised learning algorithm is a method for fitting the function f using the training set $\mathcal{T}$, usually by minimizing the loss of the function over the input-output samples in the training set, the process of fitting the function to the training data is referred to as *training*. In order to avoid *overfitting*, where the function has a low error on the training set but a high generalization error[1], the loss function often includes regularization terms that penalise function complexity, or the range of functions that can be fitted by the algorithm is constrained. Furthermore, in addition to a training set and test set, a *validation set* of samples is often used to monitor the function error on samples outwith the training set during training, in order to measure whether or not the function is overfitting.

2.2 Deep Learning

Deep learning, as applied in the supervised learning setting, refers to algorithms which model the relationships in the training set using complex, hierarchical representations of the data, usually doing so using multiple 'deep' layers, as well as feature maps such as convolutions or recurrent layers REFS. Deep learning models fit functions by passing the input signals through several layers of transformations, multiplying the signal by parameterised weights and then passing the weighted signal through non-linear *activation functions*. The combined model of weights and transformations is referred to as a 'deep network', or 'deep neural network', as a result of similarities to the functionalities of neurons in brain. This approach allows the algorithm to transform the input space in order to make the closely match the output, with the weight parameters being tuned throughout training in order minimise the network loss on the training set. Deep learning has been the subject of much research in recent years, showing state of the art performance in a range of domains including image recognition and natural language processing REFREFREF.

2.2.1 Feedforward Networks

Arguably the simplest example of deep network is a *feedforward network*, usually consisting of an input layer, an output layer and several intermediate 'hidden' layers REF, as illustrated in Figure REF. There are a variety of 'hyper-parameters' that must be set when implementing a feedforward network, or any deep model, for example the architecture (i.e. number of hidden layers and the number of nodes each layer contains), learning rate and chosen activation functions.

[1]More specifically, overfitting refers to the situation where the function has a low error on the training set and a higher generalization error than a similar function with a higher error on the training set.

In classification problems, where the aim is to classify an input x into one of several possible classes, the output activation function used is the softmax function REF, which maps the output vector to a vector of probabilties, summing to one. Denoting by **h** the signal forward propogated through the network to the output layer, the final output vector of the network is then given by $softmax(\mathbf{h})$, where:

$$softmax(\mathbf{h})_i = \frac{e^{h_i}}{\sum_{c=1}^{C} e^{h_j}}, \tag{2.1}$$

where C is the number of possible class labels. The loss function typically used to train such networks is the *cross-entropy* loss function, defining the output vector of the network for input sample x by $f(x) = \tilde{\mathbf{y}}$, cross-entropy error is defined as follows:

$$crossentropy(\tilde{\mathbf{y}}, \mathbf{y}) = -\sum_{c=1}^{C} y_c log(\tilde{y}_c). \tag{2.2}$$

In several of the experiments in this report we will use feedforward networks for image classification, using a softmax output layer and cross-entropy error as the chosen loss function.

2.2.2 Convolutional Networks

The state of the art deep models for image recognition are *convolutional networks* REF REF. Convolutional layers pass filters of a predetermined size over an image, allowing for weight sharing between nearby pixels and the subsequent extraction of visual features, as illustrated in Figure REF. Usually, multiple convolutional layers are used in order to extract a hierarchical representation of different features, before the signal is flattened to a vector output and passed through the softmax function for classification. Convolutional layers are often followed by *max pooling* layers [28**?**], which downsample the the signal into a lower dimension, and also *batch normalisation* layers [20], which normalises the signal as it propogates through the network.

2.3 Stochastic Gradient Descent

The standard method for training deep models is *gradient descent (GD)* [19] [36] REF, an optimization algorithm which varies the parameters in the model depending on the gradient of a chosen error function. Defining the current weight parameters of the network by θ_{old}, the updated weight after gradient descent as θ_{new} and the loss function of the model on the training set $\mathcal{T}$ by $L(\theta)$, we update the weight parameters by performing a gradient descent update as follows:

$$\theta_{new} = \theta_{old} + \Delta\theta, \tag{2.3}$$

where

$$\Delta\theta = -\alpha \frac{\partial L(\theta)}{\partial \theta}|_{\theta_{old}}. \tag{2.4}$$

α in this case is the pre-set *learning rate* [19] REF, controlling the step-size of the gradient descent update. To implement GD it is necessary to calculate the gradient of the error function with respect to the parameters of the model, usually this is done layer by layer, starting with the output layer, in a method referred to as *backpropogation* REF REF.

One of the drawbacks of gradient descent is that calculating this gradient over the entire training set can be very computationally expensive, particularly when dealing with large training sets; to address this issue a variant of GD, *stochastic gradient descent (SGD)*, is often used [34]. With SGD, instead of calculating the error gradient over the entire training set, only one sample is used to calculate the gradient and update the model parameters. Alternatively a selection or 'mini-batch' of training samples may be used to calculate the gradient, in which case the optimization algorithm is referred to as *mini-batch stochastic gradient descent*. To implement mini-batch SGD, batches of samples are selected uniformly from the training data, usually withou replacement, and gradient descent updates are performed using the sample in the batch. This is then repeated until all of the trianing samples have been used in a batch; a complete pass through the training data is referred to as an *epoch*, and the number of training epochs is set as a parameter of the experiment.

It can be shown that that SGD and mini-batch SGD produce an unbiased estimate of the error gradient [34], with various convergence proofs showing that SGD will eventually converge to an optimal solution under certain conditions [34]. There are many adaptations to 'vanilla' SGD, for example momentum based methods [37] and other more sophisticated optimization algortihms which build on SGD to result in better and quicker fitting of the model parameters. In this paper we will employ the ADAM optimiser [22], the details of which are laid out in Algorithm 1 of [22].

2.4 Curriculum Learning

When training a deep model, or indeed any supervised learning algorithm, one usually trains the model over the entire training set throughout the entirety of training. When using stochastic gradient descent for example, all epochs would typically consist of a full pass through all available training samples. *Curriculum learning*, as introduced in [43], hypothesises that model performance can be improved by instead training on samples in a meaningful order, with the order defining a 'curriculum'. The motivation stems from the way in which humans and other animals learn, usually beginning with easy concepts before moving onto more complex facets of the area of study. The same principle can be applied to training deep models, and the authors of [43] suggest that, by initially training only on 'easy' samples, one can reduce overall generalization error.

While the term curriculum learning in the machine leanring setting may be a relatively new one, the concept was arguably introduced with Elman's 1993 paper "Learning and development in neural networks: the importance of starting small" [9]. In this paper the author demonstrates in a language modelling task that inhibiting the memory of the network in the early stages of learning, so that it can only analyse a small subset of the training set, ultimately improves performance. The author motivates the method by analysing the learning dynamics learning of connectionist networks, specifically the sensitivity of the overall model to the early stages of training and the implications therein for what data should be used in these early epochs. Interestingly, the author suggests motivations for beginning training with either 'easy' or 'hard'/'noisy' samples. In the case of using easy samples, he argues that this will prevent the network from falling into "early commitment to erroneous hypotheses", instead enabling it to learn broader concepts at the key early stage of training that will act as "scaffolding" for more complex concepts. Conversely, he also suggests that using harder/noisier samples may also prevent the network from reaching such erroneneous hypotheses, as the high variance in noisier samples will lead to less consistent gradient descent updates and prevent the networks weights from converging to an area of parameter space from which it will be difficult to exit in the later stages of training. This dynamic, wherein initialising learning with eiher easy and hard samples somewhat paradoxically seems to improve learning in both cases, is something which will be revisited throughout this report.

The authors of [43] also offer several theoretical justifications for curriculum learning, for example comparing the technique to *continuation methods* [8]. It is proposed that the easier samples represent a smoother, more convex version of the error space of the overall problem, and that, by training on easier samples, the parameters of the model are effectively initialized into an area of parameter space closer to the global optimum. This argument is similar to that of unsupervised pre-training [10] which again has been shown to lead to better generalized models by initializing the parameters into parts of the error space closer to the global optimum [4]. Comparisons have also been drawn between curriculum learning and *transfer learning* [31], with the easier samples being seen as a separate task that the model is trained on, before using the weights for a different task (i.e. the harder samples) as in transfer learning [41].

The example given in [43] for curriculum learning is the ‘Geometric Shapes’ dataset [30], an image classification task where a network attempts to classify whether or a not an image shows a rectangle, ellipse or triangle. In this case there is a natural subset of ‘easy’ samples; specifically squares (i.e. regular rectangle), circles (regular ellipses) and equilateral triangles. The authors show that, by training initially on only the regular shapes, then transitioning to training on harder shapes, the test set performance is significantly improved compared to training simply on the harder shapes for the entirety of training. One issue with this study is that it can be argued that the curriculum trained model has seen more samples overall than the baseline, as the curriculum model is trained on both an ‘easy’ training set and a ‘hard’ training set, whereas the baseline is trained only on the hard training set. A better baseline therefore is a model trained uniformly on the union of the easy and hard training sets. While the authors do comment on this issue, and claim that the curriculum method still outperforms uniform sampling from the combined training set, the results we will set out in this paper did not reach the same conclusions.

Curriculum learning is similar in concept to *self-paced learning* [24]. In this paper, the authors train a latent SSVM model [11] by adding a regularization term to the objective function that indicates whether or not a sample is ‘easy’ or note, depending on how well the current model parameters correctly classify the sample. Using this approach they limit the number of training samples used for parameter updates, a governed by a weight parameter that is annealed throughout training until all training samples are considered. They test their method on a variety of tasks including natural language processing and image classification. While the self-paced learning (SPL) method is tested for the SSVM model in [24], an exploration of SPL as applied to convolution networks is laid in [1]. The author applies SPL and a variant defined as ‘Self-Paced Learning with Diversity (SPLD)’ [21] which expands the SPL method to emphasise a diverse representation of samples. The author of [1] tests various curriculum methods on the CIFAR 10 image classification dataset, interestingly, similar to the remarks mentioned above in [9], the author found that training on samples with *decreasing* difficulty outperformed the a standard curriculum of training on samples with increasing difficulty. However, the author reports that little overall improvements were seen in any of the curriculum methods compared to standard training procedure using the entire training set.

There are various examples of curriculum learning being applied in areas where the learning problem can be split into discrete ‘tasks’. For example in [32], the authors use an animal image classification dataset that has been manually annotated with a perceived ‘easiness’ score. The authors divide the dataset into five equal tasks depending on difficulty score, and demonstrate that their curriculum method outperforms a standard multi-task learning algorithm. The authors of [26] also find a natural curriculum in a handwritten text line recognition task, finding that by first training on short sequences of text, before progressing to to the longer sequences significantly accelerates the training time of of a recurrent neural network [27] model for text recognition. More recent applications of curriculum learning include [41], where the authors provide a theoretical study of how curriculum learning affects the convergence of stochastic gradient descent, as well as implementing a curriculum using transfer learning by using a model trained on a separate task construct a curriculum for an image classification task, finding that the curriculum improves performance. The authors also test the effect of using an “anti-curriculum”, where training progress from difficult to easy samples, as in [1], however they find this significantly underperformance the baseline, as illustrated in Figure 2.1, taken from Figure 3b of [41].

While the number of studies implementing curriculum learning methods is increasing, a key difficulty in constructing a curriculum is that it is often very difficult to delineate between ‘easy’ and ‘difficult’ samples REF, particularly for large datasets where manually annotating the difficulty of the each sample is infeasible. It is also an open question as to how best to construct a learning curriculum, given a measure of sample difficulty, in particular given the aforementioned questions around whether or not training should progress through increasingly difficult or easy samples. A key issue therefore is that of exploring methods for automating the construction of learning curricula, and it is towards this goal that this paper contributes; specifically investigating how active learning methods (introduced in the next section) can aid such curriculum construction. In Chapter REF, we will discuss a variety of other studies that have implemented methods for automated curriculum construction. We first however introduce *active learning*, including some of the methods we will use to estimate sample difficulty and construct learning curricula in Chapters 4 and 5.

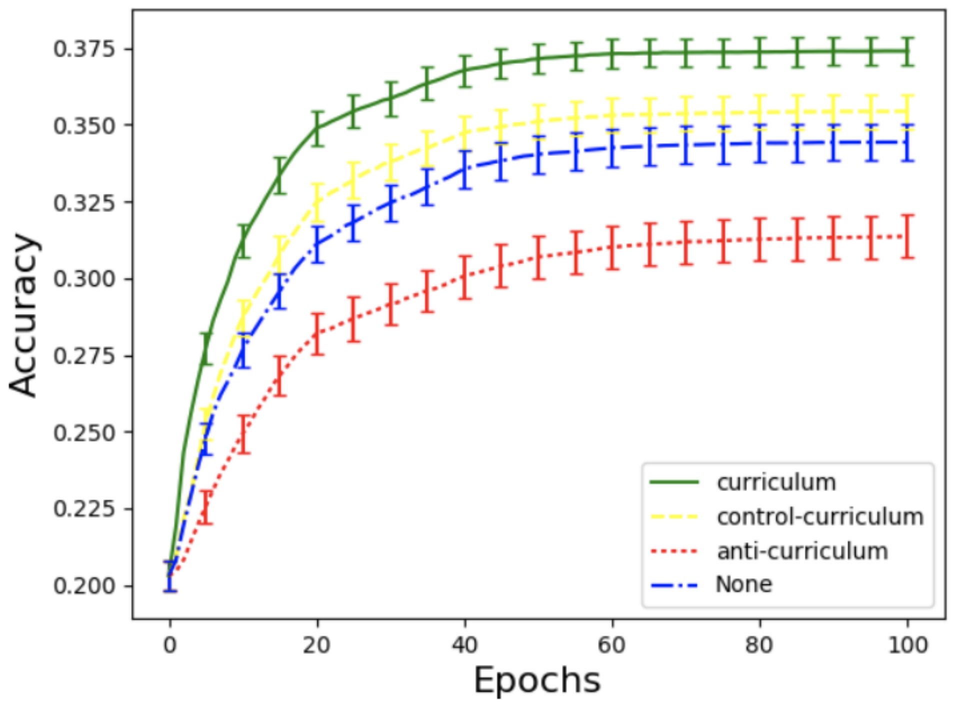


Figure 2.1: Figure 3b from [41], test classification accuracy of a convolutional network trained using various curriculum methods on the CIFAR 100 dataset [23]. The curriculum method trains on progressively more difficult samples, while the anti-curriculum method trains on progressively easier samples. The control-curriculum is trained using a random curriculum to control for differences in training between the curriculum methods and the baseline.

2.5 Active Learning

A key component in any supervised learning effort is labeled data; in many domains it is relatively easy and cheap to obtain a large number of training samples, however in others it can be far more costly, particularly acquiring accurate labels. In medical image analysis for example one may require a domain expert to spend significant time analysing each image before assigning label, or in document tagging it can take time to read a document and assign a topic label. It can therefore be very useful for a designer to understand which samples they should go to the effort of acquiring, labeling and feeding into their chosen learning algorithm, generally measured by how much the chosen samples improve the network performance, compared to if samples were instead selected randomly. A field of study that attempts ot address this is *active learn-*

ing [33] [6] [7], which studies methods for choosing, often from a pool of unlabeled candidate samples, which sample should be labeled and used to train the model.

Figure 2.2, taken from Figure 1 of [33] shows a typical active learning cycle; a machine learning model is trained on the current set of labeled training data, $\mathcal{L}$, the model then "queries" an "oracle", in this case a human anotator, for the labels of one or more samples from a pool of unlabeled samples, denoted $\mathcal{U}$. The queried samples are then labeled and added to $\mathcal{L}$, at which point the model is retrained and the expanded training set and the cycle reiterates until training is stopped. Note that, as well as pool based query strategies, some active learning methods may enable the model to query samples from the entire input space, as opposed to being limited to a pool of unlabeled samples [33].

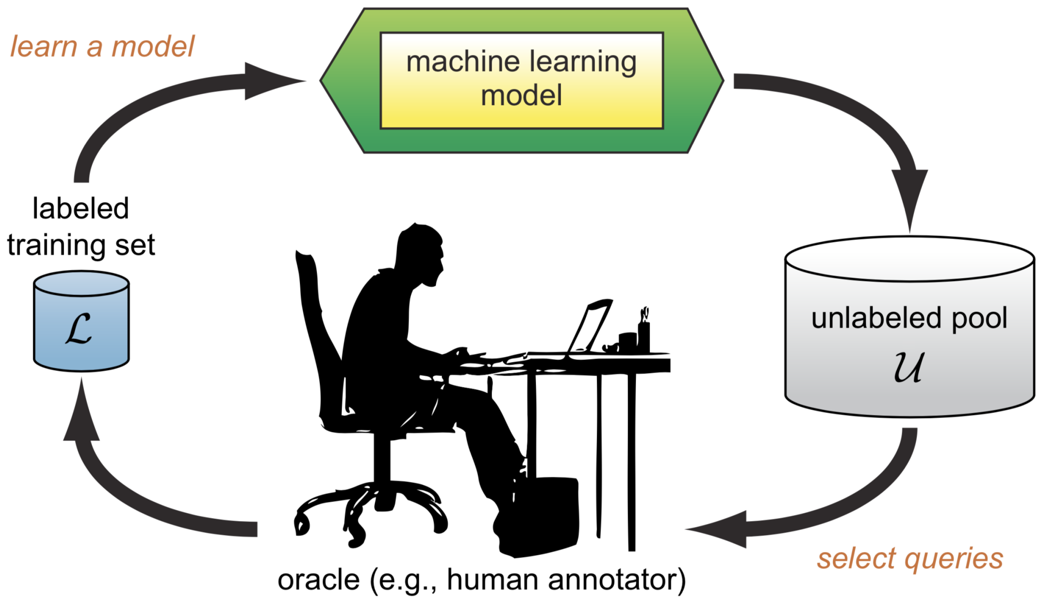


Figure 2.2: Figure 1 from [33], showing a typical "pool-based active learning cycle".

There are a variety of approaches to the active learning problem, however most involve the use of an *acquisition function*, which selects which sample from $\mathcal{U}$ (the set of candidate unlabeled examples) should be selected for labeling and training [33]. As the most appropriate training examples varies depending on the model itself, the chosen sample is said to be 'queried' by the algorithm. The motivation behind different active learning approaches vary; one of the most common approaches is that of *uncertainty sampling* [33], wherein the samples that the learning algorithm is most uncertain about labeling are queried. There are a variety of ways to measure algorithm uncertainty, and we will suggest in this paper that algorithm uncertainty can provide an effective measure of sample difficulty which can be used to automate the construction of learning curricula. It is interesting to note the somewhat opposing assumptions underlying active and curriculum learning; in the active learning approach the *most* uncertain samples, which could arguably be seen as ‘hard’ samples from a curriculum point of view, are selected for training. In curriculum learning however, the focus is usually on selecting easy samples, particularly early on in training, to improve model performance. Again, this speaks to the dynamic discussed in [9], where the author lays out potential justifications for the learning benefits of both training on both easy or hard/noisy training samples. As we will see in the results sections of Chapter 5, our experiments support this idea, with both ‘easy to hard’ and ‘hard to easy’ curricula improving model performance in some cases.

A common way of estimating model uncertainty is by analysing the distance to classification threshold of the model outputs; for example one method is to select the sample about which the model is least confident in predicting (taken from [33]):

$$x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x), \tag{2.5}$$

where

$$\hat{y} = \arg\max_y P_\theta(y|x). \tag{2.6}$$

Where x^*_{LC} is the queried training sample and $P_\theta(y_i|x)$ is the model’s predicted probability that sample x is of class y_i, given model parameters θ. Similarly, samples can be queried by their average distance to classification threshold or, similarly, the entropy of the algorithm prediction, again taken from [33]:

$$x^*_H = \arg\max_x - \sum_i P_\theta(y_i|x) \log P_\theta(y_i|x), \tag{2.7}$$

where the sum runs over the possible classes y_i. More sophisticated methods for estimating prediction uncertainty include the “Bayesian Active Learning by Disagreement (BALD)” [18] acquisition function, which is used for deep models by the authors of [14] and is introduced and used in Chapter 5 of this paper.

There are many examples of authors implementing active learning approaches for a variety of algorithms; [39] develops various methods to query unlabeled samples using suppoer vector machines to categorize news stories, [42] uses active learning to reduce the burden of human effort in labeling video data. In a more recent study, the authors of [40] employ active learning with deep network for image classification, succesfully reducing the number of training samples required to achieve promising results on challenging image classification tasks such as face recognition. Several studies investigate theoretical justifications for the benefit of active learning and the situations in which it is beneficial to learning, for example [2] and [3]. However in most cases these studies are based on severely restricted hyptothesis spaces and more general proofs are yet to be developed [33]. While the main goal of active learning is generally to reduce the number of samples required to achieve a certain level of test set performance [33], there are some studies that examine how active learning approaches can reduce overall generalization error, for example [6]. This motivates the idea for this paper that active learning methods can be used in a curriculum learning setting in order to improve the overall accuracy of deep models.

We have now introduced the main componentry that will be employed in the rest of the paper; specifically we will use methods motivated by uncertainty sampling in active learning to estimate sample difficulty scores which will be used to automatically construct learning curricula. In the following Chapter we will outline several recent studies which have tested methods for automating the process of curriculum construction, before laying out the methods and experimental results carried out for this report.

Chapter 3

Related Work

3.1 Self Paced Learning

3.2 Transfer Learning

3.3 Reinforcement Learning

3.4 Active Learning

Chapter 4

Bootstrapped Active Curricula

The first curriculum construction approach we investigate is what we term *bootstrapped active curricula (BAC)*, the name is chosen as it involves ‘bootstrapping’ a curriculum from a separate model. The intuition behind this approach is that, while it may be difficult to ascertain a-priori which samples are ‘hard’ or ‘easy’ (particularly for very large datasets where it is infeasible to analyse every sample), we can automatically infer which samples are difficult by first training a model on the data and then using an active learning uncertainty metric to investigate which samples the model is uncertain about classifying. Using prediction uncertainty to approximate difficulty, we can then construct a learning curriculum which can be used to train a new model, for example by splitting the training samples into separate tasks of increasing difficulty, as detailed below.

4.1 Curriculum Construction

In the BAC approach we train two models; the first, which we term the ‘baseline model’ and denote $\theta_{baseline}$, is trained to convergence using a standard mini-batch gradient descent optimisation on the entire available training set $\mathcal{T}$. We then use $\theta_{baseline}$ in conjunction with the Absolute Average Distance to Threshold uncertainty function as defined below in Section 4.1.1, scoring each training sample in $\mathcal{T}$. This produces $\mathbf{S}^{\theta_{baseline}}$, a vector of N scores, where N is the number of samples in $\mathcal{T}$, such that the i^{th} element of $\mathbf{S}^{\theta_{baseline}}$ is the output of the AADT uncertainty function for the i^{th} training sample inputs:

$$S_i^{\theta_{baseline}} = AADT_{\theta_{baseline}}(x_i) \tag{4.1}$$

where x_i are the inputs of the i^{th} training sample. We then abuse the function notation slightly to set $\mathbf{S}^{\theta_{baseline}} = AADT_{\theta_{baseline}}(\mathcal{T})$, i.e. signifying that $\mathbf{S}^{\theta_{baseline}}$ is the output of the AADT uncertainty function over the entire training set $\mathcal{T}$.

We then sort the training samples according to their score, producing an ordered training set $\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}$, such that the first training sample in $\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}$ is the sample which produces the highest value from the AADT function. We then split $\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}$ into equally sized 'tasks', with the number of tasks being a hyperparameter of the curriculum construction. The learning curriculum is then constructed from these tasks; the chosen approach is to split training into discrete phases, with the number of phases being equal to the chosen number of tasks. The first phase of the curriculum consists of training only on the first task, denoted $\mathcal{T}^1_{\mathbf{S}^{\theta_{baseline}}}$, in the second phase, the second task is added to the training samples, training the model on $\mathcal{T}^1_{\mathbf{S}^{\theta_{baseline}}} \cup \mathcal{T}^2_{\mathbf{S}^{\theta_{baseline}}}$. In the third phase (if the number of tasks is greater than 2), the next task is added, and so on until all tasks have been added and the final phase consist of training on the entirety of the original training set $\mathcal{T}$. Having constructed the BAC learning curriculum, we train a new 'curriculum' model', which we denote by $\theta_{curriculum}$, using the curriculum. Pseudocode for the BAC method is given in Section 4.1.3.

4.1.1 Average Absolute Distance to Threshold (AADT)

A popular active learning uncertainty method is to examine the proximity of the model's outputs to the classification bounday. The assumption is that samples that the model is uncertain about classifying will produce probabilities close to the classification boundary; indeed as mentioned in Section 2.5 the authors of [16] show that prediction variance is inversely proportional to the distance to the boundary. From a curriculum perspective we can estimate a sample's difficulty by the algorithm's uncertainty in predicting the class label, with uncertain samples being seen as hard, and vice versa. We thus define the uncertainty function *AADT* which calculates the absolute distance to classification threshold for the model outputs, averaged across all possible classes. Note that the function is dependent on a the output of the model in question, which is denoted θ.

$$AADT_{\theta}(x_i) = \frac{\sum_{c=1}^{C} \left| P_{\theta}(y_c|x_i) - \frac{1}{C} \right|}{C}. \tag{4.2}$$

Where $|.|$ represents the L1 norm/absolute value function. Here N is the number of training samples, C is the number of output classes and $P_{\theta}(y_c|x_i)$ is the output softmax probability for class y_c of the model θ, given input x_i. We also tested the average *square* of the distance to threshold, as opposed to the absolute distance to threshold, as well as testing the entropy of the outputs as an uncertainty measure, however results were extremely similar in all cases.

4.1.2 Phase Training Epochs

A naive approach to the BAC approach would be to train the curriculum model for the same number of epochs as the baseline model, split equally acrossing training phases. For example if the baseline model was trained for 100 epochs and we then constructed a two task curriculum, the curriculum model would be trained on the first task for the first 50 epochs, then on both tasks for the second 50 epochs. The issue with this method however is that, as during the first 50 epochs there are only half as many training samples, the curriculum model will not have as many parameter updates as the baseline model. To emphasize this, consider that if we were using stochastic gradient descent (i.e. a mini-batch size of 1) with a full training set of 1000 samples, the baseline model in this instance would have (# epochs * # training samples) parameter updates, i.e. 100 * 1000 = 100,000 parameter updates. The curriculum model on the other hand would have (50*500) parameter updates the first training phase and (50*1000) in the second, resulting in a total of (50*500) + (50*1000) = 75,000 parameter updates, 25% less updates than the baseline model. To address this, we increase the number of epochs in each phase by the ratio of the number of samples used in the phase to the size of the whole training set. Specifically, we set the number of training epochs in each phase as follows:

$$NumEpochs^t = \lfloor \frac{BaselineEpochs}{t} \rfloor \tag{4.3}$$

Where $BaselineEpochs$ is the number of epochs used to train the baseline model and $NumEpochs^t$ denotes the number of training epochs in the t^{th} training phase. For example, in the first phase, $NumEpochs^1 = \frac{BaselineEpochs}{1} = BaselineEpochs$. $\lfloor . \rfloor$ represents the floor function; as $\frac{BaselineEpochs}{i}$ will not always be integer, we round down the number of epochs in each phase. The curriculum model will therefore be trained for a higher number of epochs (precisely, $\sum_{t=1}^{NumTasks} \frac{1}{t}$ times more epochs), however the number of parameter updates will be equalised.

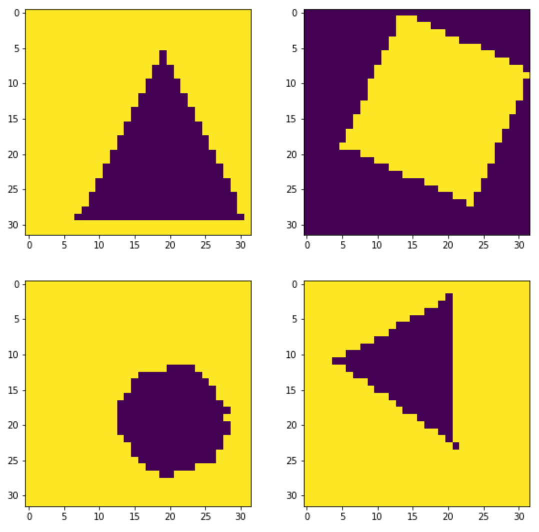
4.1.3 Pseudocode for BAC

Pseudocode for the boostrapped active curriculum method (assuming the use of the AADT uncertainty function and an easy to hard curriculum):

```
for $Num\_Epochs$ do
    train $\theta_{baseline}$ on training set $\mathcal{T}$
end for
$\mathbf{S}^{\theta_{baseline}} = AADT_{\theta_{baseline}}(\mathcal{T})$
$\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}} = \mathcal{T}$, sorted by $\mathbf{S}^{\theta_{baseline}}$ (descending order)
$Num\_Samples = |\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}|$
for $t = 0$ to $Num\_Tasks$ do
    $TaskStartIndex = Num\_Samples * \frac{t}{Num\_Tasks}$
    $TaskEndIndex = Num\_Samples * \frac{t+1}{Num\_Tasks}$
    $\mathcal{T}^{t}_{\mathbf{S}^{\theta_{baseline}}} = \mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}[TaskStartIndex : TaskEndIndex, :]$
end for
for $t = 0$ to $Num\_Tasks$ do
    $Num\_Epochs\_Task = \lfloor \frac{Num\_Epochs}{t} \rfloor$
    for $Num\_Epochs\_Task$ do
        train $\theta_{curriculum}$ on training set $\mathcal{T}^{0:t}_{\mathbf{S}^{\theta_{baseline}}}$
    end for
end for
```

4.2 Geometric Shapes Dataset

To test the boostrapped active curriculum approach we use the ‘Geometric Shapes’ dataset [30], as used in [43]. This dataset consists of 32x32 pixel images of geometric shapes, specifically ellipses, rectangles and triangles; the class labels are one-hot vectors which indicate to which of the three classes each sample belongs. As well as varying the shapes shown in the image, the samples also vary in colour, orientation, size and position. This dataset is used in [43] as it is easy to construct a predefined curriculum for geometric shapes; circles, squares and equilateral triangles represent regular, ‘easy’ versions of the broader classes of ellipses, rectangles and triangles, respectively. The authors of [43] train a curriculum model by first training only on an easy training set consisting only of circles, squares and equilateral triangles, before then training on a hard training set with the more general shapes. The easy and diffi-


(a) Samples from ‘easy’ training set

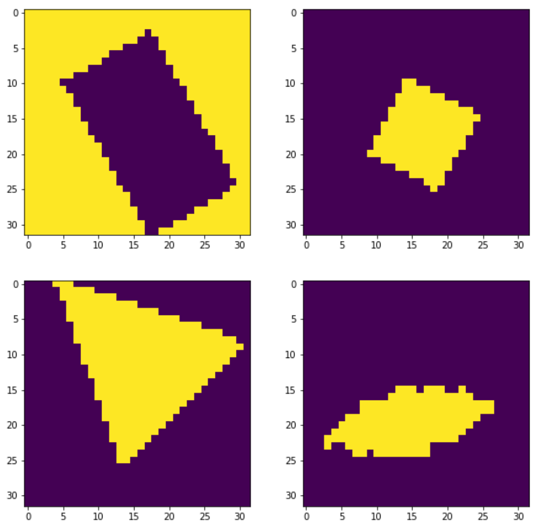

(b) Samples from ‘hard’ training set

Figure 4.1: Sample figures from the Geometric Shapes dataset

cult training sets consist of 10,000 training samples, giving a total of 20,000 training samples, while the test set, consisting only of hard samples, contains 5,000 images. The authors demonstrate that their approach outperforms an identical model trained only on the harder shapes; the curriculum model consistently achieves greater test accuracy, with the greatest improvement coming when the first half of the training epochs train on the easier shapes, and the second half the harder ones, as shown in Figure 3 of their paper. As discussed by the authors, one potential pitfall of their experiments is that the curriculum model has seen more samples than the benchmark model, as it has been trained on both the easy and the difficult training sets, to avoid this in our experiments, the training set we use is the union of both the easy and difficult samples. Our training set therefore consists of 20,000 geometric shape images, including both the easy, regular shapes and the more difficult shapes, the test set is unchanged however, consisting of the 5,000 difficult samples.

Some example images for the Geometric Shapes dataset are given in Figure 4.1 below, with Figure 4.1a showing examples from the easy training set featuring circles, squares and equilateral triangles, while Figure 4.1b shows samples from the more difficult training set of general shapes.

In the results Section 4.4, we analyse how succesful our tested curriculum method is at automatically identifying which samples come from the easy or hard training sets, investigating which training samples fall into the different tasks automatically constructed through the BAC curriculum method.

4.3 Experiments

In order to measure the effect of the curriculum on test performance, we set $\theta_{baseline}$ and $\theta_{curriculum}$ to have identical architectures, hyperparameters and initial weights, essentially minimizing any differences in training besides the learning curriculum. We use both the Average Absolute Distance to Threshold (AADT) function, detailed in Section 4.1.1, to score the training samples with the trained baseline model in each experiment. Furthermore, as well as testing 'easy to hard' curricula, where the training phases progress from $\mathcal{T}^1_{\mathbf{S}^{\theta_{baseline}}}$ to the final task, we also test the opposite approach, with the first training phase using only the hardest task, then incorporating the other, easier tasks throughout training. We run the experiments using the Geometric Shapes dataset, as introduced in Section 4.2, allowing us to compare results with the predefined curriculum as in [43]. To do so, as well as training the baseline and BAC curriculum models, we also train a model using the predefined curriculum method laid out in [43], specifically training on only the easy samples in for the first half of the training epochs, and only the hard samples in the second half. In order to better compare the predefined curriculum method from [43], we also run an adjusted version of their method; unlike the BAC approach, where we correct the difference in number of parameter updates between the curriculum model and the baseline model, the predefined curriculum model will end up undergoing significantly less parameter updates than our baseline (in fact it will have half as many updates). We therefore correct for this in two ways; in the first phase of training, in which the model is trained only on the easy images, lasts for twice as many epochs as in the unadjusted method (correcting for the fact that it consists of half as many samples as the full training set). The second phase of the adjusted model is then trained on the *full* training set, consisting of the union of the hard and easy samples. This should allow for a better comparison between the BAC models, the baseline model, and the predefined curriculum approach. We therefore train 5 separate models:

- Baseline Model - Trained on the full training set of easy and hard geometric shape images for all epochs, with standard mini-batch stochastic gradient descent.

- Easy to Hard Model - Trained with a boostrapped active curriculum construct from the above Baseline Model, beginning with the easiest/least uncertain task and including the other tasks throughout training.
- Hard to Easy Model - As above, but training begins with the hardest/most uncertain tasks before incoporating the easier tasks throughout training.
- Predefined Curriculum - As in [43], the first half of the training epochs use only the 10,000 easy training samples, while the second half train on only the 10,000 hard training samples.
- Adjusted Predefined Curriculum - As above, but the number of epochs in the first phase of training is doubled to account for the difference in parameter updates, and the second phase uses both easy and hard training samples for better comparison with the BAC models.

All models are tested on the same test set of 5000 images, all containing 'hard' geometric shapes; experiments are repeated multiple times with different weight initialisations. To analyse the effect of the curriculum on learning we report the accuracy and cross-entropy error of the different models over the held out test set. The exact model architecture is given in tables 5.1 - 5.3 below ('FC' = fully connected Layer):

Table 4.1: Geometric Shapes Dataset Model Architecture

Geometric Shapes Dataset Model Architecture							
	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7
Layer Type	FC	Dropout	FC	Dropout	FC	Dropout	FC
Units	300	NA	300	NA	300	NA	3
Activation	Tanh [1]	NA	Tanh	NA	Tanh	NA	Softmax

We also lay out the hyperparameters for the training procedures in table 4.2 below, experiments were carried out in Keras [5] with the Adam optimiser implemented with the default hyperparameters (except for the learning rate which is given in Table 4.2):

Architectures and hyperparameters were chosen and tuned in order to deliver a good level of performance without prohibitive training times, robustness tests were however carried out varying the model architectures and hyperparameters resulting in little change in the relative performance of the different methods.

[1]The Hyperbolic Tangent activation is chosen as this was this the activation used in [43].

Table 4.2: Experiment Hyperparameters

Experiment Hyperparameters						
	Epochs	Optimiser	Learning Rate	Dropout %	Batch Size	Num Tasks
GeoShapes	350	Adam	0.0001	0.25	32	2

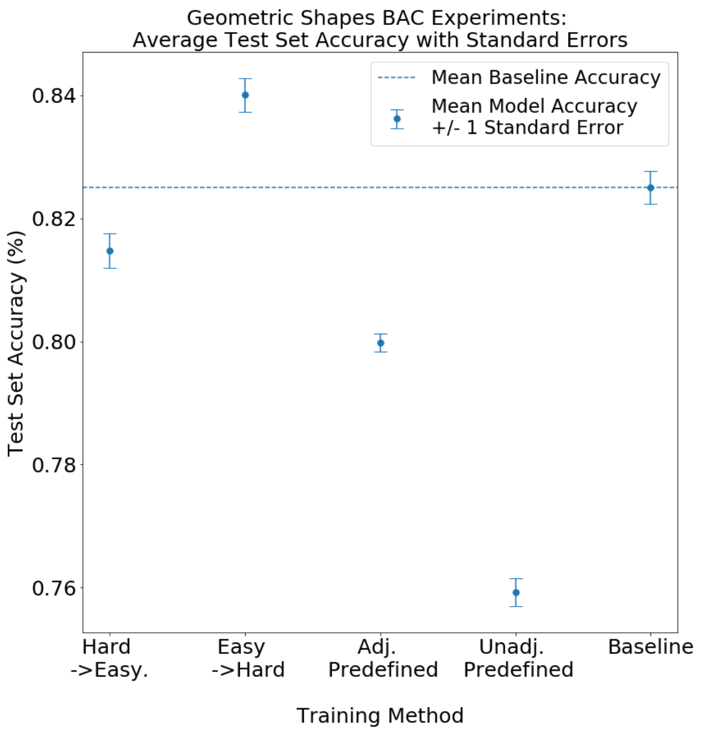
4.4 Results and Discussion

The results of running the bootstrapped active curriuculum experiments with the Geometric Shapes dataset are summarised in Table 4.3 and Figure 4.2 which shows the test set accuracies and cross entropy errors, including standard errors, derived from 24 experiments with different initialisations.

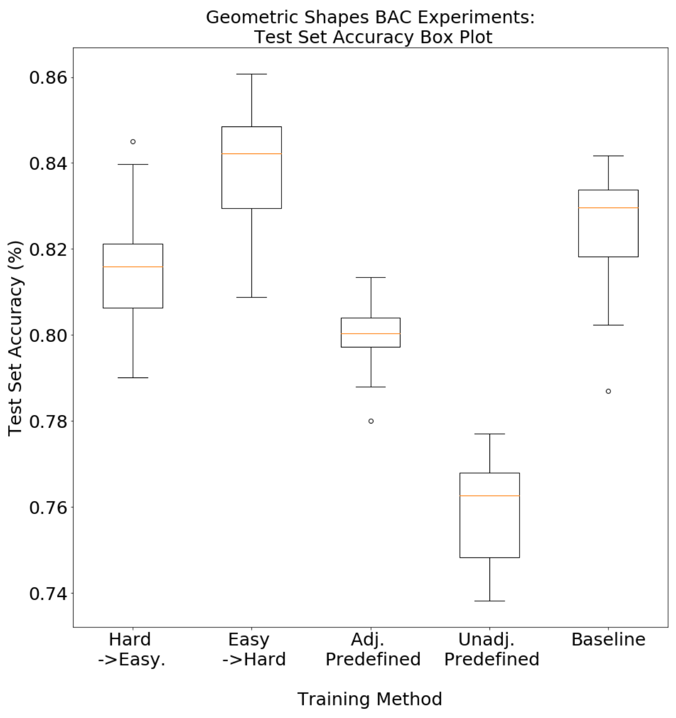
Table 4.3: Geometric Shapes BAC Results

GeoShapes BAC Results		
	Test Accuracy (%)	Test Cross-Entropy Error
Uniform Baseline	0.825 ± 0.00269	0.440 ± 0.00614
Easy to Hard Curriculum	0.840 ± 0.00267	0.398 ± 0.00670
Hard to Easy Curriculum	0.815 ± 0.00278	0.467 ± 0.00699
Adjusted Predefined Curriculum	0.800 ± 0.00143	0.486 ± 0.00409
Predefined Curriculum	0.757 ± 0.00279	0.576 ± 0.00489

We see from the results that the Easy to Hard curriculum model significantly outperforms the baseline model, with an average test accuracy improvement of around 2.5%, over 5 standard errors, higher than the baseline test performance. Cross-entropy error is also reduced, with the Easy to Hard curriculum test error over 6 standard errors below that of the baseline model. These results suggest that, not only is there a benefit to training the model with a curriculum, the BAC model seems to be succesful at identifying which samples should be used in the different curriculum learning phases. Conversely, the other curriculum methods all underperform the baseline model; the Hard to Easy curriculum in some ways acts as a control, showing that the improvement in the Easy to Hard model is driven by the order in which the tasks are included in training, as opposed to being a consequence of another part of the BAC method. It is interesting to note that both of the predefined curriculum models underperform the baseline model; the lower accuracy of the unadjusted method would seem to support the authors' suggestion in [43] that the performance differences in their experiments



(a) Mean Test Set Accuracies



(b) Test Set Accuracy Boxplots

Figure 4.2: Geometric Shapes Bootstrapped Active Curriculum (BAC) Experiment Results

may have been a result of their curriculum model having seen overall more training samples than their baseline. However, even our adjusted version of the predefined curriculum results in a significantly lower test accuracy than the baseline. This is interesting as this method is effectively trained in the same way as the BAC approach, however the tasks are predetermined by an intuitive sense of difficulty, as opposed to being derived from the baseline. This would suggest that, even for datasets where there as a somewhat clear distinction between hard and easy samples, a bootstrapped curriculum may be more useful. Potentially, this may because the BAC method specifically calculates which samples the model itself is uncertain in classifying, as opposed to assuming that our intuitive sense of difficulty corresponds to the best curriculum for the model. We would note however, that given the relatively low resolution of the images (as illustrated in Figure 4.1), there is perhaps less observable difference between the easy and hard training sets than might be assumed, and that this may contribute to the poor performance of the predefined curriculum.

With that in mind, we can analyse one of the the Easy to Hard curriculum experiments in more depth, in order to ascertain which samples are included in the two tasks. Recall that in the BAC method we train a baseline model, then use the uncertainty in the model's output class probabilities for the training samples, calculated using the AADT function defined in Section4.1.1, to score and rank the training samples. The training samples are then split into two tasks, the first consisting of the training samples that the model is least uncertain in classifying, and the second task containing the samples about which the model's outputs are most uncertain. We can analyse the composition of the two tasks to infer which samples the model is least/most uncertain about. Doing so for one of the experiments presents some interesting conclusions; first of all, there is a significant lack of balance in the target classes in the two tasks. In task 1, the easier task, 43% of the images are of triangles, while 31% are of ellipses and 26% are of rectangles, implying that the baseline model is significantly more confident in classifying whether or not an image shows a triangle than it is in classifying the other shapes. The bottom right image in Figure 4.1b illustrates why this may be, with an example of an ellipse that looks quite similar to a mis-shapen rectangle, again this is probably the result of the low resolution of the images. We also observe that 64% of the samples in the first task are from the easy training set of squares, circles and equilateral triangles, showing that the AADT function is somewhat succesful in separating the two training sets, however the first task still contains a substantial number of the harder samples. In appendix REF APPENDIX, we show a number of samples from either task, illustrating how the first, easier task is predominantly made up of regular shapes, as well as triangle, which appear to be easier to classify than rectangles and ellipses.

A potential enhancement to the BAC method would be to control the balance of the classes in either task, however if one class is indeed easier than the others to classify then it may be better for it to be over-represented in the easier task(s). While we used two tasks for the Geometric Shapes dataset as this corresponded to the number of tasks of the predefined curriculum, we also ran some initial experiments with a larger number of tasks. We found some variation in results with a larger number of tasks, with some choices of task number underperforming the baseline model, however further tests could be carried out in future work to ascertain how robust these results are and investigate what is driving the different performances.

4.5 Summary

To summarize, these experiments seem to support the hypothesis that training a deep model with a curriculum can reduce generalization error, and furthermore that using an active learning approach to inferring sample difficulty through model prediction uncertainty can be an effective way of automatically constructing such curricula. In particular such curriculum construction methods may even outperform a preconstructed curriculum using an intuitve sense of sample difficulty, as model uncertainty incorporates information about which samples the model itself finds difficult. One of the arguable drawbacks to the BAC method is that it requires that a baseline model is trained to convergence, in order to use its outputs to construct the curriculum to train the curriculum model. This effectively doubles the training time and, while this may not be too significant a computational burden in some problems, motivates the approach of the next chapter, which investigates whether or not a curriculum can be constructed dynamically throughout training, without first training a baseline model.

Chapter 5

Dynamic Active Curricula

The results laid out in Chapter 4 suggest that generalisation performance can indeed be improved through the use of a learning curriculum, and that using active learning approaches to score which training samples are hard or difficult can be an effective way of automatically constructing such curricula without manually analysing the training samples. The disadvantage with the BAC method from Chapter 4 however is that it is necessary to first train a ‘baseline’ model that can be used to derive a curriculum based on which samples it is uncertain about classifying. While in some cases this may not too significant a computational burden, it does effectively double the overall training time, motivating the approach laid out in this section. Specifically, we wish to investigate methods that can dynamically construct a curriculum throughout training, without needed to reference another model. We do this by calculating the model uncertainty on the training samples throughout training, using the uncertainty scores to construct dynamic curricula which evolve throughout training to focus on the samples that the model is more, or less, uncertain in its predictions. If succesful, these methods should lead to model performance which beats a the benchmark test set performance of a baseline model trained with normal mini-batch stochastic gradient descent on the entire training set, and hopefully will achieve results similar to those achieved by the bootstrapped active curricula from Chapter 4.

5.1 Curriculum Construction

5.1.1 Dynamic Task Curricula (DTC)

The first dynamic curriculum method we test we term *dynamic task curricula*; this approach is very similar to that of the BAC method laids out in Chapter 4, however instead of constructing tasks based on the uncertainty scores of a fully trained baseline model, tasks are constructed dynamically using the uncertainty of the curriculum model itself throughout training. To do this, we calculate the model's uncertainty in predicting the classes of the training samples in the full training set $\mathcal{T}$ using the AADT function 4.1.1 (or another active learning uncertainty measure) at the end of every epoch, then ranking the training sample by the the model's uncertainty. We then split the training data into separate tasks, with the first task consisting of the first $\frac{1}{T}$ samples where T is the chosen number of tasks. At the start of each epoch, we will obtain a new set of ordered tasks, $\mathcal{T}_i^1, \mathcal{T}_i^2, ..., \mathcal{T}_i^T$, where i is the current epoch, and T is the chosen number of tasks. Training is divided into T phases, such that, during epoch i of phase t, the model is trained on a training set consisting $\mathcal{T}_i^1 \cup \mathcal{T}_i^2 \cup ... \cup \mathcal{T}_j^t$, where t denotes the current training phase.The final training phase will therefore consist of training on the full training set, as $\mathcal{T} = \mathcal{T}_i^1 \cup \mathcal{T}_i^2 \cup ... \cup \mathcal{T}_i^T, \forall i$. As in the BAC curriculum method 4, we normalise the number of parameter updates in each phase by scaling the number of epochs relative to the number of samples in the training set for that phase. Similarly in Equation 4.3, the number of epochs in training phase t is set as follows:

$$NumEpochs^t = \lfloor \frac{TotalEpochs}{t} \rfloor \tag{5.1}$$

where $TotalEpochs$ is the number of epochs the model would need to be trained for on the whole training set in order to achieve the same number of parameter updates as the DTC curriculum method.

Note that we can rank the samples from low to high uncertainty or from high to low uncertainty; in the low to high case the first tasks will contain samples about which the model is confident in predicting, corresponding to dynamically constructing an easy to hard curriculum. Alternatively, ranking the samples from high to low uncertainty will produce a curriculum more similar to an active learning approach, focussing on training the model on areas of the input space that it is more uncertain about classifying. While the latter method may contradict some of the underlying motivations for curriculum learning, for example that learning with an easy to hard curriculum allows the model to explore a smoother version of the error space and better initialise the parameters, we would still argue that the hard to easy approach still qualifies as a curriculum method, and may be appropriate in instances where the model already has achieved a high level of expertise in the problem, and improvements are therefore more likely to be made by studying harder samples than easy ones. We therefore have two variations of the Dynamic Task Curriculum method; Easy to Hard DTC and Hard to Easy DTC. This approach is similar to that of *Self-Paced Learning* REF KOLLER ET AL, where they apply a similar approach with a latent SSVM model REF, however they implement their method by introducing a regularization term reflecting the difficulty of each sample into the objective function of the model, as opposed to employing an actual curriculum. As the model parameters are initialised randomly, it would be inappropriate to infer model uncertainty before the first epoch, as such, the first epoch is trained using the whole training set, and the use of the curriculum method begins starting with the second epoch. Pseudocode for the DTC method is given in section 5.1.1.1.

5.1.1.1 Pseudocode for the DTC curriculum

Pseudocode for the dynamic task curriculum method (assuming the use of the AADT uncertainty function and an easy to hard curriculum):

```
for $i = 1$ to $Num\_Epochs$ do
    $TrainingPhase = \lfloor \frac{i * Num\_Tasks}{Num\_Epochs} \rfloor$
    $PhaseEpochScale = \frac{Num\_Tasks}{TrainingPhase}$
    for $PhaseEpochScale$ do
        $\mathbf{S}^{\theta_{curriculum}} = AADT_{\theta_{curriculum}}(\mathcal{T})$
        $\mathcal{T}_{\mathbf{S}^{\theta_{curriculum}}} = \mathcal{T}$, sorted by $\mathbf{S}^{\theta_{curriculum}}$ (descending order)
        $Num\_Samples = |\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}|$
        for $t = 0$ to $Num\_Tasks$ do
            $TaskStartIndex = Num\_Samples * \frac{t}{Num\_Tasks}$
            $TaskEndIndex = Num\_Samples * \frac{t+1}{Num\_Tasks}$
            $\mathcal{T}^{t}_{\mathbf{S}^{\theta_{baseline}}} = \mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}[TaskStartIndex : TaskEndIndex, :]$
        end for
        train $\theta_{curriculum}$ on training set $\mathcal{T}^{0:TrainingPhase}_{\mathbf{S}^{\theta_{baseline}}}$
    end for
end for
```

5.1.2 Dynamic Sampling Curricula (DSC)

The second dynamic curriculum construction method we test is *Dynamic Sampling Curricula (DSC)*; one of the potential problems with some curriculum and active learning methods is that of *diversity* REF PAPER ON DIVERSITY. Diversity refers to how well the whole training set is represented in the samples used to train the model; if a sample selection method is extremely undiverse then it may end up selecting only a very small selection of the training samples, leading to the model not training on a suitably broad set of training samples, resulting in high generalization error. One way to approach this problem is to avoid deterministic methods of selecting training examples, and instead sample from the training data using some probability distribution. Mini-batch stochastic gradient descent REF? is one such method, sampling batches of examples from the training data using a uniform probability distribution, however by varying the sampling distribution away from a uniform distribution we can bias training towards different samples, resulting in them being selected more or less often than others.

We implement the DSC curriculum method by biasing the sampling probability proportionally to the model uncertainty in predicting the labels of the sample. In particular, at the start of every epoch we use an uncertainty function (for example the AADT function introduced in 4.1.1) to infer the model's current uncertainty in predicting the labels of the training data, giving $\mathbf{S}^{\theta_{baseline}}$, a vector of N scores as in Equation 4.1, where the N is the number of samples in the full training set $\mathcal{T}$ and the i^{th} element of $\mathbf{S}$ corresponds to the ouput of the AADT function for the i^{th} training input. We then pass the vector of scores through a softmax function, effectively transforming the scores into probabilities that can be used as the sampling probability function (defined assuming the uncertainty function used is the AADT function):

$$\tilde{p}_j(x_i|\theta_j) = \frac{exp(\frac{AADT_{\theta_j}(x_i)}{\tau})}{\sum_k^N exp(\frac{AADT_{\theta_j}(x_k)}{\tau})} \tag{5.2}$$

where θ_j is the model being trained with the curriculum after $j-1$ epochs, $\tilde{p}_j(x_i|\theta)$ is the sampling probability of training sample i in the j^{th} training epoch, and τ is the softmax temperature parameter that can be set as a hyperparameter or tuned each epoch to achieve a target diversity level in the sampling probability. The sampling probability is updated at the start of each epoch and training proceeds as with uniform mini-batch stochastic gradient descent, using the sampling probabilities calculated in Equation 5.3. As we wish to bias training towards certain certain/uncertain samples, we also do not follow the typical SGD practice of sampling without replacement, instead sampling with replacement. As sampling with replacement implies that some samples will probably not be used in every epoch, this in effect introduces a stochastic curriculum where, instead of removing samples entirely from certain training phases, we instead decrease their probability of being sampled during an epoch. Unlike the DTC curriculum method in Section 5.1.1, the DSC method does not have distinct training phases; the sampling probability is biased throughout the entirety of training. As we are no longer using training phases with smaller task training sets we do not need to modify the number of epochs relative to the baseline as we did with the BAC method in Chatper 4 and the DTC method. As with the DTC method in the previous section, we train the model using the whole training set, with uniform mini-batch SGD, for one epoch, before we begin estimating model uncertainty and training with the DSC curriculum. Pseudocode for the DSC curriculum method is given in Section 5.1.2.1.

5.1.2.1 Pseudocode for the DSC curriculum

Pseudocode for the dynamic sampling curriculum method, except for the softmax temperature control method, given in REF (assuming the use of the AADT uncertainty function and an easy to hard curriculum):

```
for $j = 1$ to $Num\_Epochs$ do
    $\forall x_i \in \mathcal{T}, \tilde{p}_j(x_i|\theta_j) = \frac{exp(\frac{AADT_{\theta_j}(x_i)}{\tau})}{\sum_k^N exp(\frac{AADT_{\theta_j}(x_k)}{\tau})}$
    train $\theta_{curriculum}$ on training set $\mathcal{T}$, sampled with probabilities $\tilde{p}(.|\theta_j)$
end for
```

5.1.3 Dynamic Sampled Task Curricula (DSTC)

The final dynamic curriculum method implemented combines the the DTC and DSC methodologies, into an approach we term *dynamic sampled task curricula (DSTC)*. As in the dynamic task curricula method, at the start of every epoch we calculate the model uncertainty on the samples in the training set $\mathcal{T}$, dividing the training set up into several equally sized tasks of increasing or decreasing uncertainty. Again as in the DTC approach, the model is trained in several phases; in the first phase only the first task is using to train the model, then in the second phase the first two tasks are used, and so on until the entirety of $\mathcal{T}$ is being used. In the DSTC approach however, instead of sampling uniformly from training samples used in each phase, we bias the sampling probability as in the DSC model. We therefore calculate the sampling probability for sample x_i in epoch j as follows, assuming j falls in phase t:

$$\tilde{p}_j(x_i|\theta_j) = \begin{cases} \frac{exp(\frac{AADT_{\theta_j}(x_i)}{\tau})}{\sum_k^K exp(\frac{AADT_{\theta_j}(x_k)}{\tau})} & \text{if } x_i \in \mathcal{T}_j^1 \cup \mathcal{T}_j^2 \cup ... \cup \mathcal{T}_j^t \\ 0 & \text{if } x_i \notin \mathcal{T}_j^1 \cup \mathcal{T}_j^2 \cup ... \cup \mathcal{T}_j^t \end{cases} \tag{5.3}$$

where K is the number of training samples in the phase training set $\mathcal{T}_j^1 \cup \mathcal{T}_j^2 \cup ... \cup \mathcal{T}_j^t$, (assuming we are using the AADT uncertainty function to estimate model uncertainty).

The DSTC method therefore combines the DTC and DSC methods, dynamically splitting the training data into uncertainty based tasks, and then sampling from the tasks proportionally to the uncertainty of the samples relative to the rest of the task. DSTC could therefore be considered a more general version of the DSC method, where the DSC method is simply a special case of DSTC, where the number of tasks is equal to one. Pseudocode for the DSTC method is given in the following section.

5.1.3.1 Pseudocode for the DSTC curriculum

```
for $i = 1$ to $Num\_Epochs$ do
    $TrainingPhase = \lfloor \frac{i * Num\_Tasks}{Num\_Epochs} \rfloor$
    $PhaseEpochScale = \frac{Num\_Tasks}{TrainingPhase}$
    for $PhaseEpochScale$ do
        $\mathbf{S}^{\theta_{curriculum}} = AADT_{\theta_{curriculum}}(\mathcal{T})$
        $\mathcal{T}_{\mathbf{S}^{\theta_{curriculum}}} = \mathcal{T}$, sorted by $\mathbf{S}^{\theta_{curriculum}}$ (descending order)
        $Num\_Samples = |\mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}|$
        for $t = 0$ to $Num\_Tasks$ do
            $TaskStartIndex = Num\_Samples * \frac{t}{Num\_Tasks}$
            $TaskEndIndex = Num\_Samples * \frac{t+1}{Num\_Tasks}$
            $\mathcal{T}^t_{\mathbf{S}^{\theta_{baseline}}} = \mathcal{T}_{\mathbf{S}^{\theta_{baseline}}}[TaskStartIndex : TaskEndIndex, :]$
        end for
        $\forall x_i \in \mathcal{T}^{0:TrainingPhase}, \tilde{p}_j(x_i|\theta_j) = \frac{exp(\frac{AADT_{\theta_j}(x_i)}{\tau})}{\sum_k^N exp(\frac{AADT_{\theta_j}(x_k)}{\tau})}$
        train $\theta_{curriculum}$ on training set $\mathcal{T}^{0:TrainingPhase}_{\mathbf{S}^{\theta_{baseline}}}$, sampled with probabilities $\tilde{p}(.|\theta_j)$
    end for
end for
```

5.1.4 BALD Active Score Function

As well as using the AADT uncertainty function 4.1.1, we will also use the BALD acquisition function[18] [14]. Recent advances in Bayesian neural networks and variational inference motivate an alternative approach to measuring uncertainty [12]; while the distance to classification threshold may encapsulate classification uncertainty for samples close to the boundary, it does not consider the uncertainty associated with analysing samples from parts of the feature space that is not represented in the training data [12]. One approach to this problem is given by [13], where they motivate using the *Monte Carlo dropout* method as a way of approximating variational inference in neural networks. As with the usual dropout procedure [35], weights are randomly set to zero throughout the training phase, however, unlike the usual approach, dropout is maintained at the test stage, and a number of forward passes are carried out, resulting in a distribution of outputs. The resultant distribution can subsequently be analysed to infer which test samples the model is more or less confident in predicting, for example by comparing the variance of the output distributions. In [14], the authors use the MC dropout method to implement the BALD acquisition function, which queries points which “maximise the mutual information between predictions and model posterior”, identifying samples that have a high probability of being placed into different classes in the different stochastic forward passes. One interpration of the BALD method is that it is similar to the ‘Query by Committee’ active learning methods [33], with the different forward passes representing different models’ votes.

We calculate the BALD active score function as follows, as in [14]:

$$BALD_\theta(x_i) = -\sum_c^C \bar{P}_\theta(y_c|x_i)\log(\bar{P}_\theta(y_c|x_i)) + \frac{1}{M}\sum_m^M(\sum_c^C P_\theta^m(y_c|x)\log(P_\theta^m(y_c|x))), \tag{5.4}$$

and

$$\bar{P}_\theta(y_c|x_i) = \frac{\sum_m^M P_\theta^m(y_c|x)}{M}. \tag{5.5}$$

Here M is the number of stochastic forward passes carried out and $P_\theta^m(y_c|x)$ is the softmax probability of class c from the m^{th} forward pass. The score can therefore be interpreted as the difference between the entropy of the average softmax output and the average entropy of the output of each forward pass [14].

5.1.5 Softmax temperature

In order to homogenize the outputs of the different active score functions, we pass the the scores through a softmax functions, resulting in an output of softmax probabilities summing to 1. Using the softmax function also allows us to use the softmax temperature in order to control the diversity of the sampling probabilities. A common issue with active learning is that the acquisition functions can end up sampling from an unrepresentative subset of the input space, resulting in significant bias in the training of the model [33]. We control this effect by using the softmax temperature to target a preset *maximum probability ratio*, defined as follows:

$$MaxRatio = \frac{\max_i P_\theta(x_i)}{\min_i P_\theta(x_i)}. \tag{5.6}$$

To do this we begin with a softmax temperature of 1, then calculate the current maximum probability ratio and increment the temperature until the target ratio is achieved. Pseudocode is given below:

PSEUDOCODE FOR SOFTMAX CONTROL

The downside to this method is it could be skewed by outlier probabilities; if there were one sample with an extremely large sampling probability this method would still achieve a target maximum probability ratio, without achieving sufficient diversity in the sampling probabilities. We investigated using winsorization [15] as an outlier removal technique prior to tuning the temperature parameter, however, as the active score functions we use generally did not result in outliers, this did not affect results. While we do not lay out a thorough analysis of this in this paper, we did found that controlling the maximimum probability ratio throughtout training helped to stabilise model performance in the DSC and DSTC curriculum methods. Without this softmax temperature control approach, for example by simply setting the softmax temperature to a constant value of 1, there were times when the sampling probabilities in the DSC and DSTC methods were such that either the maximum probability ratio grew extremely large, resulting in a lack of diversity in the sampling, or the ratio was so small as to render the biased sampling method negligible. In the experiments laid out below, we use a target maximum probability ratio of 10, which we generally found led to a good level of diversity without limiting the impact of the sampling bias.

5.2 Datasets

In order to test the methods on a broader range of image classifiction tasks, as well as testing the different dynamic active curriculum methods on the Geometric Shapes data, introduced in Section 4.2, we also test the curriculua on the MNIST and CIFAR 10 datasets, detailed below:

5.2.1 MNIST

The MNIST dataset [25] contains images of handwritten digits, consisting of a training set of 60,000 training images and 10,000 test images (in our experiments we further split the training set into a training set and a validation split using 75% training, 25% validation split). The input data are 28x28 greyscale pixel values, while the target labels are one-hot vectors indicating which digit is written in the image, from 0 to 9. MNIST is a popular image classification task due to its relative simplicity, with state of the art architectures achieving test accuracies well in excess of 99%. Several examples images from the MNIST dataset are showin in Figure 5.1.

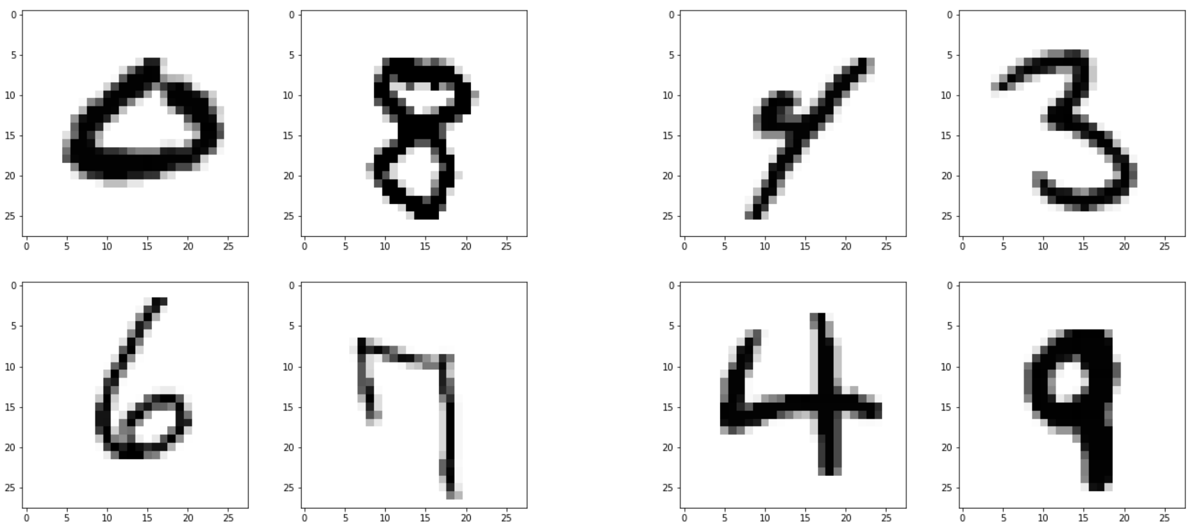


Figure 5.1: Sample figures from the MNIST dataset

5.2.2 CIFAR 10

The CIFAR 10 dataset [23] is another image classification dataset; the inputs for each samples are 32x32x3 RGB colour pixel values (3 separate channels for the Red, Green and Blue colour value), while the labels specify the class of each image. The possible classes are ‘airplane’, ‘automobile’, ‘bird’, ‘cat’, ‘deer’, ‘dog’ ,’frog’, ‘horse’, ‘ship’ and truck’. As can be seen in Figure 5.2, the images are significantly downsampled, resulting in images that can be difficult even for a human observer to classify, resulting in a significantly more difficult task than either MNIST or the Geometric Shapes database. For this dataset we use convolutional networks, as introduced in Section 2.2.2, this will both improve the results of the experiments with CIFAR 10, but will also test the performance of the curriculum methods with a different type of network.

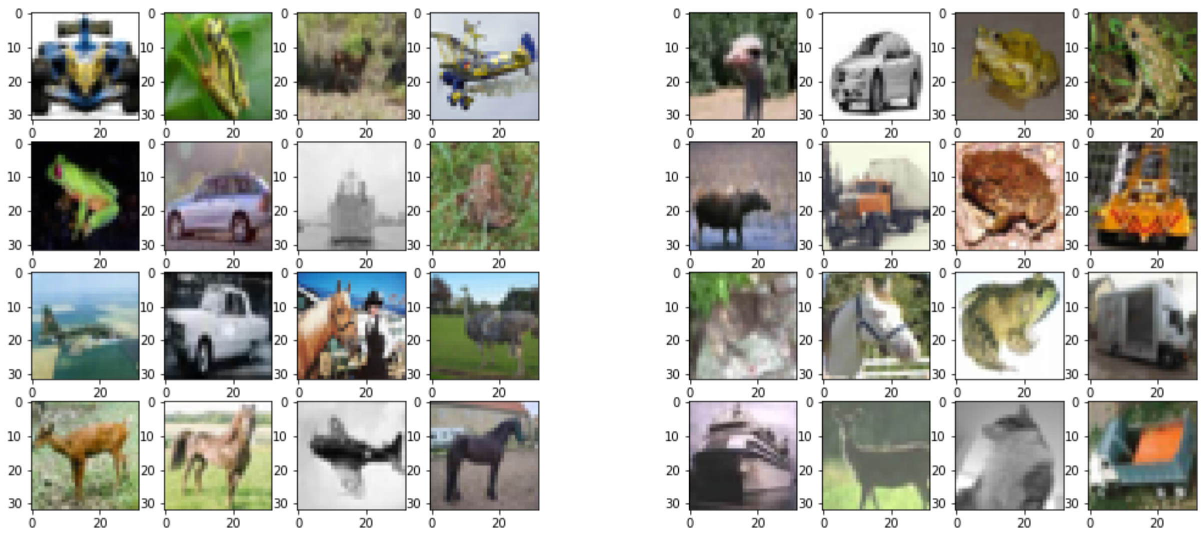


Figure 5.2: Sample figures from the CIFAR 10 dataset

5.3 Experiments

We test the different dynamic curriculum methods introduced above on the Geometric Shapes, MNIST and CIFAR 10 image classification datasets, as introduced in Sections 4.2, 5.2.1 and 5.2.2, respectively. For each dataset we used the validation set to construct an architecture and hyperparameters which achieved reasonable accuracies, then used those setting in the experiments for the different curriculum methods, the architectures for each dataset are laid out in Tables 5.1 to 5.3. Note that in the case of CIFAR, the convolutional layer (denoted “conv”) are following by a max pooling layer [28] and a batch normalization layer [20], and ReLU denotes the Rectified Linear Unit activation function[29].

We run experiments across all three datasets using the curriculum methods laid out above, testing both ‘easy to hard’ curricula and ‘hard to easy’ curricula, as well as tested using different uncertainty functions. In each case we also train a baseline model with an equal number of epochs to provide a comparison for the curriculum test

Table 5.1: Geometric Shapes Dataset Model Architecture

Geometric Shapes Dataset Model Architecture							
	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7
Layer Type	FC	Dropout	FC	Dropout	FC	Dropout	FC
Units	300	NA	300	NA	300	NA	3
Activation	Tanh	NA	Tanh	NA	Tanh	NA	Softmax

Table 5.2: MNIST Dataset Model Architecture

MNIST Dataset Model Architecture							
	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7
Layer Type	FC	Dropout	FC	Dropout	FC	Dropout	FC
Units	300	NA	300	NA	300	NA	3
Activation	ReLU	NA	ReLU	NA	ReLU	NA	Softmax

Table 5.3: CIFAR Dataset Model Architecture

CIFAR Dataset Model Architecture							
	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7
Layer Type	Conv	Conv	Flatten	Dropout	FC	Dropout	FC
Units	50	50	NA	NA	100	NA	10
Activation	ReLU	ReLU	NA	NA	ReLU	NA	Softmax
Kernel Size	3x3	3x3	NA	NA	NA	NA	NA

performance. We did not test all permutations of curricula and uncertainty functions as it became clear in early experiments some cases that there was little difference from the baseline performance, the experiments for which we performed repeated tests and presents present results are as follows:

- Geometric Shapes, using the AADT uncertainty function:
  - Dynamic Task Curriculum, Easy to Hard
  - Dynamic Task Curriculum, Hard to Easy
  - Dynamic Sampling Curriculum, Hard to Easy
  - Dynamic Sampled Task Curriculum, Hard to Easy
  - Uniform Sampling Baseline

- MNIST, using the AADT uncertainty function:
  - Dynamic Task Curriculum, Easy to Hard
  - Dynamic Sampling Curriculum, Easy to Hard
  - Dynamic Sampled Task Curriculum, Easy to Hard
  - Dynamic Task Curriculum, Hard to Easy
  - Dynamic Sampling Curriculum, Hard to Easy
  - Dynamic Sampled Task Curriculum, Hard to Easy
  - Uniform Sampling Baseline
- CIFAR 10, using the BALD uncertainty function[1]:
  - Dynamic Task Curriculum, Easy to Hard
  - Dynamic Sampled Task Curriculum, Easy to Hard
  - Dynamic Task Curriculum, Hard to Easy
  - Dynamic Sampled Task Curriculum, Hard to Easy
  - Uniform Sampling Baseline

Table 5.4: Experiment Hyperparameters

Experiment Hyperparameters						
	Epochs	Optimiser	Learning Rate	Dropout %	Batch Size	Num Tasks
Geometric Shapes	350	Adam	0.0001	0.25	32	2
MNIST	50	Adam	0.0001	0.25	50	2
CIFAR 10	50	Adam	0.0001	0.25	100	2

We run the experiments 10 times with different random initialistions, reporting average results and standard errors. In Table 5.4 we lay out the other hyperparameters for the experiments. Note that in each case we keep the number of tasks at 2, results were robust to differing the number of tasks however as we did not have time to do a full analysis we decided to keep the experimental setups relatively homogeneous in order to avoid tinkering with hyperparameters to achieve positive results. As in Chapter 4, experiments were carried out in Keras [5] and the Adam optimiser is implemented with the default hyperparameters (except for the learning rate which is given in Table 5.4).

[1]Tests for CIFAR 10 with the AADT function resulted in little difference with from baseline, however using the BALD uncertainty function improved performance significantly.

5.4 Results and Discussion

The results for the experiments are given in Tables 5.5 to 5.7[2], and illustrated in Figures 5.3 and 5.4. Note that the denotation 'easy' implies that the curriculum was trained on an easy-hard curriculum, with increasingly difficult tasks in the DTC case and by biasing sampling towards easy samples in the DSC method, and similarly for the DSTC method. Conversely, 'Hard' denotes that the curriculum used was a hard to easy curriculum.

Table 5.5: Geometric Shapes Dynamic Active Curricula Results

GeoShapes Dynamic Active Curricula Results		
	Test Accuracy (%)	Test Cross-Entropy Error
Uniform Baseline	0.826 ± 0.00281	0.437 ± 0.00696
DSC - Hard	0.846 ± 0.00369	0.391 ± 0.00827
DTC - Hard	0.865 ± 0.00157	0.362 ± 0.00350
DSTC - Hard	0.866 ± 0.00192	0.355 ± 0.00502
DTC - Easy	0.830 ± 0.00198	0.420 ± 0.00469

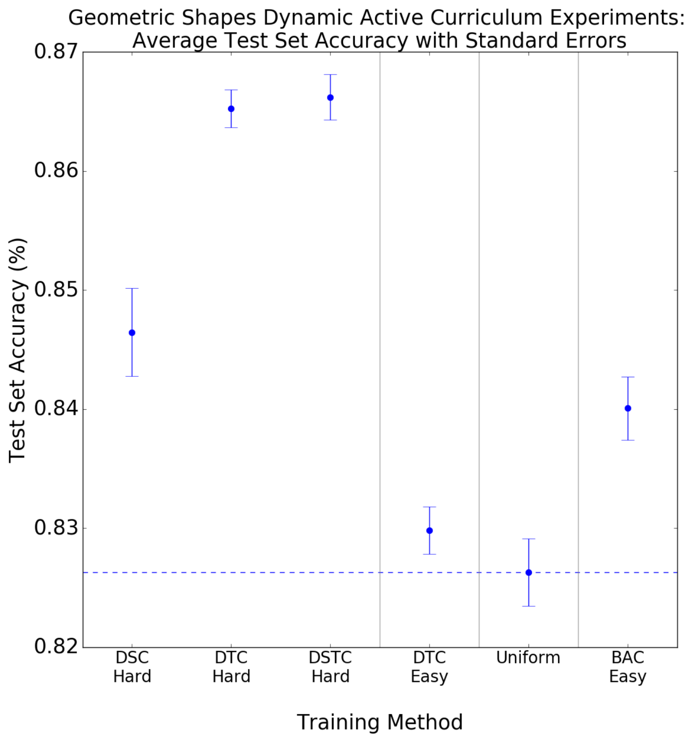
Table 5.6: MNIST Dynamic Active Curricula Results

MNIST Dynamic Active Curricula Results		
	Test Accuracy (%)	Test Cross-Entropy Error
Uniform Baseline	0.975 ± 0.000188	0.229 ± 0.000562
DSC - Hard	0.977 ± 0.000216	0.250 ± 0.000497
DTC - Hard	0.977 ± 0.000216	0.227 ± 0.000611
DSTC - Hard	0.979 ± 0.000327	0.251 ± 0.000736
DSC - Easy	0.970 ± 0.000240	0.227 ± 0.00102
DTC - Easy	0.970 ± 0.000276	0.224 ± 0.000696
DSTC - Easy	0.964 ± 0.000314	0.228 ± 0.000595

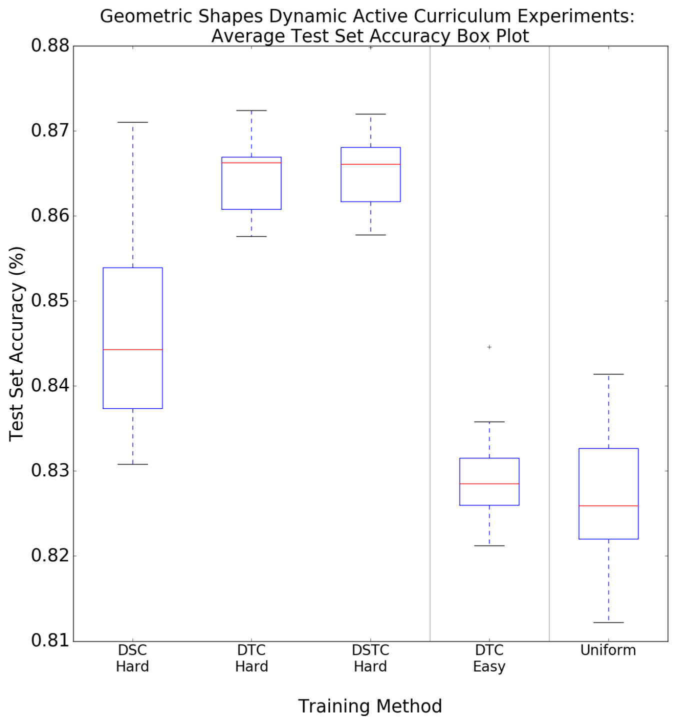
[2]Due to the experimental setup for the CIFAR 10 experiments, only test accuracies were recorded.

Table 5.7: CIFAR 10 Dynamic Active Curricula Results

CIFAR 10 Dynamic Active Curricula Results	
	Test Accuracy (%)
Uniform Baseline	0.713 $\pm$ 0.00439
DTC - Hard	0.730 $\pm$ 0.00132
DSTC - Hard	0.725 $\pm$ 0.00129
DTC - Easy	0.727 $\pm$ 0.00197
DSTC - Easy	0.722 $\pm$ 0.00429

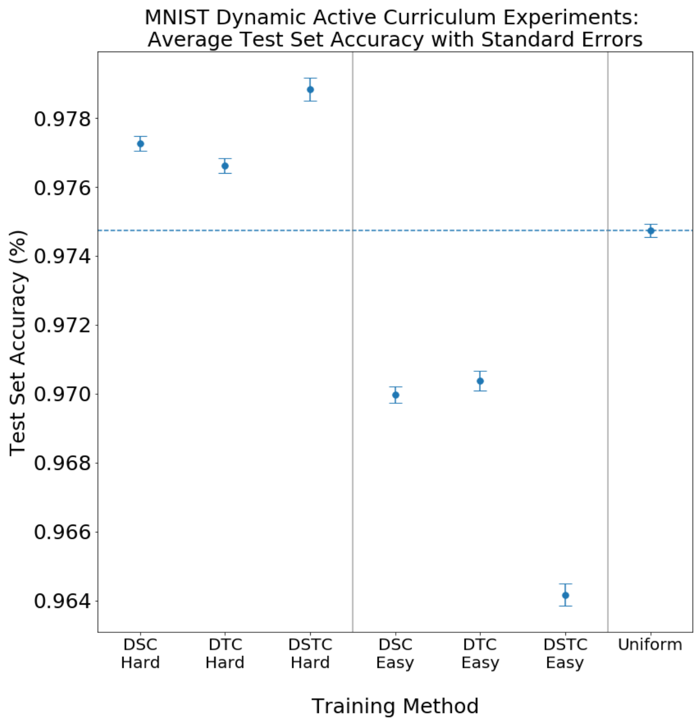


(a) Mean Test Set Accuracies

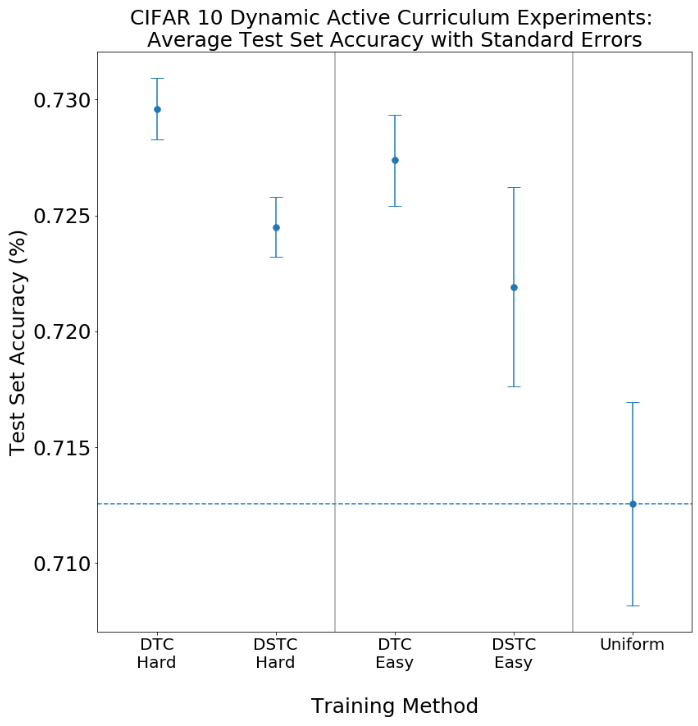


(b) Test Set Accuracy Boxplots

Figure 5.3: Geometric Shapes Dynamic Active Curriculum (DAC) Experiment Results



(a) Mean Test Set Accuracies



(b) Test Set Accuracy Boxplots

Figure 5.4: MNIST and CIFAR 10 Dynamic Active Curriculum (DAC) Experiment Results

Chapter 6

Conclusion and Further Work

Bibliography

[1] V. Avramova. Curriculum learning with deep convolutional neural networks, 2015.

[2] M.-F. Balcan, A. Beygelzimer, and J. Langford. Agnostic active learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 65–72. ACM, 2006.

[3] M.-F. Balcan, S. Hanneke, and J. W. Vaughan. The true sample complexity of active learning. *Machine learning*, 80(2-3):111–139, 2010.

[4] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[5] F. Chollet et al. Keras, 2015.

[6] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

[7] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[8] E.L.Allgower and K.Georg. *Numerical continuation methods. An introduction*. Springer-Verlag, 1980.

[9] J. L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71 – 99, 1993.

[10] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160, 2009.

[11] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[12] Y. Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[13] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[14] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

[15] D. Ghosh and A. Vogt. Outliers: An evaluation of methodologies. In *Joint statistical meetings*, pages 3455–3460. American Statistical Association San Diego, CA, 2012.

[16] A. H-S Chang, E.Learned-Miller. Active bias: Training more accuracy neural networks by emphasizing high variance samples. *Advances in Neural Information Processing Systems*, 31:1–122, 2017.

[17] T. Hastie, R. Tibshirani, and J. Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009.

[18] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[19] E. I.H.Witten and M.A.Hall. *DATA MINING. Practical Machine Learning Tools and Techniques*. Morgan Kaufman, 3 edition, 2011.

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

[21] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*, pages 2078–2086, 2014.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[24] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.

[25] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[26] J. Louradour and C. Kermorvant. Curriculum learning for handwritten text line recognition. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 56–60. IEEE, 2014.

[27] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[28] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*, pages 342–347. IEEE, 2011.

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

[30] J. O.Breuleux and F.Bastien. Geometric shapes database. 2008.

[31] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[32] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015.

[33] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[34] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International Conference on Machine Learning*, pages 71–79, 2013.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[36] S.Theodoridis and K.Koutroumbas. *Pattern Recognition.* Academic Press, 4 edition, 2009.

[37] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[38] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[39] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[40] K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2017.

[41] D. Weinshall and G. Cohen. Curriculum learning by transfer learning: Theory and experiments with deep networks. *arXiv preprint arXiv:1802.03796*, 2018.

[42] J. Yang et al. Automatically labeling video data using multi-class active learning. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 516–523. IEEE, 2003.

[43] R. Y.Bengio, J.Louradour and J.Weston. Curriculum learning. *Procedures of the International Conference on Machine Learning*, 26:41–48, 2009.