DEGREE PROJECT IN MACHINE LEARNING 120 CREDITS, SECOND CYCLE

STOCKHOLM, SWEDEN 2015

Curriculum Learning with Deep Convolutional Neural Networks

VANYA AVRAMOVA

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

CURRICULUM LEARNING WITH DEEP CONVOLUTIONAL NEURAL NETWORKS

VANYA AVRAMOVA
avramova@kth.se

DD221X, Master's Thesis in Computer Science (30 ECTS credits)
Master's Programme in Machine Learning 120 credits
Department of Computer Science and Communication

Supervisor at CSC: Josephine Sullivan
Examiner: Stefan Carlsson

KTH Royal Institute of Technology Stockholm
September 2015

ABSTRACT

Curriculum learning is a machine learning technique inspired by the way humans acquire knowledge and skills: by mastering simple concepts first, and progressing through information with increasing difficulty to grasp more complex topics. Curriculum Learning, and its derivatives Self Paced Learning (SPL) and Self Paced Learning with Diversity (SPLD), have been previously applied within various machine learning contexts: Support Vector Machines (SVMs), perceptrons, and multi-layer neural networks, where they have been shown to improve both training speed and model accuracy. This project ventured to apply the techniques within the previously unexplored context of deep learning, by investigating how they affect the performance of a deep convolutional neural network (ConvNet) trained on a large labeled image dataset. The curriculum was formed by presenting the training samples to the network in order of increasing difficulty, measured by the sample's loss value based on the network's objective function. The project evaluated SPL and SPLD, and proposed two new curriculum learning sub-variants, p-SPL and p-SPLD, which allow for a smooth progresson of sample inclusion during training. The project also explored the "inversed" versions of the SPL, SPLD, p-SPL and p-SPLD techniques, where the samples were selected for the curriculum in order of *decreasing* difficulty.

The experiments demonstrated that all learning variants perform fairly similarly, within $\approx$ 1% average test accuracy margin, based on five trained models per variant. Surprisingly, models trained with the inversed version of the algorithms performed slightly better than the standard curriculum training variants. The SPLD-Inversed, SPL-Inversed and SPLD networks also registered marginally higher accuracy results than the network trained with the usual random sample presentation. The results suggest that while sample ordering does affect the training process, the optimal order in which samples are presented may vary based on the data set and algorithm used.

The project also investigated whether some samples were more beneficial for the training process than others. Based on sample difficulty, subsets of samples were removed from the training data set. The models trained on the remaining samples were compared to a default model trained on all samples. On the data set used, removing the "easiest" 10% of samples had no effect on the achieved test accuracy compared to the default model, and removing the "easiest" 40% of samples reduced model accuracy by only $\approx$ 1% (compared to $\approx$ 6% loss when 40% of the "most difficult" samples were removed, and $\approx$ 3% loss when 40% of samples were randomly removed). Tak-

ing away the "easiest" samples first (up to a certain percentage of the data set) affected the learning process less negatively than removing random samples, while removing the "most difficult" samples first had the most detrimental effect. The results suggest that the networks derived most learning value from the "difficult" samples, and that a large subset of the "easiest" samples can be excluded from training with minimal impact on the attained model accuracy. Moreover, it is possible to identify these samples early during training, which can greatly reduce the training time for these models.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude and appreciation to my supervisor at CVAP, Prof. Josephine Sullivan, for her insightful guidance, patience and encouragement during the course of this project. I would like to thank Hossein Azizpour for the helpful suggestions, code reviews, and his enthusiasm about my work. I am also grateful to Yang Zhong, my lab mate, for the lively discussions on ConvNets and Deep Learning.

I want to acknowledge and extend my gratitude to my former colleagues at Microsoft, Dimitre Novatchev and Johan Sundström. I could not have undertaken this degree without their support.

I wish to thank my dear friends Morgan Svensson and Andrea de Giorgio – for their pure awesomeness, and for making this journey so memorable.

To my parents: for their unconditional love and support, and the inspiration and center of calm they have provided throughout my life.

To my spouse and daughter: for nonchalantly dropping everything and moving continents so I can pursue this, for making the entire venture possible, and for putting up with me throughout the process.

CONTENTS

List of Figures viii
List of Tables xi
Listings xi

i INTRODUCTION 1

1 INTRODUCTION 2
1.1 Motivation . . . 3
1.2 Problems Addressed in the Thesis . . . 4
1.3 Organization of the Report . . . 5

ii THEORY AND BACKGROUND 6

2 THEORY AND BACKGROUND 7
2.1 A Brief History of Feed Forward Neural Networks . . . 7
2.2 Convolutional Neural Networks . . . 14
2.2.1 Network topology . . . 15
2.2.2 Network layers . . . 16
2.2.3 Training algorithm (Learning the filters) . . . 19
2.3 Curriculum-based learning . . . 20
2.3.1 Self-paced Learning (SPL) . . . 21
2.3.2 Self-paced Learning with Diversity (SPLD) . . . 22

iii METHODOLOGY 24

3 METHODOLOGY 25
3.1 The Data Set . . . 25
3.2 Sample difficulty . . . 25
3.2.1 Logistic loss and sample difficulty . . . 25
3.2.2 Difficult vs. easy samples in practice . . . 27
3.3 Persistence of easy vs. difficult categorization through training . . . 36
3.3.1 Persistence within each training run (within each image set) . . . 36
3.3.2 Persistence across image databases . . . 38
3.4 Curriculum Learning variants explored . . . 41
3.4.1 p-SPL . . . 42
3.4.2 p-SPLD . . . 43
3.5 Implementation Details . . . 43
3.5.1 SPL and p-SPL Implementation . . . 44
3.5.2 SPLD and p-SPLD Implementation . . . 44
3.5.3 ConvNet architecture . . . 45
3.5.4 Training parameters . . . 47
3.6 Experiment Design . . . 47
E1 Experiment Set I: Default Network with Standard Training . . . 48

EII Experiment Set II: SPL Training 49
EIII Experiment Set III: SPLD Training 50
3.7 Data analysis 51
3.7.1 Metrics 51
3.7.2 Hypotheses 51
3.8 Delimitations 52
3.8.1 Network architectures 52
3.8.2 Training parameters: iterations, learning rate changes, λ, γ, m%, etc. 52
3.8.3 Data sets 52
3.8.4 Batch sizes 52

IV RESULTS AND DISCUSSION 53

4 RESULTS AND DISCUSSION 54
4.1 Diagram details 54
4.2 Experiment Results 55
EI Experiment Set I: Default Network 55
EII Experiment Set II: SPL Training 61
EIII Experiment Set III: SPLD Training 76
4.3 Summary and Discussion 87

V CONCLUSION 90

5 CONCLUSION 91
5.1 Summary of Findings 91
5.2 Future Work 93

BIBLIOGRAPHY 94

VI APPENDIX 98

A APPENDIX 99
A.1 Default ConvNet architecture 99
A.2 Quick Training ConvNet architecture 103

LIST OF FIGURES

Figure 1 *A neuron accepts inputs from other neurons via its dendrites. Any output signal travels through the axon to the synaptic terminals, where it is transmitted to other nearby neurons.* 7

Figure 2 *A figure of an uninhibited McCulloch-Pitts neuron, inspired by Minsky [14], shown with sum transfer function and threshold activation function.* 9

Figure 3 *McCullog-Pitts neuron: separation of input space for the OR function.* 10

Figure 4 *Perceptron diagram for a sample unit* j 11

Figure 5 *A 3-layer fully connected feedforward network, where* φ, ψ *and* ω *represent the activation functions of each layer, and weighted sum is used as the transfer function in all layers.* 13

Figure 6 *Pyramid structure of the layers in a typical image recognition network, where the resolution of the image is reduced from layer to layer by a concatenation of basic operations like convolutions and pooling.* . 15

Figure 7 *Network diagram for a ConvNet, exhibiting the typical features of sparse connectivity and shared weights.* 16

Figure 8 *Example of a* 2D *convolution operation.* 17

Figure 9 *Example of a* 2D *max pooling operation, inspired by [27]* 18

Figure 10 *Histogram of the output probabilities for the correct class labels after training epoch* 1*. An epoch is the unit of measure of how many times the training algorithm has processed all training samples, in the case of CIFAR-10 this equates to one run through all 50,000 training images.* 27

Figure 11 *Histogram of the output probabilities for the correct class labels after epoch* 70 28

Figure 13 *Top* 100 *easiest samples, in raster scan order, after the* 1*st epoch* 28

Figure 12 *Histogram of the output probabilities for the correct class labels after epoch* 140 29

Figure 14 *Top* 100 *most difficult samples, in raster scan order, after the* 1*st epoch* 30

Figure 15 *Top* 100 *easiest samples, in raster scan order, after* 70 *epochs* 31

Figure 16 *Top* 100 *most difficult samples, in raster scan order, after* 70 *epochs* 32

Figure 17 *Top 100 easiest samples, in raster scan order, after 140 epochs* . . . 33
Figure 18 *Top 100 most difficult samples, in raster scan order, after 140 epochs* . . . 34
Figure 19 *Top 100 "easy" and "difficult" class histograms at epochs 1, 70 and 140* . . . 35
Figure 20 *Set intersections between top and bottom sets for 4 different CIFAR-10 databases* . . . 38
Figure 21 *Set intersections between top and bottom 20% sets across 2 CIFAR-10 databases (image sets)* . . . 40
Figure 22 *Set intersections between top and bottom 20% sets across 3 CIFAR-10 databases* . . . 41
Figure 23 *Default network diagram.* . . . 46
Figure 24 *Average accuracy on test set from 5 runs using Default network.* . . . 55
Figure 25 *Average loss on test set from 5 runs using Default network* . . . 56
Figure 26 *Average accuracy on test set from 5 runs using Default network, with percentage random samples removed.* . . . 57
Figure 27 *Accuracy for different % of easiest samples removed* 58
Figure 28 *Accuracy for different % of most difficult samples removed* . . . 59
Figure 29 *Avg. accuracy vs. % sample removal plot for the random, most difficult first, and easiest first partial data experiments.* . . . 60
Figure 30 *Average accuracy on test set from 5 runs using SPL network.* . . . 61
Figure 31 *Convergence speed on test set from 5 runs using SPL network.* . . . 62
Figure 32 *Sample inclusion for 4 different start percentage points* . . . 62
Figure 33 *Average accuracy on test set from 5 runs using SPL-Inversed network.* . . . 64
Figure 34 *Convergence speed on test set from 5 runs using SPL-Inversed network.* . . . 65
Figure 35 *Sample inclusion for 4 different start percentage points* . . . 66
Figure 36 *Avg. accuracy for regular, p-SPL and p-SPL-Inversed networks.* . . . 68

Figure 37 *Boxplot for the accuracy measured from 20 runs on randomized databases with CIFAR10 images with 3 types of networks. The median is represented by the horizontal line inside the box. The borders of the box are set at the 25th and the 75th percentiles. The whiskers extend to the most extreme data point within* $1.5 * (75\% - 25\%)$ *data range* 69
Figure 38 *Average accuracy on test set from 5 runs using p-SPL network.* 70
Figure 39 *Convergence speed on test set from 5 runs using p-SPL network.* 71
Figure 40 *Sample inclusion for 4 different start percentage points for p-SPL network* 72
Figure 41 *Average accuracy on test set from 5 runs using p-SPL-Inversed network.* 73
Figure 42 *Convergence speed on test set from 5 runs using p-SPL-Inversed network.* 74
Figure 43 *Sample inclusion for 4 different start percentage points for p-SPL-Inversed network* 75
Figure 44 *Average accuracy on test set from 5 runs using SPLD network.* 76
Figure 45 *Convergence speed on test set from 5 runs using SPLD network.* 77
Figure 46 *Sample inclusion for 2 different start* λ *values for SPLD network* 77
Figure 47 *Average accuracy on test set from 5 runs using SPLD-Inversed network.* 78
Figure 48 *Convergence speed on test set from 5 runs using SPLD-Inversed network.* 79
Figure 49 *Sample inclusion for 4 different start percentage points for SPLD network* 80
Figure 50 *Average accuracy on test set from 5 runs using p-SPLD network.* 81
Figure 51 *Convergence speed on test set from 5 runs using p-SPLD network.* 82
Figure 52 *Sample inclusion for 4 different start percentage points for p-SPLD network* 83
Figure 53 *Average accuracy on test set from 5 runs using p-SPLD-Inversed network.* 84
Figure 54 *Convergence speed on test set from 5 runs using p-SPLD-Inversed network.* 85
Figure 55 *Sample inclusion for 4 different start percentage points on p-SPLD-Inversed network.* 86

Figure 56 *A box and whisker plot of all explored methods at iteration 70,000. The median is represented by the horizontal line inside the box. The borders of the box are set at the 25th and the 75th percentiles. The whiskers extend to the most extreme data point within* $1.5 * (75\% - 25\%)$ *data range* 88

LIST OF TABLES

Table 1 Training parameters for Default, SPL and SPLD networks on CIFAR-10 47
Table 2 Training parameters for Experiment Set II: p-SPL Quick Training 50
Table 3 Accuracy 56
Table 4 Random sample removal: effect on accuracy . 57
Table 5 Ranked Removal: Easy First 58
Table 6 Removal Difficult First 59
Table 7 Avg. accuracy results for partial data experiments. 60
Table 8 Result summary from best experiment for SPL 63
Table 9 Result summary for SPL-Inversed 66
Table 10 Autem usu id 67
Table 11 Result summary for best experiment from p-SPL network 72
Table 12 Result summary for p-SPL-Inversed 75
Table 13 Result summary for SPLD network 78
Table 14 Result summary for SPLD-Inversed network. . 80
Table 15 Result summary for p-SPLD 83
Table 16 Result summary for p-SPLD-Inversed 86
Table 17 Avg. accuracy and avg. loss result summary from best experiment results for all explored methods. 87

LISTINGS

Listing 1 CAFFE network definition for Default Network 99

Listing 2 CAFFE network definition for Experiment Set II: %-SPL Quick Training Network 103

Part I

INTRODUCTION

The following chapter provides the introduction to the project, its goals, and describes the motivation behind pursuing this line of research. It sets up the expectation for the precise problems that are addressed in this thesis paper. Finally, it gives a brief overview of how the paper is organized.

1 INTRODUCTION

The mind is not a vessel to be filled,
but a fire to be kindled.

— Plutarch (c. 46 - 120 AD)

The ability to observe and comprehend patterns, infer similarities and relationships between concepts, and remember previously acquired information shapes the core of learning in intelligent beings. For humans, this process does not occur randomly, but gradually, as we master simpler knowledge and behavior first, and use what we have acquired to progress further. This characteristic is especially pronounced in formal learning settings: the information to be taught is structured in a curriculum, where the learning objectives have been curated by an expert to create an often linear progression from easy to difficult topics. We intuitively understand why this approach is beneficial for human beings, and it has naturally inspired similar techniques in the field of machine learning. Formally classified under the umbrella of "curriculum learning" and "active learning", the techniques focus on presenting the learning material to the algorithm in an organized way, usually in an easy to difficult progression (curriculum learning), and further giving the algorithm latitude in selecting what action or information to process next based on its own previous experience (active learning). Thus, the algorithm acquires knowledge by being exposed to the training data in a structured fashion, and is no longer a passive recipient, but an active participant in its own learning process. The two techniques that incorporate both of these characteristics and will be explored in this report are SPL [2] and SPLD [3]. Previously, the effect of these techniques has been documented using Support Vector Machines (SVMs), perceptrons, and 3-layer neural networks. In this work, the techniques will be used to train a "deep" ConvNet (convolutional neural network) on a labeled image data set with stochastic gradient descent.

Recently, multi-layer neural networks trained with supervised learning techniques have risen to prominence by braking several important performance records in speech recognition and computer vision by a large margin. Specifically, deep learning networks marked the highest performance in the following challenges: ImageNet, German Traffic Signs, Handwriting, and several Kaggle competitions (Facial Expressions, Connectomics, Multimodal Learning, Merck Molecular Activity, Adzuna Salary Prediction, among others), and TIMIT Phoneme

Recognition. Prior to the resurgence of neural networks, speech recognition and image recognition used to employ varied machine learning methods based on custom, hand-crafted feature extraction. The strong performance of deep learning networks in both scenarios is remarkable, because it hints at the existence of universal algorithms nursed by evolution for learning complex hierarchical feature representations, which are likely suitable to train with a curriculum.

Deep learning has now become a hot topic in computer vision, and is the dominant method for the central tasks of object recognition, object detection and semantic segmentation. It is also being adopted for acoustic modeling in speech recognition, natural language processing, and temporal prediction. It is viewed as one potential stepping stone towards the realization of the highest ambition of AI, the development of intelligent agents, capable of human-comparable performance on a wide variety of tasks. Human reliance on technology is growing, and so is the demand for machines that can further automate, assist or entertain humans in their activities. Very high value is placed on algorithms that can accept, process and interpret visual and auditory signals with near-human accuracy.

1.1 MOTIVATION

Supervised learning using ConvNets is currently the state of the art technique in image recognition tasks [4][5][6]. Regardless of the advances in raw computational power and the exploit of GPUs in deep learning implementations, the training of ConvNets is still an expensive and lengthy process. Any methods that can speed up or improve training are highly sought after.

Different variations of gradient descent [7][8], activation functions [9], regularization techniques [10] and sparse coding have provided important insights in how to improve training for these networks. While these methods concern themselves with the learning algorithms utilized by the network, another viable option is to explore how the presentation of the training data affects the quality of learning. In the usual supervised training setup, the samples are presented randomly to the algorithm without regard for any particular sample characteristics. The obvious alternative is to take the sample properties into account during training, and devise a particular order of presentation. The main idea behind this concept is to mimic how humans learn by mastering easier concepts first, before being introduced to more complex topics, so that the difficulty is systematically increased as learning progresses. In supervised learning context, this translates to "curating" the quality (and quantity) of samples presented to the learning algorithm at each iteration according to a set of pre-defined rules, so that samples are processed in order of increasing difficulty as training progresses. This idea was introduced under the concept

of "Curriculum Learning" by Bengio, Louradour, Collobert, *et al.* [11]. Curriculum learning has inspired derivative methods like self-paced learning (SPL) [2] and self-paced learning with diversity (SPLD) [3], which are discussed in detail in Chapter 3. These techniques have been studied in several contexts, but their potential influence on the training of deep ConvNets has not been explored. This project's main goal is to investigate how SPL and SPLD applied on a supervised learning problem using a ConvNet with large image data sets affects the performance of such networks.

1.2 PROBLEMS ADDRESSED IN THE THESIS

In neural networks, the model parameters are learned in an iterative fashion using a variation of stochastic gradient descent, by minimizing an objective ("loss") function. The loss function in ConvNets has a highly non-convex shape. Finding the global minimum of such a function is not computationally feasible. So, the algorithm focuses instead on finding one of many possible local minima. Naturally, some local minima are better than others, and there is no guarantee which one the algorithm will choose. Based on the observations in [2], self-paced learning (SPL) seems to act as a regularization term in the objective function, and empirically yields better (and faster) results in finding a better local minimum. In [3], the authors suggest that the convergence speed may be due to the fact that the examples explored first are "more informative". By intuition, the advantages of SPL and SPLD should translate well when applied within the context of ConvNets.

This project integrates SPL [2] and SPLD [3] techniques in the training process of Deep Convolutional Neural Networks. The training samples are selectively ordered based on difficulty as determined by the network's objective function. The experiments are designed to provide insight into whether SPL and SPLD have a positive effect on convergence speed and attained accuracy. Additionally, it investigates how excluding certain samples from the training set altogether influences the outcome of learning.

Specifically, the following aspects of ConvNet training using curated sample presentation are explored:

- Investigate how SPL affects convergence speed and accuracy
- Investigate how SPLD affects convergence speed and accuracy
- Investigate how reversing the curriculum, so that samples are presented in order of decreasing difficulty, influences SPL training

- Investigate how reversing the curriculum, so that samples are presented in order of decreasing difficulty, influences SPLD training
- Investigate how several warm-up iterations during which all samples are always included affect convergence speed and accuracy in SPL and SPLD
- Investigate how removing a certain percentage of samples from the training set based on difficulty criteria affects convergence speed and accuracy
- Explore the "quality" of training samples – are some samples better, more informative than others? Can irrelevant samples be filtered our early in the training process and excluded from training altogether without a notable effect on accuracy?

1.3 ORGANIZATION OF THE REPORT

This Master's Thesis project report is organized as follows:

CHAPTER 1 provides an introduction, details the motivation for the project, the problems addressed, and a brief overview of findings.

CHAPTER 2 is focused on the theory and background behind the concepts explored in this thesis in the areas of neural networks and curriculum learning.

CHAPTER 3 explains in detail the method used to implement and perform the experiments. It details the data set used, the structure and parameters of the ConvNet, the parameters used for the alternative SPL and SPLD implementations, and how the experiments are set up.

CHAPTER 4 contains the results of the experiments, and a detailed discussion of the findings.

CHAPTER 5 concludes the project report by summarizing the results and suggesting further topics of research that can built upon this work.

APPENDIX A contains the CAFFE network configuration parameters that, for brevity, were not included in Chapter 3, but may still be of interest to the reader.

Part II

THEORY AND BACKGROUND

The following chapter provides a broad overview of the history of feed forward neural networks, and their use in supervised learning tasks. It traces some of the main developments in the field that have shaped neural networks in their contemporary form. The chapter also devotes special care to explore the nature of convolutional neural networks in particular. Finally, it outlines current research trends in curriculum based learning.

2

THEORY AND BACKGROUND

2.1 A BRIEF HISTORY OF FEED FORWARD NEURAL NETWORKS

The main computational unit in neural networks was inspired from a model of a particular brain cell found in biological organisms – the neuron. The neuron is the core computational unit of the brain, and in oversimplified terms it is capable of processing input signals from other neurons, and making a decision on whether or not to produce an output based on these signals. A neuron diagram is shown on Figure 1.

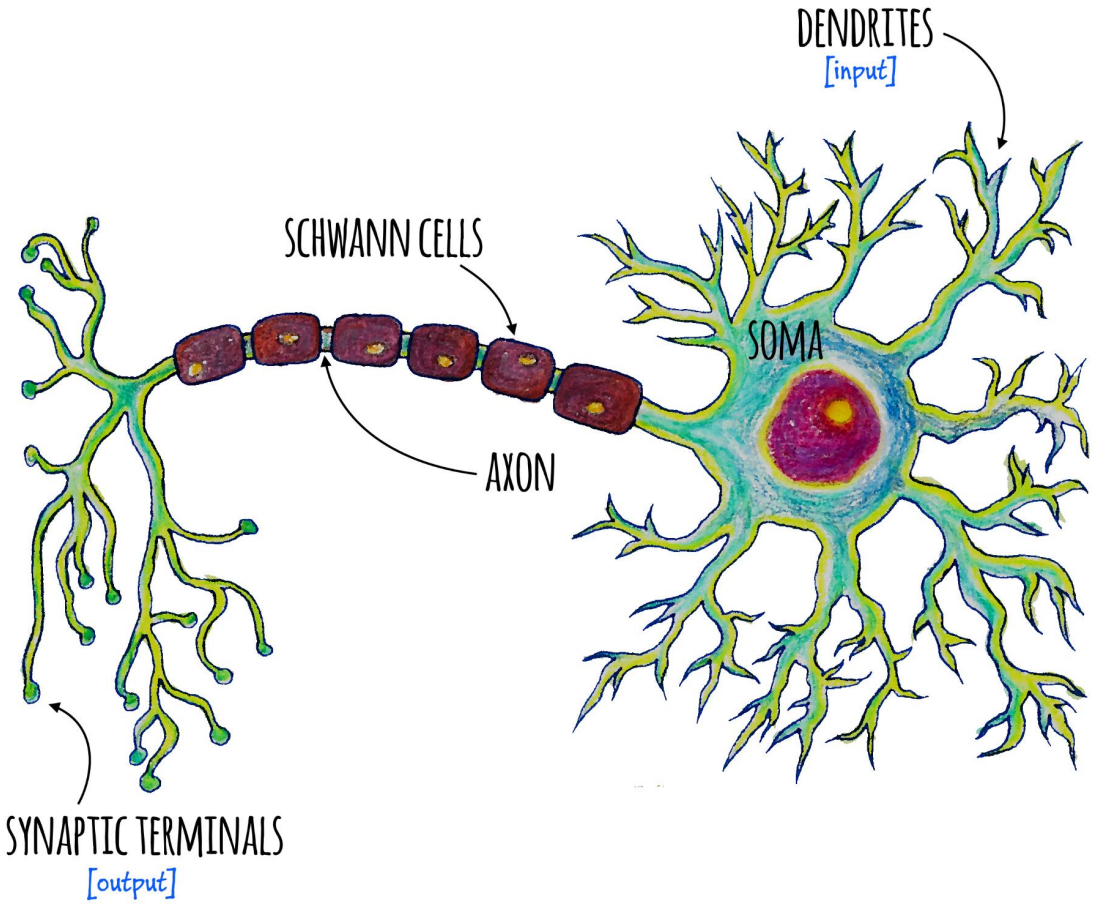


Figure 1: *A neuron accepts inputs from other neurons via its dendrites. Any output signal travels through the axon to the synaptic terminals, where it is transmitted to other nearby neurons.*

The average human brain has 100 billion neurons, each wired up to 10,000 other neurons in functionally related networks. Neurons pass signals to each other via trillions of synaptic connections. The behavior of the network is determined by its structure, and various connection strengths. Information transfer in the brain is an electrochemical process. Neuron output is continuous (usually measured as firing frequency, e.g. number of action potentials per second). The neuron acts like a voltage-to-frequency converter, by translating membrane potential into output frequency [12]. As higher voltage inputs come to the neuron, it fires in higher frequency, but the magnitude of the output is the same. The dendrites (inputs) to a neuron can be excitatory or inhibitory. The neuron fires or does not fire based on the summation value of its excitatory and inhibitory inputs. If the value is greater than the neuron's threshold, the signal propagates through the axon and is transmitted to the inputs of other neurons through the axon (synaptic) terminals.

The first mathematical model of a neuron as a computational unit was devised by neurophysiologist Warren McCulloch and Walter Pitts, a mathematician, in their paper "A logical calculus of the ideas immanent in nervous activity" [13]. In this paper, they model a simple neural network with electrical circuits. The neuron model has two inputs and a single output. In effect, this mimics two dendrites, a cell body and an axon. The inputs were binary, there were no weights, and the neuron produced a binary output. A diagram of the McCulloch-Pitts neuron is shown on Figure 2. A McCulloch-Pitts neuron operates in the following way: it receives d inputs, $x_1, x_2, ...x_d$ through d excitatory or inhibitory edges, where $d > 0$. It also receives between 0 and m inhibitory inputs. If at least one of the inhibitory inputs is 1, the entire neuron is "inhibited" and the output is 0 (absolute inhibition). Otherwise, the d excitatory inputs are summed up, and the result compared with the threshold θ. If the result $\geqslant \theta$, the neuron outputs 1, and 0 otherwise:

$$f(y) = \begin{cases} 1 & \text{if } y \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\text{where } y = \sum_{i=1}^{d} x_i \tag{2}$$

The function in equation 1, which determines how the neuron fires based on its inputs, is known as the activation function, while the function in equation 2 that processes the inputs before submitting to the activation function is known as the transfer function. The McCulloch-Pitts unit uses a simple sum of inputs as the transfer function, and a step-wise function at threshold θ as the activation function. By connecting several of these units in different configurations, Mc-

Culloch and Pitts were able to simulate logic gates, (implement any logical function), which proves that neurons as computational units can be theoretically used as basic components for Von-Neumann computer systems.

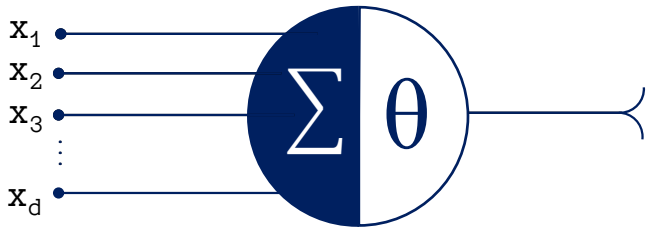


Figure 2: *A figure of an uninhibited McCulloch-Pitts neuron, inspired by Minsky [14], shown with sum transfer function and threshold activation function.*

It has been further shown that these neurons can synthesize any logical function of any number of arguments. Also, even though they use absolute inhibition, it has been proven that networks of McCulloch-Pitts units are equivalent to networks with relative inhibition. With relative inhibition, the edges are weighted with a negative factor, and in effect they raise the firing threshold when 1 is transmitted through the negative edge. Using relative weighted inhibition only simplifies the network topology, because less units are required if the signals can be weighted. The McCulloch-Pitts neuron as a computational unit has an important geometric interpretation. It separates the input space in two, using a hyperplane. It produces a distinct output (1) for points located in one of the half-spaces, and a different output (0) for points in the other half. An example is shown on Figure 3 for the OR function.

The next influential work into the concept of neurons was by Donald Hebb, in "The Organization of Behavior" [15]. Hebb's major contribution was the famous observation that "neurons that fire together, wire together", which is now known as Hebb's rule. This means that the strength of connections between neurons is proportional to how often they are used. The Hebb's rule is the first algorithm used to train neural networks. Associative networks like Hopfield networks are trained using Hebb's rule.

The introduction of weights came through the work of the American physiologist Frank Rosenblatt. In 1958, he described a computational unit known as "perceptron" [16]. This resulted in the first single layer perceptron built in hardware, the Mark I Perceptron in 1960. A single layer perceptron had a binary output, and could classify in-

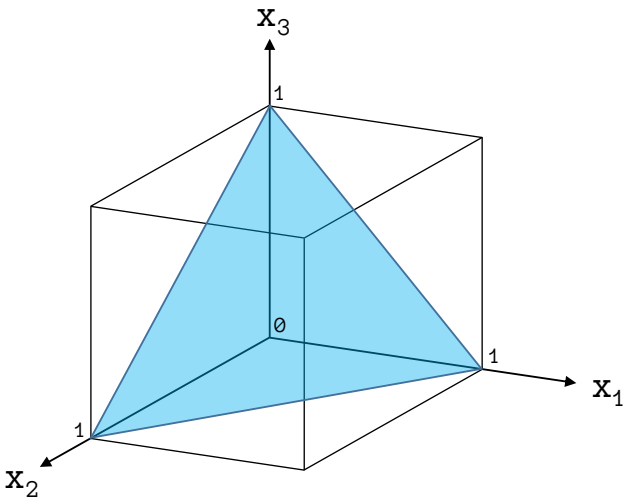


Figure 3: *McCullog-Pitts neuron: separation of input space for the OR function.*

puts in two output classes. A diagram of the perceptron is shown on Figure 4.

The main addition over the McCulloch-Pitts neuron were weighted connections. They effectively change the transfer function from a simple sum (McCulloch-Pitts) to a weighted sum. The transfer function remains the same. Effectively, the perceptron computes a weighted sum of the inputs, compares the result to a threshold, and passes one of two possible values as a result as shown in equations 3 and 4:

$$f(y) = \begin{cases} 1 & \text{if } y \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\text{where } y = \sum_{i=1}^{d} w_i x_i \tag{4}$$

The inputs can come from other perceptrons, or another class of computing units. Rosenblatt randomly (stochastically) interconnected the neurons and used trial and error to change the weights and facilitate learning. The geometric interpretation of Rosenblatt's perceptron is the same as for the McCulloch-Pitts neuron.

At this point it is clear that learning in these networks occurs in the weights. The problem is how to adjust the values of the weights. Selfridge was the first to elaborate the existence of local optima in weight-space, and introduce the concept of "hill-climbing" to reach such optima [17]. He observed that the weights can be adjusted by

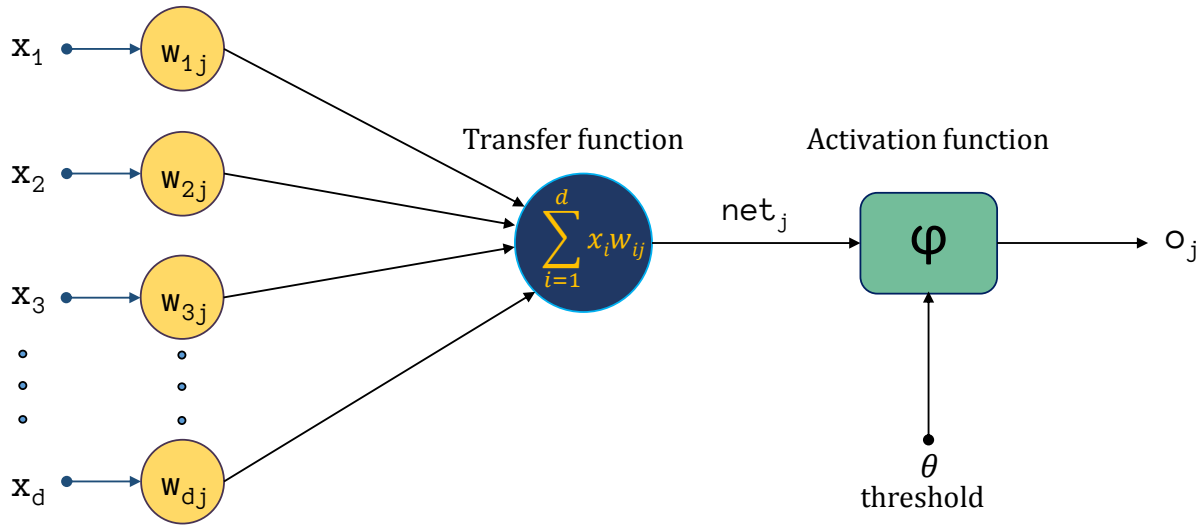


Figure 4: *Perceptron diagram for a sample unit* j

randomly choosing direction vectors, and taking small steps in the direction of the vector if the performance improves. If not, another random vector is chosen. This is the precursor to the classical gradient descent method used to train these networks today.

In 1959, Bernard Widrow and Marcian Hoff developed "adaptive learning models" they called "ADALINE" (Adaptive Linear Neuron) and "MADALINE" (Many ADALINEs) [18][19]. MADALINE was constructed from a composition of ADALINES, to form a 3-layer perceptron network. Widrow and Hoff trained the ADALINE networks using an algorithm that minimized the mean square error of the training set. The goal of the algorithm is to find the optimal set of weights $\mathbf{w}$ that minimize the error between target and inputs vectors.

Assume we have a training dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes the i^{th} training sample and $\mathbf{y}_i \in \mathbb{R}^j$ represents its target vector. Assume that $\mathbf{W}$ is a $d \times j$ matrix containing the weight vectors. Then, the difference between the desired and actual output can be expressed by the function shown in equation (5).

$$E(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}_i - \mathbf{W}^\top \mathbf{x}_i)^2 \tag{5}$$

This algorithm became known as Least Mean Squares (LMS). It was shortly after discovered that the LMS algorithm minimizes the error by following the path of steepest gradient descent. No longer a trial and error method, it provides a mathematical formulation to finding an answer (local or global optimum) to minimizing the training error. LMS has been used extensively in a variety of optimization applications.

It has to be noted that the perceptron also contains an additional single constant bias input (usually set to 1), which allows the decision boundary to be shifted by an offset value as necessary to fit the training data. It is incorporated as an entry in the $\mathbf{x}$ vector, and has its own set of weights in $\mathbf{W}$.

In 1969, Marvin Minsky and Seymour Papert published "Perceptrons" [20]. This book studied the essential features from Rosenblatt's model, and made important conclusions about the computational capabilities of perceptrons. The authors pointed out that a single layer network of perceptrons couldn't even learn the XOR Boolean function, because it can only solve linearly separable problems. The book demonstrated that a network of $N-1$ layers is needed in order to solve a N-separable problem. Since there was no algorithm in existence that could train multi-layered networks, this book is often credited as one of the reasons for the "AI winter", a period where funding for research into neural networks rapidly declined, and neural networks went "out of favor" with the general research community.

While the shortcomings of the single layer perceptron were apparent, it is proven that networks with a minimum of 3 layers are capable of approximating any computable function, which means that they can serve as universal approximators. The algorithm that made learning weights across multiple layers possible was backpropagation.

The backpropagation algorithm was developed by Paul Werbos [21]. It expands upon the Widrow-Huff LMS algorithm. The weights are once more adjusted based on the difference between the actual output and the known desired output. Thus, the error function is defined in weight space, usually incorporating the number of misclassified samples. Minimization of the function maximizes the performance of the network. The weights are adjusted backward through the network in the opposite direction of the computed gradient, by a fraction of the gradient value. This fraction is called the learning rate, and it controls how "fast" the network is learning, or how fast the algorithm performs the descent to reach the local minima. Normally the training process starts with a larger learning rate value, and decreases it after a number of iterations as the descent approaches the local minima. The weight update process is performed starting at the output layer, and going back layer-wise through each hidden layer until the input layer is reached. To make this possible, networks train by backpropagation have to use a differentiable activation function.

The "chain rule" for differentiating compositions of functions is used to facilitate this process: the partial derivative of the error with respect to last layer's weights is calculated and passed down to the previous layer to update the weights. Then the partial derivatives are computed for the previous layer and so on, until the weights of the input layer are reached. To be able to achieve non-linear mapping between input and output, the activation function itself

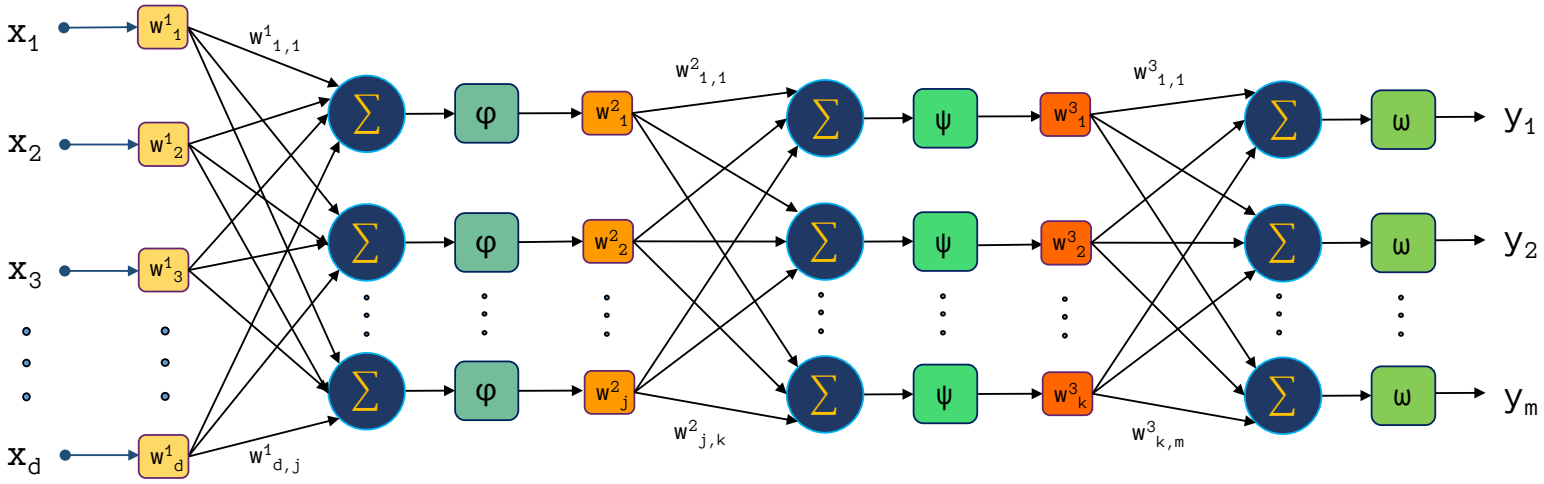


Figure 5: *A 3-layer fully connected feedforward network, where φ, ψ and ω represent the activation functions of each layer, and weighted sum is used as the transfer function in all layers.*

must be non-linear. Functions that are often used are the sigmoid (6), hyperbolictangent (7), and recently rectifiedlinear (ReLU) (8), which does not have a derivative at 0, but this limitation is easy to overcome in actual implementations. ReLU is preferred in modern networks for reasons we will discuss in section 2.2.2.4.

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{6}$$

$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{7}$$

$$f(x) = \max(0, x) \tag{8}$$

The major advancements outlined above were critical for the development of feed forward neural networks in the form they are presently known. To summarize, neural networks are designed to learn how to approximate a particular function. They can map from any d-dimensional input vector $\mathbf{x}$ to any m-dimensional output vector $\mathbf{y}$. Each node in the network is a composite in the total network

function, and may implement its own arbitrary function between inputs and outputs. A sample multi-layer network is shown on Figure 5. Its final function is given in Equation (9).

$$\mathbf{y} = f(\mathbf{x}) = \omega(\mathbf{W}_3^T \psi(\mathbf{W}_2^T \varphi(\mathbf{W}_1^T \mathbf{x}))) \tag{9}$$

This function is highly non-linear, because it represents a combination of non-linear activation functions (across several network layers), and can serve as an universal approximator.

The behavior of the network is defined by the following elements:

- The topology of the network (number of layers and neuron connectivity)
- The type of transfer and activation functions
- The learning algorithm used to adjust the weights

2.2 CONVOLUTIONAL NEURAL NETWORKS

Despite being universal function approximators, fully connected forward networks do not perform well when dealing with visual information. One of the reasons is that computer vision deals with very high dimensional data, where local values are highly correlated and translation invariant. The amount of neurons required to map to the input pixels, and connect fully into several layers grows exponentially with the size of the input image. This makes training the network very challenging. Another issue with full connectivity is that it causes spatial ignorance in the network - the inability to recognize the same object in different parts of the visual field (image), because separate weights are responsible for the different regions. These constraints led to the development of a network architecture that does not use fully connected neurons, but a hierarchical structure of locally connected neurons that eventually transmit output to a fully connected (often the last) layer. ConvNets embody this architecture.

ConvNets were again inspired by biological models: the visual cortex. Huber and Wiesel's studies of the visual cortex of cats [22] were largely influential for the development of these networks. Huber and Wiesel define the notion of receptive fields of a retina cell: a localized area in the retina that can influence the firing of that particular cell. They recognized that the ventral pathway in the visual cortex has multiple stages, and that information from receptive fields is passed layer by layer to different components of the system responsible for each stage. Huber and Wiesel identified two types of receptive fields: simple and complex. Simple cells give responses that can be directly attributed to the arrangement of excitatory and inhibitory responses

in their receptive fields. Complex cells fire in more varied and intricate ways that seem to indicate sensitivity to temporal changes in the visual signal.

2.2.1 *Network topology*

ConvNets were introduced by Kunihiko Fukushima. He designed a network model based on computational units called cognitron and neocognitron, in a way that mimics the vision pathway outlined by Huber and Wiesel. The network relied on pattern recognition in small receptive fields and convolution operators [23][24]. These networks exhibit a pyramidal architecture consisting of several planes (layers), where the resolution of the image is reduced from plane to plane by a given factor as shown on Figure 6. The important feature to note is that the input of several lower level pixels is "reduced" to a single output when fed to a single upper level neuron. This is the basis of the convolutional and sub-sampling operations essential in ConvNets.

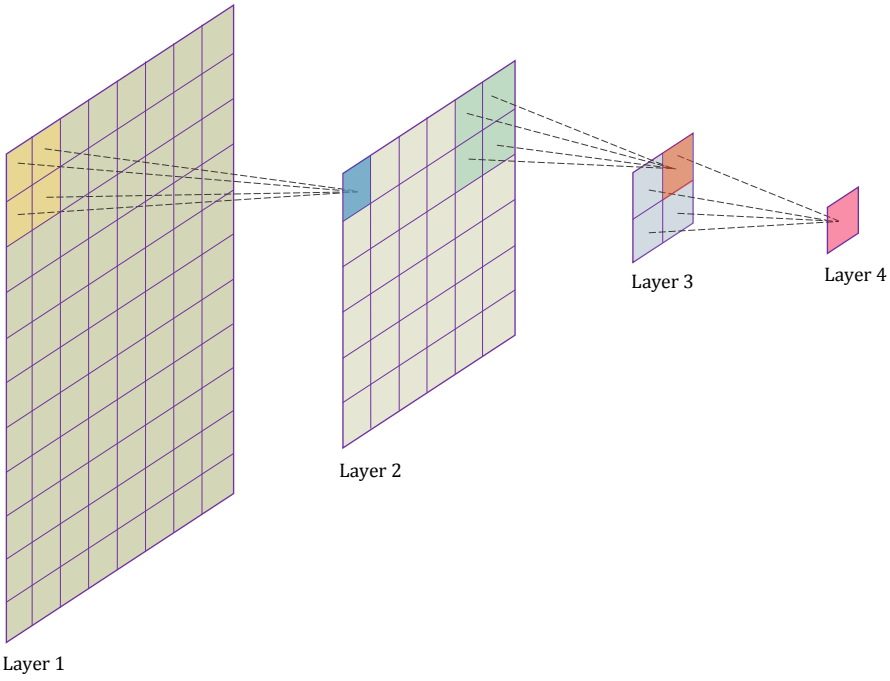


Figure 6: *Pyramid structure of the layers in a typical image recognition network, where the resolution of the image is reduced from layer to layer by a concatenation of basic operations like convolutions and pooling.*

Each neuron is connected to several neurons from the lower plane, which form this neuron's receptive field. Receptive fields do not overlap. These local connections capture local dependencies and features, which can be applied anywhere in the image. To further this effect, weights are shared across neurons with different receptive fields. Units in a layer share the same set of weights. A ConvNet diagram with a receptive field of size 3 is shown on Figure 7.

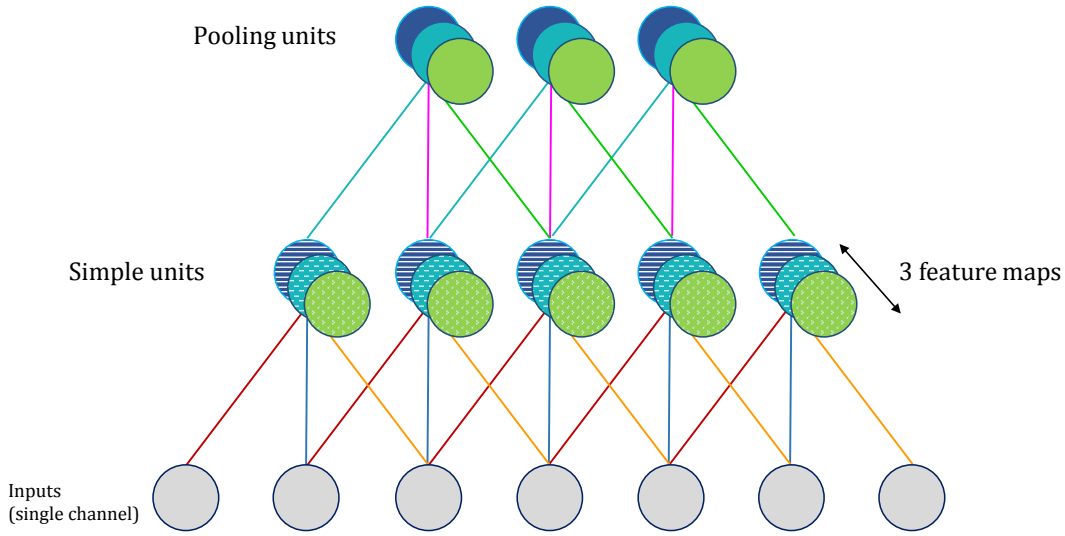


Figure 7: *Network diagram for a ConvNet, exhibiting the typical features of sparse connectivity and shared weights.*

2.2.2 *Network layers*

In addition to the particular sparse connectivity discussed above, ConvNets utilize several essential types of specialized layers.

2.2.2.1 *Convolutional layers*

Convolution as a mathematical term represents applying a function repeatedly over the output of another function. In the context of image processing, this represents applying a filter effect over the image. During this process, the value of a central pixel is determined by

adding the weighted values of all its neighbors as shown in equation 10 and on Figure 8.

$$p_{k,l} = \sum_{i=1}^{w} \sum_{j=1}^{h} A_{k+i-\lfloor \frac{w}{2} \rfloor, l+j-\lfloor \frac{h}{2} \rfloor} F_{i,j} \tag{10}$$

Here, A is the image patch matrix, and F is the filter (kernel) matrix of size $w \times h$.

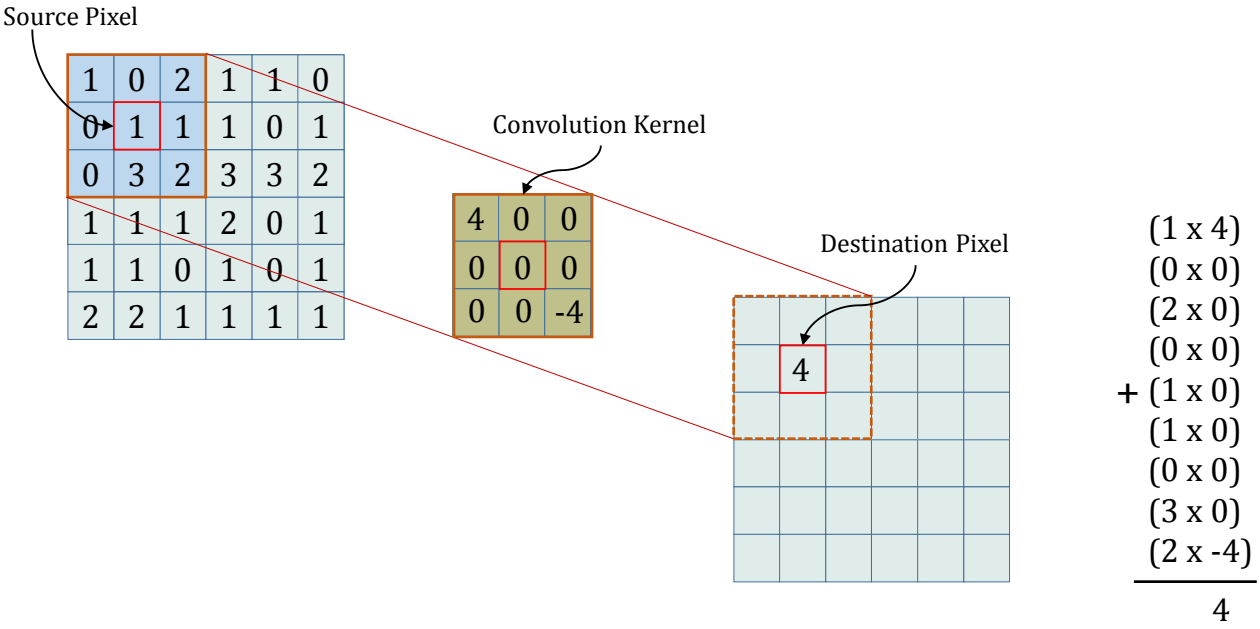


Figure 8: *Example of a* 2D *convolution operation.*

Common filter effects are: sharpen, blur, intensify, etc. The filter can be applied over the image at certain strides, or offsets. The larger the value of the stride, the bigger the size reduction (compression) of the original image. If the stride is 1, then the resulting image is of the same size as the original.

One convolutional layer may apply several filters, or feature maps. The network on Figure 7 has 3 feature maps. In essence, the filter is represented by a set of weights connected to a small patch of the original image, and it produces a single output. The resulting network structure mimics a series of overlapping receptive fields, which produces a series of "filter" outputs. Since all receptive fields share

the same weights, we only have to compute the weight updates for a single instance of the filter during backpropagation.

2.2.2.2 *Pooling (sub-sampling) layers*

In the general case, pooling (or sub-sampling) represents reduction of the overall size of a signal. In the context of image processing, it refers to reducing the size of the image. In ConvNets dealing with image recognition, it is used to increase the invariance of the filters to geometric shifts of the input patterns. Pooling can be achieved by using the average, L1, L2, or max of given signal data in a local patch. In effect, it promotes dimensionality reduction and smoothing. LeNet-5 [25] uses max pooling. The matrix of filter outputs is split into small non-overlapping grids (patches), and the maximum (or average) value of each grid becomes the output, as shown on Figure 9. Applying max pooling layers between convolutional layers increases spatial and feature abstractness [26][25].

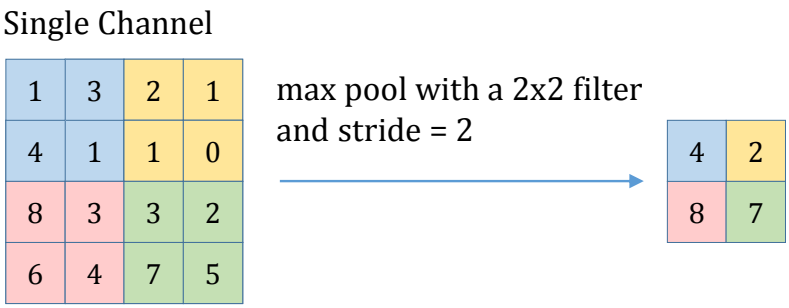


Figure 9: *Example of a* 2D *max pooling operation, inspired by [27]*

2.2.2.3 *Normalization layers*

The intention of this type of layer is to perform a type of "lateral inhibition". It is useful in combination with ReLU units because of their unbound activations. It allows for the detection of features with a spike in response value by normalizing over local input values. At the same time, it inhibits regions with uniformly large response values. The normalization can be performed across or within channels.

WITHIN CHANNEL: Here, the local regions extend spatially, but are each in its own channel (i.e., they have shape $1 \times s \times s$, where the normalization region is of size $s \times s$.). Each input value $p_{x,y}$ is updated to $p_{x',y'}$ as shown in equation (11):

$$p'_{x,y} = \frac{p_{x,y}}{\left(1 + \frac{\alpha}{s^2} \sum_{i=x-\lfloor \frac{s}{2} \rfloor}^{s} \sum_{j=y-\lfloor \frac{s}{2} \rfloor}^{s} p_{i,j}^2\right)^{\beta}} \tag{11}$$

where the sum is taken over the region centered at $p_{x,y}$ (with zero padding added if necessary), and α and β are variables that can be set to fine-tune the normalization.

ACROSS CHANNELS: In this mode, the local regions used for normalization extend across nearby channels, but have no spatial extent. Their dimension is $s \times 1 \times 1$, where s is a variable specifying the size of the local normalization region. Each input value is updated as shown in equation (11).

$$p'_{x,y} = \frac{p_{x,y}^{(s)}}{\left(1 + \frac{\alpha}{s} \sum_{c=1}^{s} (p_{i,j}^{(c)})^2\right)^{\beta}} \tag{12}$$

The output dimensionality of this layer is equal to the input dimensionality.

2.2.2.4 *Rectified Linear Unit (ReLU) layers*

A ReLU layer consists of neurons that use a rectifier activation function shown in (8). The rectifier function adds non-linearity to the network, and is much more computationally cost-efficient compared to other activation functions like sigmoid and hyperbolic tangent. Using this activation function greatly accelerates the convergence of the stochastic gradient descent algorithm [28][4] compared to the sigmoid and hyperbolic tangent functions. It also combats the problem of vanishing gradients.

2.2.3 *Training algorithm (Learning the filters)*

Fukushima et. al. used unsupervised learning techniques in [24]. The introduction of gradient based learning through backpropagation in these networks came through the work of several independent research groups, most notably Rumelhart, Hinton, and Williams [7] and Le Cun [26]. The training techniques for ConvNets have been further refined and used successfully on many image recognition tasks [29][25].

The architecture outlined above remains central to the ConvNets used for image recognition tasks today. The defining characteristics of these networks remain sparse connectivity, shared weights, pooling layers followed by sub-sampling, and supervised training with stochastic gradient descent. The algorithm's goal is to optimize a selected objective (cost) function (often `softmax loss`, which is discussed in detail in 3.2.1).

2.3 CURRICULUM-BASED LEARNING

Convex learning is invariant to the order of sample presentation, but both human learning and ConvNets are not: the order in which things are learned is significant. Usually, in any formal academic setting, the information to be taught to students is carefully structured in a curriculum, so that "easier" concepts are introduced first, and further knowledge is systematically acquired by mastering concepts with increasing difficulty. The same tactic has been successfully applied in animal studies, where the technique was called shaping [30][31]. The idea of training neural networks with a curriculum can be traced back to Elman [32]. The experiment involved learning a simple grammar with a recurrent network. Elman noted that networks who start "small" (with limited working memory) succeed in the task better than "adultlike" networks who are exposed to the full grammar at their maximum learning capacity. The idea of shaping was explored by Krueger and Dayan [33]: they used an abstract neural network model on a hierarchical working memory task. The authors found that the network trained with shaping acquired the task faster than a network with conventional training. Bengio, Louradour, Collobert, *et al.* formally introduced a technique called "curriculum-based learning" [11], where the "easy" samples are used for training first, and the rest of the examples are presented with increasing difficulty. The authors explored the effects of curriculum learning using SVMs, a perceptron, and a 3 layer neural network on a toy image dataset where the samples were categorized manually as "easy" or "difficult". Their experiments confirmed that curriculum-based learning affects positively the resulting trained models in terms of accuracy, convergence speed and ability to generalize.

Another technique inspired by human learning is active learning, introduced by Cohn, Ghahramani, and Jordan [34]. Here, the agent or algorithm "learner" still learns concepts in some relevant order, but is not treated as a passive information recipient, and can actively choose what action or information to process next based on previous experience. Thus, it is capable of positively affecting its learning objective by shaping its own curriculum. Even though active learning was initially explored with unlabeled datasets, and semantically intersects with reinforcement learning, the principle is universal and does not

depend on the specific learning task. Several heuristics on how the agent should select its curriculum have been explored:

- explore places with highest variance (highest potential to affect the model) [35]
- explore where there is little data available [36]
- explore places with highest uncertainty [37]
- explore where prediction errors are high [38]
- explore places with highest entropy [39]

As a natural progression, one may say that curriculum learning adds another heuristic: explore items of slightly higher difficulty compared to the set that was explored last. Curriculum-based learning and active learning have inspired techniques like self-paced learning (SPL) [2], and self-paced learning with diversity (SPLD) [3], where the learning algorithm selects by itself what to learn next, and focuses on items with increasing difficulty. These techniques have been successfully applied on various learning tasks within the context of SVM and latent SSVM [2][3].

2.3.1 *Self-paced Learning (SPL)*

When Bengio et. al. performed their experiment, the data sets used for training was manually curated into "easy" and "difficult" examples [11]. In practice, such datasets are not easy to come by. Kumar et. al. address this problem by introducing the SPL algorithm [2], which removes the need to label the training data with difficulty levels, and enables sample selection to occur during the training process in order of increasing difficulty. Difficulty is expressed in terms of the magnitude of the difference between the actual value of the objective function and expected value for a given training sample. The larger the difference, the more difficult the example is considered to be.

SPL: Given the training dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ denotes the i^{th} training sample and y_i represents its label, let $L(y_i, f(\mathbf{x}_i, \mathbf{w}))$ be the loss function which calculates the difference between the true label y_i and the estimated label $f(\mathbf{x}_i, \mathbf{w})$. To enable SPL, another term is added to the optimization problem - a binary variable v_i that indicates whether the i^{th} training sample is easy or not [2][3]. This allows the algorithm to define its own curriculum at each step by learning both the model parameters $\mathbf{w}$ and the binary vector $\mathbf{v} = [v_1, \ldots, v_n]$, as shown in equation (13):

$$(\mathbf{w}_{SPL}, \mathbf{v}_{SPL}) = \underset{\mathbf{w}\in\mathbb{R}^d, \mathbf{v}\in[0,1]^n}{\operatorname{argmin}} \left[L_{SPL}(\mathbf{w}, \mathbf{v}; K) \right] \tag{13}$$

where

$$L_{SPL}(\mathbf{w}, \mathbf{v}; K) = \sum_{i=1}^{n} v_i L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \frac{1}{K} \sum_{i=1}^{n} v_i \tag{14}$$

Solving this optimization problem globally is not computationally feasible, so it is approached as an iterative, 2-stage optimization, performed by alternatively solving subproblems (15) and (16):

$$\mathbf{v}_{t+1} = \begin{cases} 1 & \text{if } L(y_i, f(\mathbf{x}_i, \mathbf{w}_t)) < \frac{1}{K} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$\mathbf{w}_{t+1} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \left[\sum_{i=1}^{n} v_{t+1,i} L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \frac{1}{K} \sum_{i=1}^{n} v_{t+1,i} \right] \tag{16}$$

K is a parameter that determines the level of difficulty of the examples to be considered. If K is large, the learning step tends to consider "easy" samples with a small value of $L(\cdot, \cdot)$. The algorithm selects which examples are included at each step based on the value of the threshold $\frac{1}{K}$. If $L(\cdot, \cdot) < \frac{1}{K}$, then $v_i = 1$ and the sample is considered easy and included for training. Otherwise, $v_i = 0$ and the sample is not included. The value of K starts high at $t = 0$, and is decreased iteratively at each learning step as t increases, so that the algorithm starts out with a few easy examples, and gradually sees examples with increased difficulty until the entire data set is processed.

2.3.2 *Self-paced Learning with Diversity (SPLD)*

Jiang, Meng, Yu, *et al.* propose another advancement to the learning process, termed SPLD [3]. In SPLD, sample diversity is included as an additional criteria when selecting samples for the curriculum, so that the curriculum consists of diverse examples (in addition to certain difficulty level) at each iteration.

SPLD: To implement SPLD [3], another term is added to the optimization problem of (13), as shown in equation (17):

$$L_{SPLD}(\mathbf{w}, \mathbf{v}; \lambda, \gamma) = \sum_{i=1}^{n} v_i L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \lambda \sum_{i=1}^{n} v_i - \gamma \sum_{j=1}^{b} \|\mathbf{v}^{(j)}\|_2 \tag{17}$$

Here, $\lambda = 1/K$, and b is the number of similarity groups (clusters) that the training examples are partitioned in. The last term in (17) represents an $l_{2,1}$-norm [40], which allows for selection of diverse samples residing in different data "clusters". γ is a variable that can

be used to set importance, or weight, of the diversity term in the equation. Similarly to SPL training, this function is optimized iteratively, by solving the following subproblems:

$$\mathbf{v}_{t+1} = \begin{cases} 1 & \text{if } L(y_i, f(\mathbf{x}_i, \mathbf{w}_t)) < \lambda + \frac{\gamma}{\sqrt{j}+\sqrt{j-1}} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

$$\mathbf{w}_{t+1} = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left[\sum_{i=1}^{n} v_{t+1,i} L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \lambda \sum_{i=1}^{n} v_{t+1,i} - \gamma \sum_{j=1}^{b} \|\mathbf{v}_{t+1}^{(j)}\|_2 \right] \tag{19}$$

The algorithm selects samples in terms of both easiness and diversity based on the following rules [3]:

- Samples with $L(y_i, f(\mathbf{x}_i, \mathbf{w})) < \lambda$ will have $v_i = 1$ and will be selected in training. These represent the easy examples. This is the same as in equation (13).
- Samples with $L(y_i, f(\mathbf{x}_i, \mathbf{w})) > \lambda + \gamma$ will have $v_i = 0$ and will not be selected in training. These samples represent examples that are "too complex" to be considered at this step.
- The remaining samples will be selected for training if $L(y_i, f(\mathbf{x}_i, \mathbf{w})) < \lambda + \frac{\gamma}{\sqrt{j}+\sqrt{j-1}}$, where j is the sample's rank based on its $L(\cdot,\cdot)$ value within its cluster

As stated previously, the loss function in ConvNets usually has a highly non-convex shape with many local minima, so the order of sample presentation affects learning. In this report, SPL and SPLD principles are utilized to train a contemporary ConvNet on a large image data set.

Part III

METHODOLOGY

This chapter details the strategy used to fulfill the main objective of this report, which is to provide a comprehensive investigation of curriculum learning techniques in the context of ConvNet trained with labeled image data. It outlines the rationale behind the experiments, establishes some sample metrics needed to enable curriculum learning, lists the experiments that are performed and the hypotheses to be evaluated. It also describes two new subvariants of SPL and SPLD that arose as alternative implementations during the course of the project, and provides the technical implementation details of the project. Finally, it includes the project delimitations.

3 METHODOLOGY

3.1 THE DATA SET

The experiments were performed using the CIFAR-10 dataset [41]. The CIFAR-10 dataset contains $60,000$ 32×32 RGB color images. There are 10 classes: "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", and "truck", with $6,000$ images per class. The classes are mutually exclusive, i.e. there is no overlap between "automobiles" and "trucks". These class labels were used to form the diversity clusters needed for SPLD. The dataset is divided into five training batches and one test batch, of $10,000$ images each. The test batch contains $1,000$ randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.

Since the ordering of images as they are presented to the network matters for these experiments, 20 image databases were generated from the CIFFAR-10 data, with the image order randomized in each database. The experiments were performed on subsets of these databases as needed.

3.2 SAMPLE DIFFICULTY

The core feature to enable the experiments in this project is the ability to rank the training samples based on their level of presumed difficulty. As seen in sections 2.3.1 and 2.3.2, the difficulty level of a sample is proportional to a chosen distance metric between the output label and the actual classification label of said sample. This distance metric is often conveniently provided by the loss function utilized by the learning algorithm, since it already attempts to minimize the distance between the expected and actual sample labels. The ConvNet in our experiments uses the multinomial logistic loss function, or softmax regression, and we will use the difference between the true label (1) and actual label probability value to measure sample difficulty.

3.2.1 *Logistic loss and sample difficulty*

In its last fully connected layer, for each training sample $\mathbf{x}_i$ where $i \in [1 \dots n]$, our ConvNet produces an output vector $\mathbf{p}$ of size $1 \times K$, where K is the number of classes. This vector is passed through the

softmax function, which effectively squashes the values so that they form a valid probability distribution (equation (20)):

$$\sigma(p_j) = \frac{e^{p_j}}{\sum_{k=1}^{K} e^{p_k}} \qquad \text{for } j = 1, \ldots, K \tag{20}$$

This operation results in a vector $\mathbf{p}_i \in [0,1]^K$, which contains K predicted probabilities for sample $\mathbf{x}_i$'s classification, where:

$$\sum_{k=1}^{K} \mathbf{p}_{ik} = 1 \tag{21}$$

We also have n label vectors $\mathbf{l} \in \{0,1\}^K$, where each vector $\mathbf{l}_i = [0, \ldots, 1, \ldots, 0]$ contains a single 1 at position t, so that $\mathbf{l}_{it}$ represents the correct label index for sample $\mathbf{x}_i$ among the K classes. In information theory, the cross-entropy between two probability distributions r and q is given by equation (22):

$$\mathbb{H}(r, q) = -\sum_{i} r_i \log(q_i) \tag{22}$$

This principle can be used as an objective function, which effectively causes the network to minimize the cross-entropy between the estimated class probabilities $\mathbf{p}_i$ and the true distribution $\mathbf{l}_i$, contained by the label vector. This objective function is known as logistic loss:

$$L(\mathbf{p}_i, \mathbf{l}_i) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{l}_{ik} \log\left(\mathbf{p}_{i\mathbf{l}_{ik}}\right) = -\frac{1}{n} \sum_{i=1}^{n} \log\left(\mathbf{p}_{i\mathbf{l}_{it}}\right) \tag{23}$$

Therefore we can define the difficulty for a training sample x_i as the difference between its actual label and the predicted probability value from (20) for this label:

$$\mathcal{D}(\mathbf{x}_i) = 1 - \mathbf{p}_{i\mathbf{l}_{it}} \tag{24}$$

Higher values of $\mathcal{D}$ represent higher difficulty.

To be able to set the λ and γ parameters for SPL and SPLD, it is important to understand how the predicted probability values start out and change during the course of training. At the beginning of training, the true label predictions are fairly random, so most of the values are distributed under the 0.2 mark, as shown on Figure 10.

Mid-point in training, the values are noticeably more evenly distributed (Figure 11), and at the end of training (Figure 12), the predicted values are strongly shifted towards 1.0, which means that the

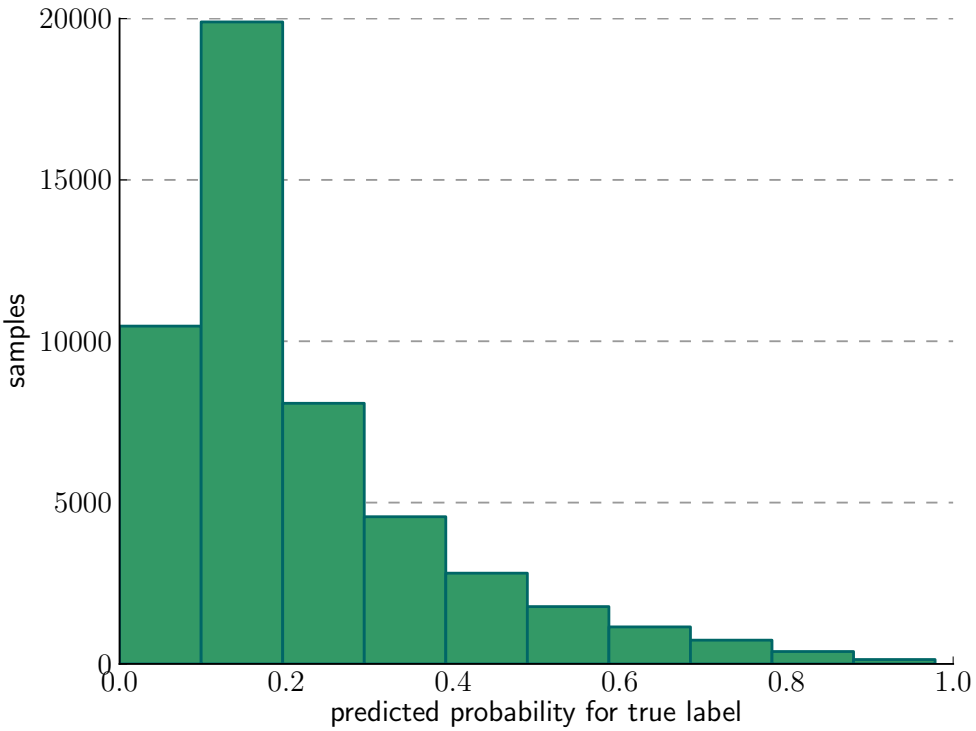


Figure 10: *Histogram of the output probabilities for the correct class labels after training epoch* 1. *An epoch is the unit of measure of how many times the training algorithm has processed all training samples, in the case of CIFAR-10 this equates to one run through all 50,000 training images.*

network is now able to predict the correct labels on the majority of the training set with a high degree of confidence.

The progression from low to high prediction confidence is fairly consistent across training runs. It can be used as a guideline to select suitable values for setting the difficulty parameters during SPL and SPLD training, and to select proper step values to facilitate reasonable progression of sample inclusion during training.

3.2.2 *Difficult vs. easy samples in practice*

Let us take a look at how using the softmax prediction values to form a notion of difficult and easy samples works in practice. The following figures show the top 100 easy and difficult samples as determined during the training of an SPLD ConvNet. The snapshots were again taken after the first, median and last epoch of training.

Figure 13 shows the easiest 100 samples after 1 epoch. After the first epoch, we can observe that the network is doing best when classifying airplanes, automobiles and ships (see histogram on Figure 19).

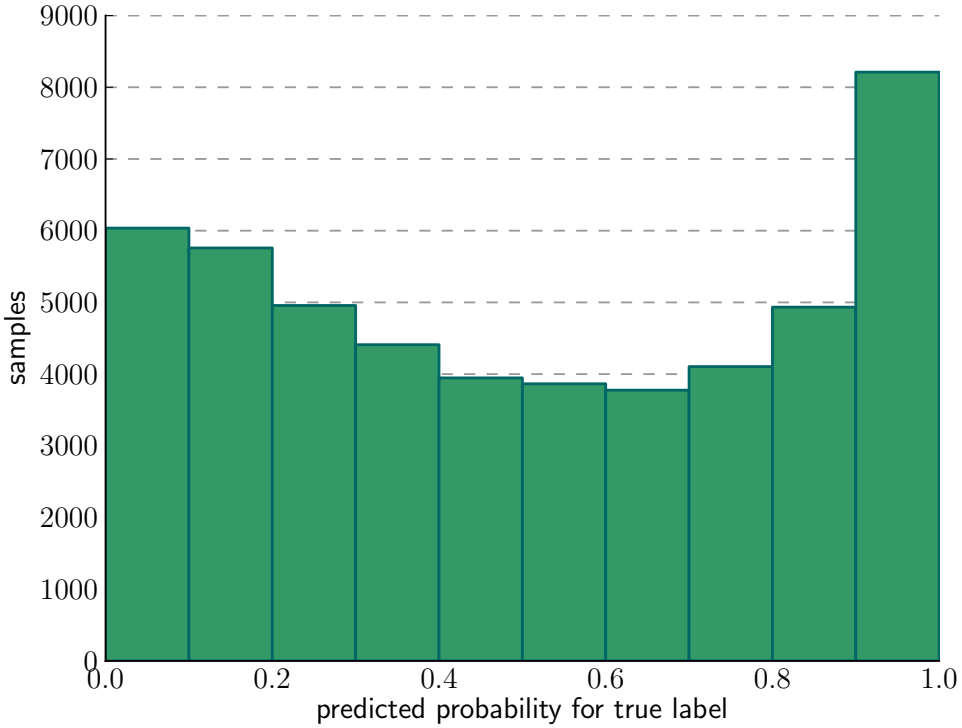


Figure 11: *Histogram of the output probabilities for the correct class labels after epoch 70*

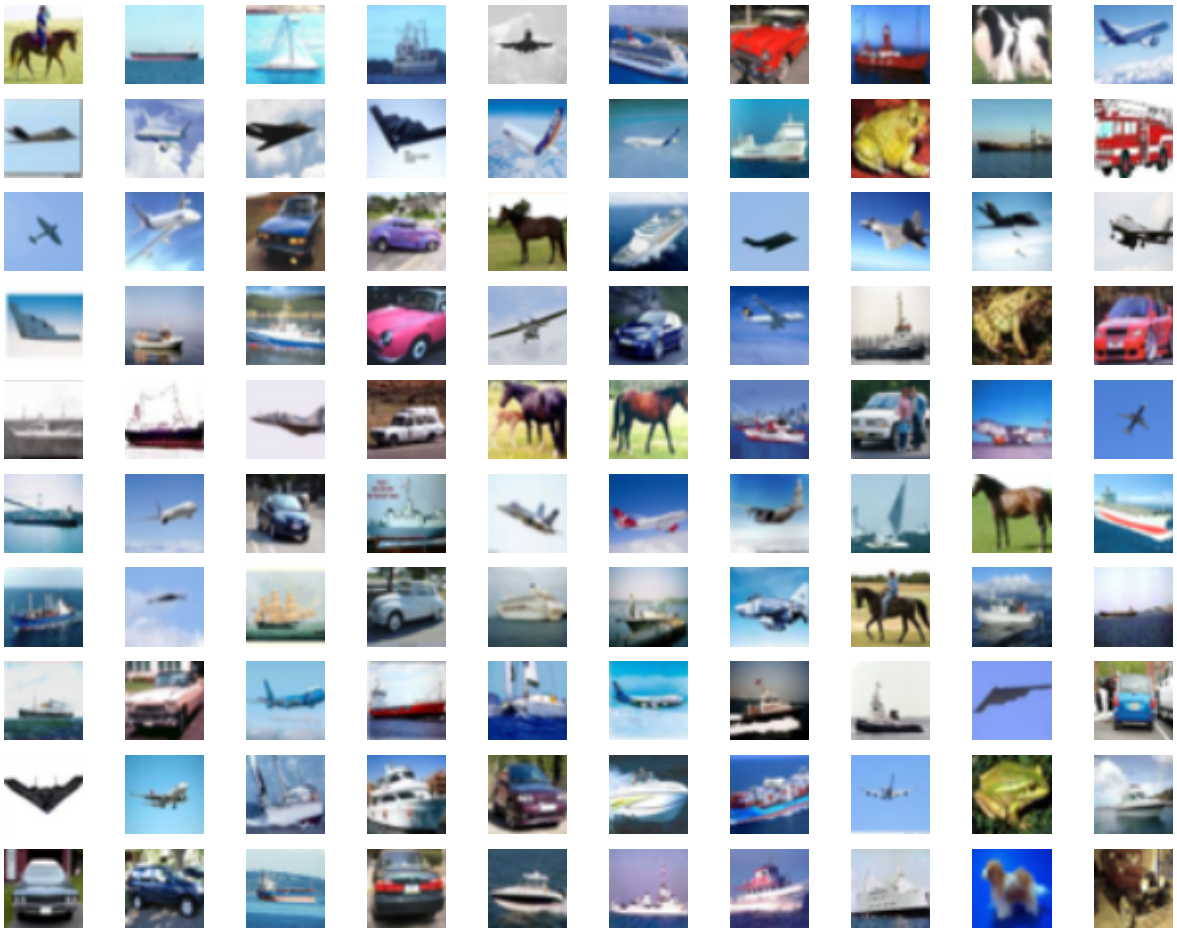


Figure 13: *Top 100 easiest samples, in raster scan order, after the 1st epoch*

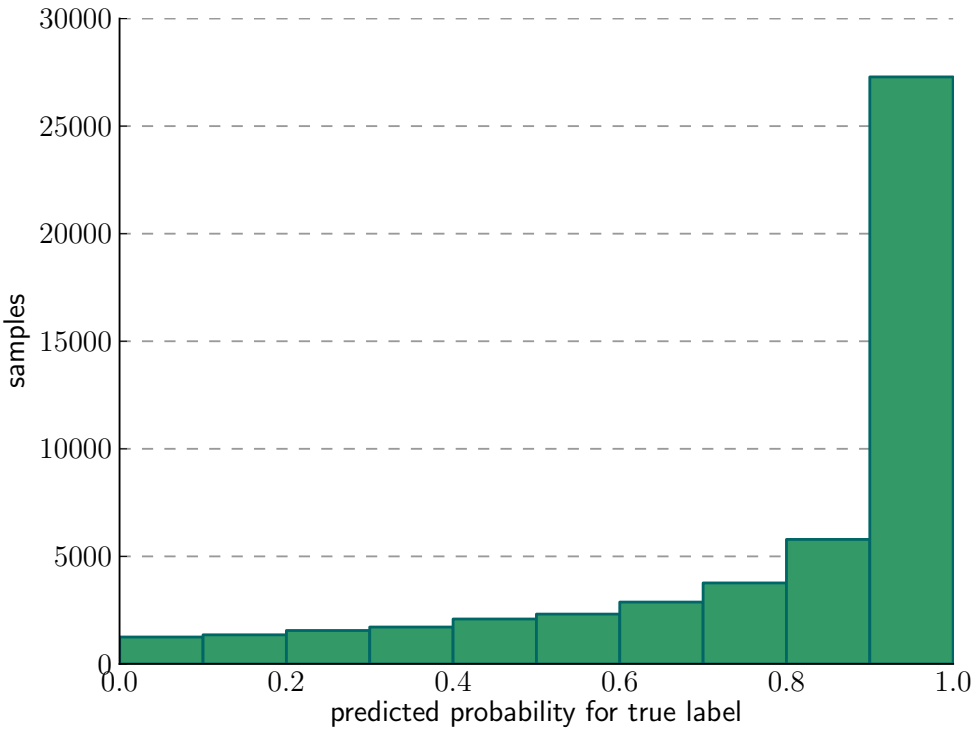


Figure 12: *Histogram of the output probabilities for the correct class labels after epoch* 140

Figure 14 shows what the network believed to be the most difficult 100 samples after 1 epoch. These samples are more or less random at this point, most likely a set close to the first 100 samples that the network encountered in epoch 1, at the very beginning of training. It is interesting to note that airplanes, automobiles and ships form the predominant set in difficult samples as well.

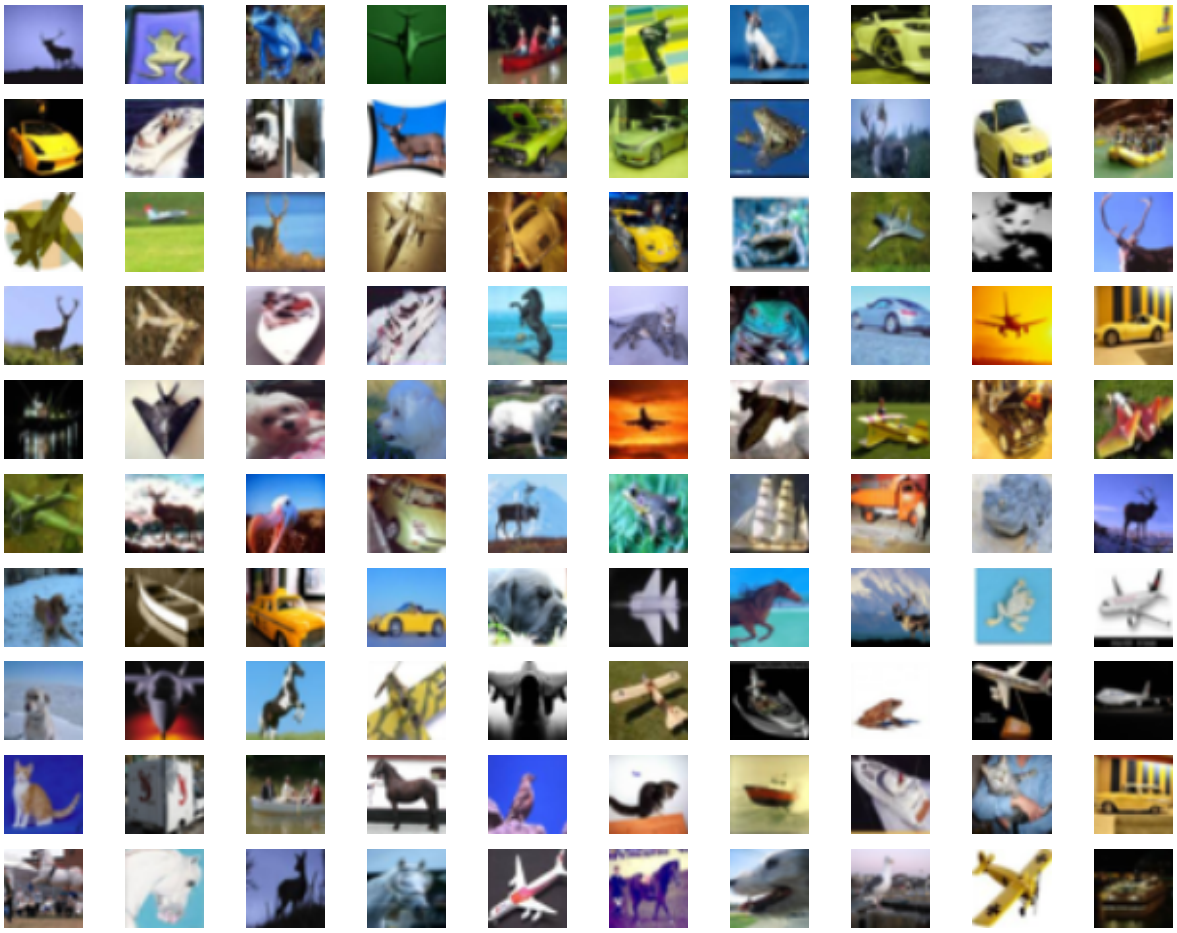


Figure 14: *Top* 100 *most difficult samples, in raster scan order, after the* 1*st epoch*

Figure 15 shows the easiest 100 samples at mid-point in training, after the 70th epoch. It appears that the network is shifting its opinion of "easy" from airplanes to cars. Figure 16 shows what the network believed to be the most difficult 100 samples after the 70th epoch. Here, the challenges of classifying some of the difficult images become more evident.

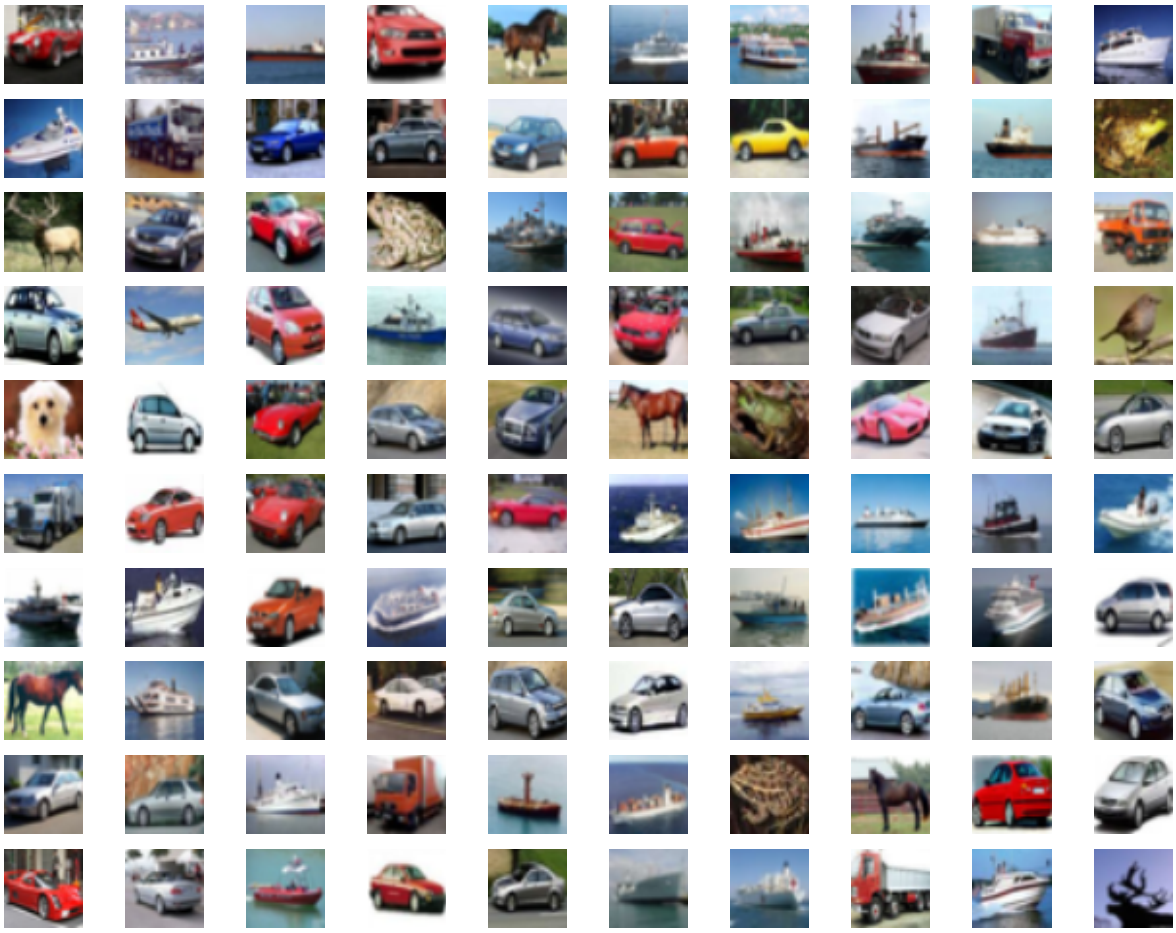


Figure 15: *Top* 100 *easiest samples, in raster scan order, after* 70 *epochs*

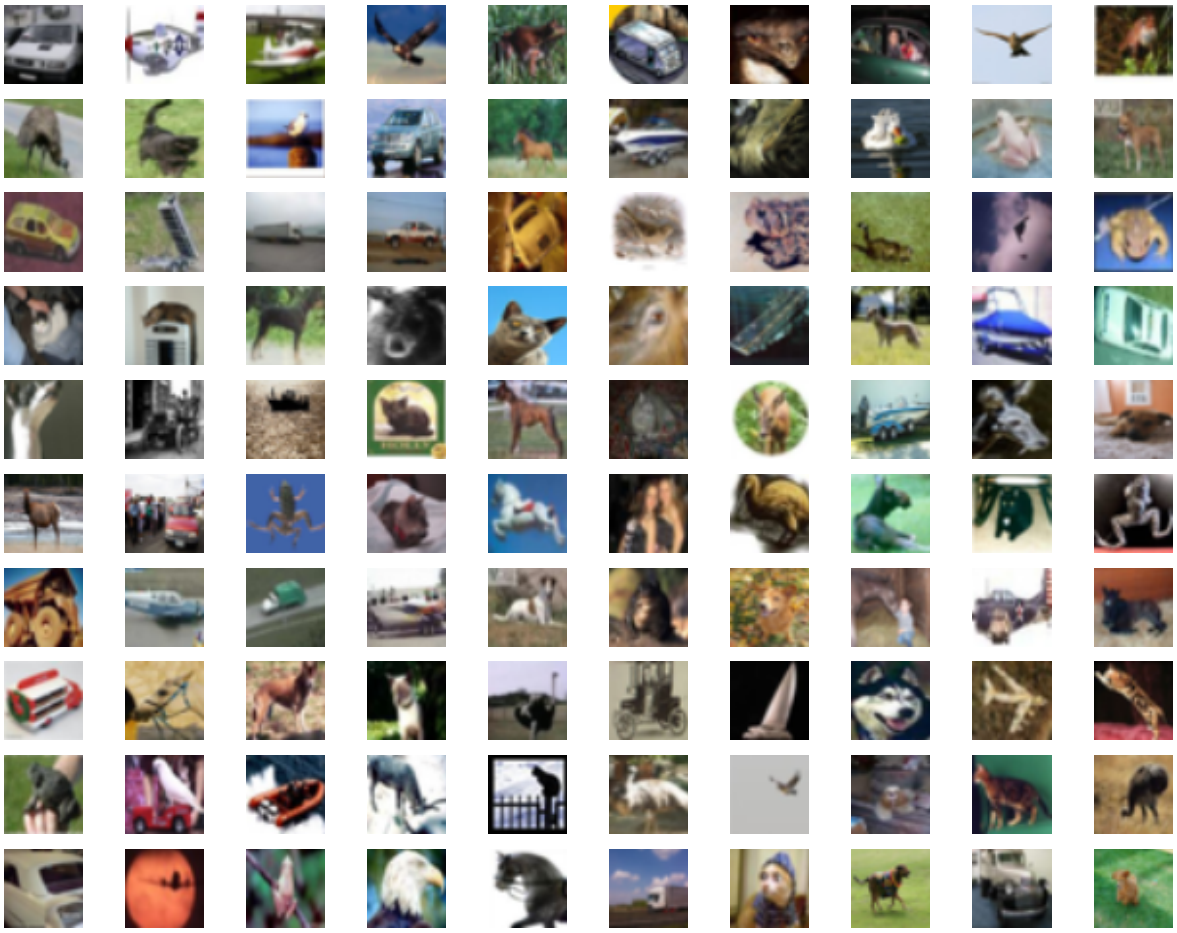


Figure 16: *Top* 100 *most difficult samples, in raster scan order, after* 70 *epochs*

At epoch 140 (Figures 17 and 18), the easy and difficult distributions are fairly consistent with the set from epoch 70. The easiest classes remain automobile and ship. The most difficult classes are bird and dog. Overall, the network has a very clear opinion of what (few) classes are easiest, while the difficulty mass is more evenly distributed across classes.

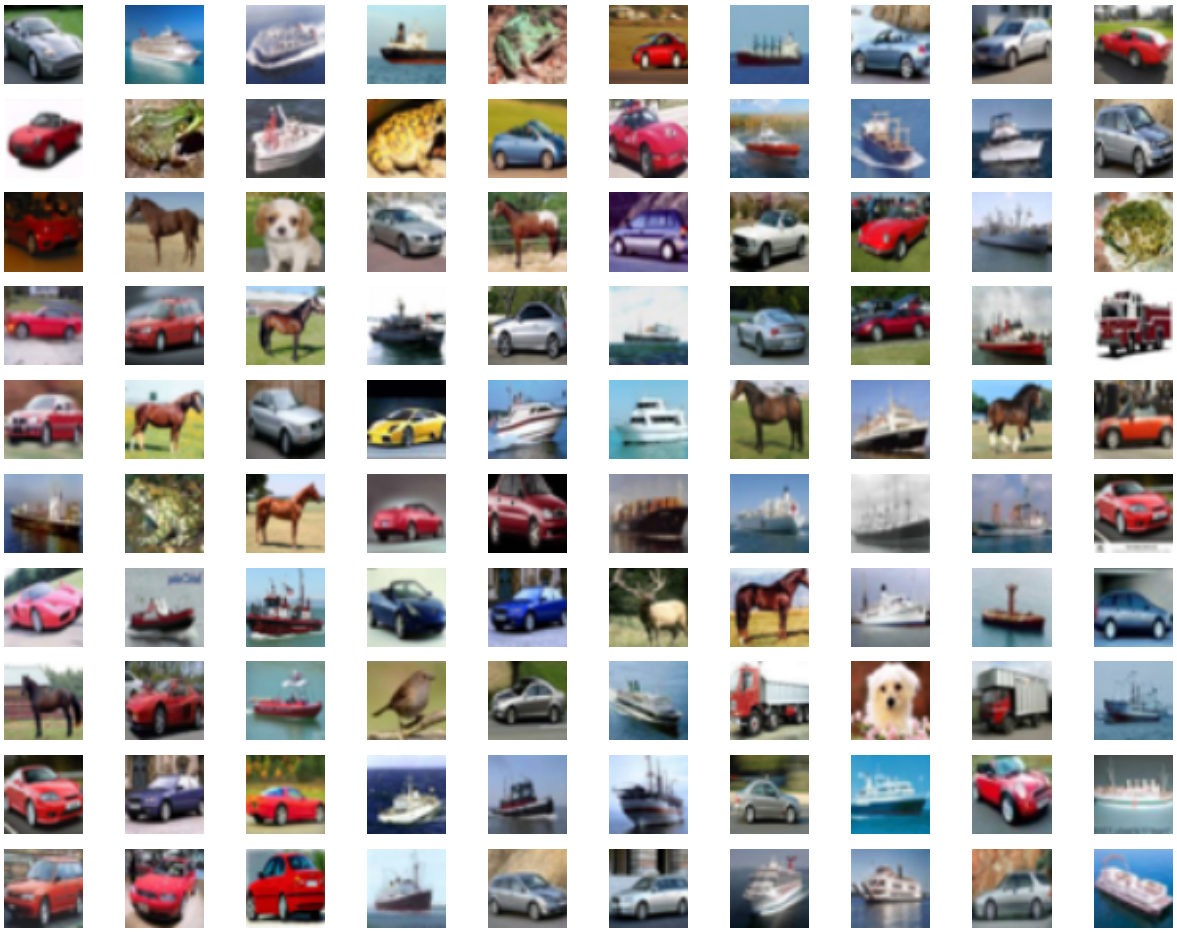


Figure 17: *Top* 100 *easiest samples, in raster scan order, after* 140 *epochs*

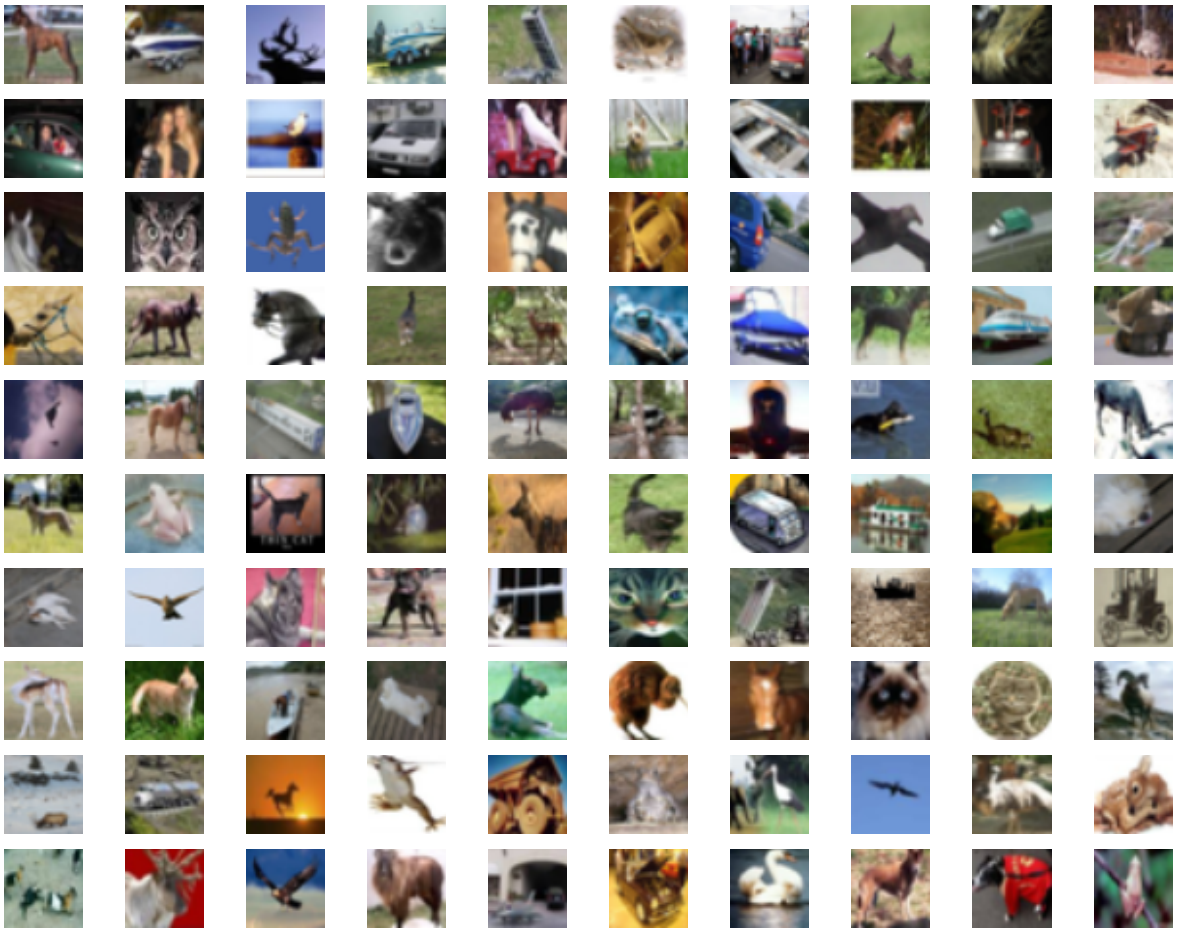
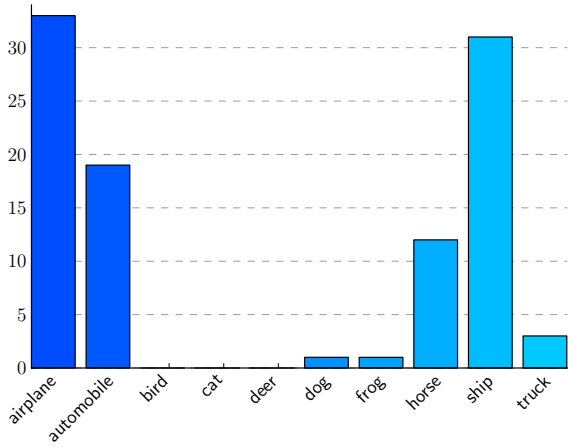


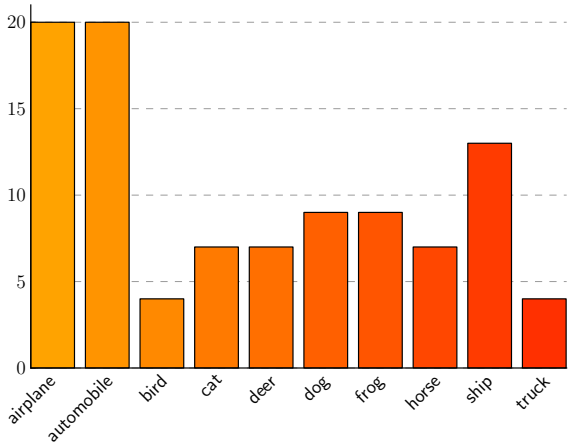
Figure 18: *Top* 100 *most difficult samples, in raster scan order, after* 140 *epochs*

Figure 19 shows class distribution histograms for the top 100 easiest and most difficult training sample sets taken at epochs 1, 70 and 140. Notably, the sets remain fairly consistent between the 70 and 140 epoch. Also, the sets of easy samples are more similar than the sets of difficult samples.

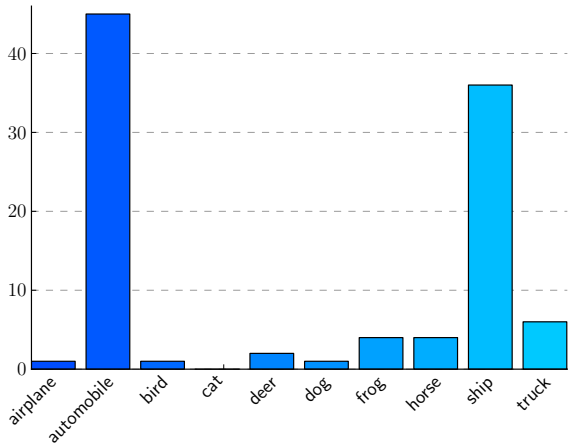
Figure 19: *Top 100 "easy" and "difficult" class histograms at epochs* 1, 70 *and* 140



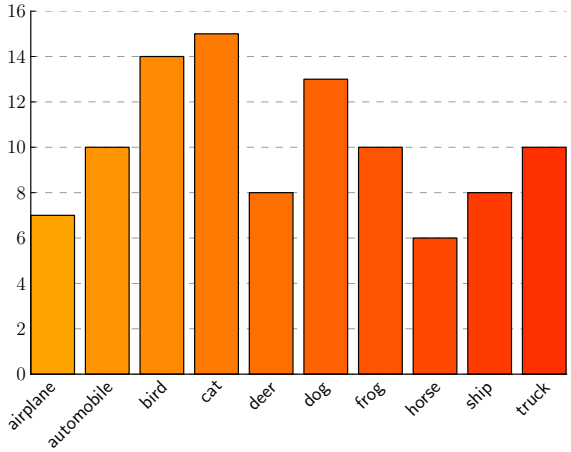
(a) Easy classes, epoch 1



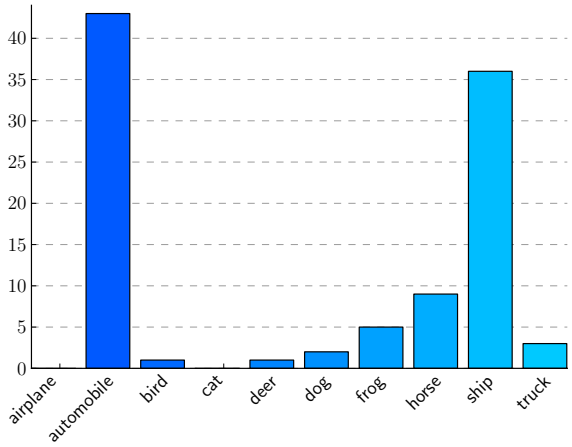
(b) Difficult classes, epoch 1



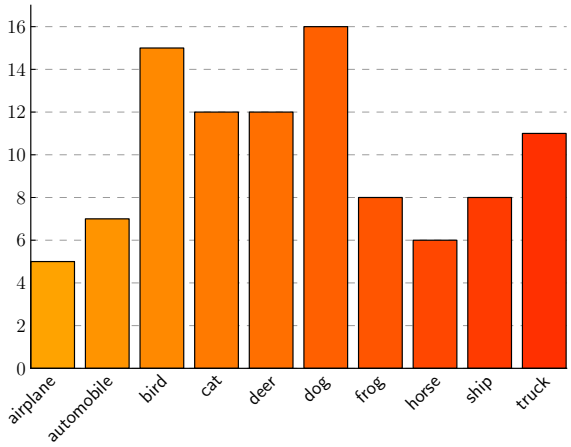
(c) Easy classes, epoch 70



(d) Difficult classes, epoch 70



(e) Easy classes, epoch 140



(f) Difficult classes, epoch 140

From human perspective, it appears that assigning difficulty level to the sample based on its loss value makes sense. The majority of the images in the most difficult set at the mid- and end-points of training exhibit some challenging characteristics:

- Vignetting
- Main subject with poor proportions with respect to the image size: either clipped, or occupying a very small area of the image
- Subject occluded, or displayed in an usual way, like animals wearing hats and/or clothing
- Low contrast images
- Cluttered backgrounds
- Images where the subject can be confused for one of the other possible classes, for example large dogs that look like horses, or birds that look like airplanes

On the other hand, the easy samples make sense as well: the network has easily mastered red and gray cars, ships, several very similar frog images with a distinctive body pattern. Most of these images are high contrast, on a clear background, and positioned auspiciously within the image.

3.3 PERSISTENCE OF EASY VS. DIFFICULT CATEGORIZATION THROUGH TRAINING

It is informative to investigate how the categorization of "easy" and "difficult" samples persists throughout training. For this purpose, the top (most difficult) and bottom (easiest) 20% of samples, sorted by difficulty level in descending order, were recorded every 10 epochs for a training run on 4 random CIFAR-10 image sets (using the databases described in section 3.1). For each of the 4 runs, this resulted in 14 sets of $10,000$ images for the top 20% sets, and 14 sets of $10,000$ images for the bottom 20% sets.

3.3.1 *Persistence within each training run (within each image set)*

Four types of intersections of these top and bottom sets were explored within each training run:

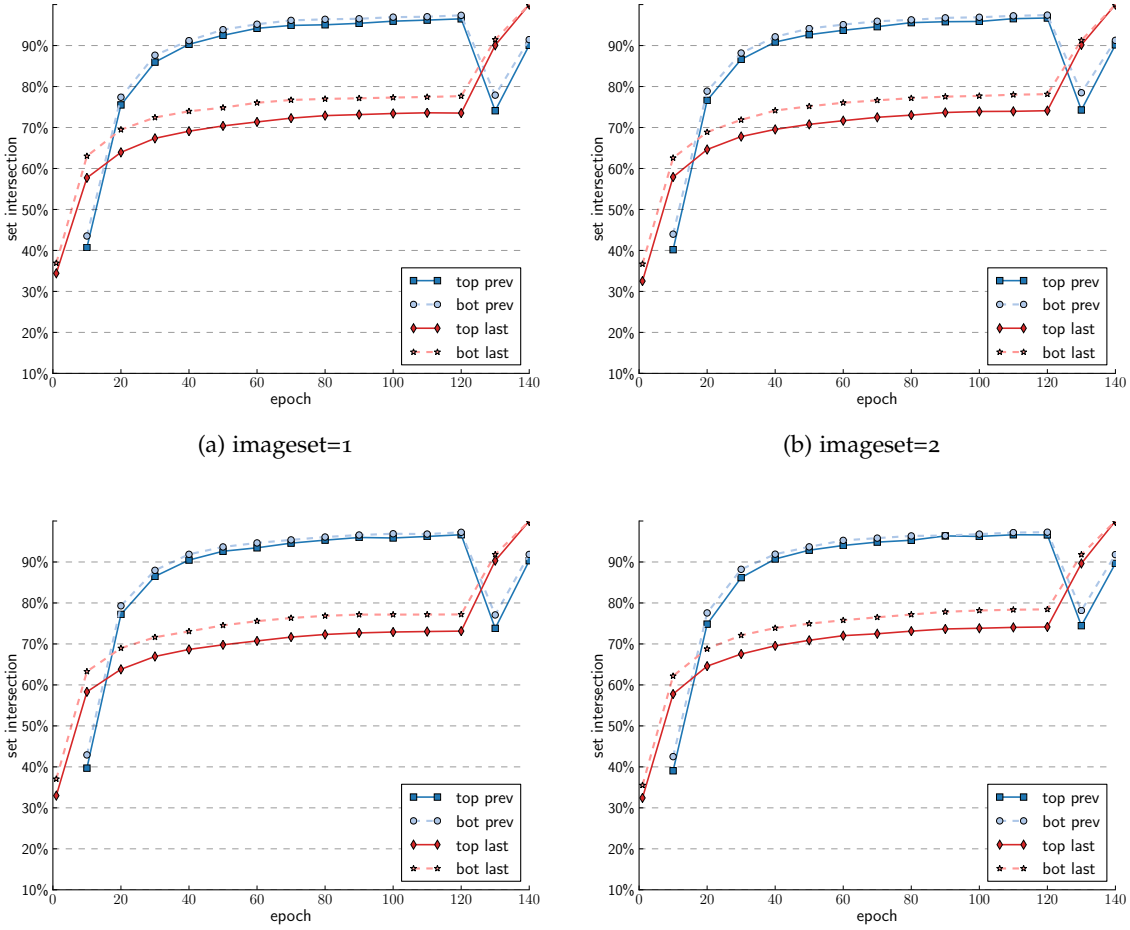
1. Intersection of the top set from epoch i with the top set from epoch $i - 10$.
2. Intersection of the bottom set from epoch i with the bottom set from epoch $i - 10$.

3. Intersection of the top set from epoch i with the set from epoch 140 (the last epoch of training)

4. Intersection of the bottom set from epoch i with the set from epoch 140 (the last epoch of training)

The results are shown on Figure 20. The intersection curves look very similar across the 4 image sets. It is apparent that both the top and bottom set intersections, taken 10 epochs apart, are highly homogeneous, with over 90% overlap during most of the training. The overlap of each set with the final set is over 70% after epoch 50. We can also observe that the bottom sets (easiest samples) retain higher similarity than the top sets (most difficult samples). Another interesting point is the change in the intersection curves that occurs at epoch 120. Epoch 120 is significant, because it corresponds to iteration $60,000$, when the learning rate changes for the first time during training (120 epochs $\times 50,000$ samples per epoch $= 60,000$ iterations $\times 100$ samples per iteration). Changing the learning rate causes a significant shuffle of the top and bottom sets in the following few epochs. This is likely related to the gradient of the loss function changing direction at this point.

Figure 20: *Set intersections between top and bottom sets for 4 different CIFAR-10 databases*



(a) imageset=1

(b) imageset=2

(c) imageset=3

(d) imageset=4

3.3.2 *Persistence across image databases*

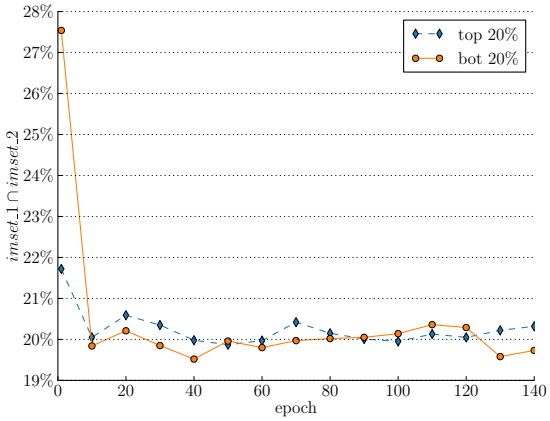
The next item to investigate was whether this high overlap between the top and bottom sets is sustained across the 4 different training runs across different databases. Here, the following intersections were explored:

1. Intersection between the top sets from each epoch across any 2 training runs (image databases)
2. Intersection between the bottom sets from each epoch across any 2 training runs (image databases)
3. Intersection between the top sets from each epoch across any 3 training runs (image databases)

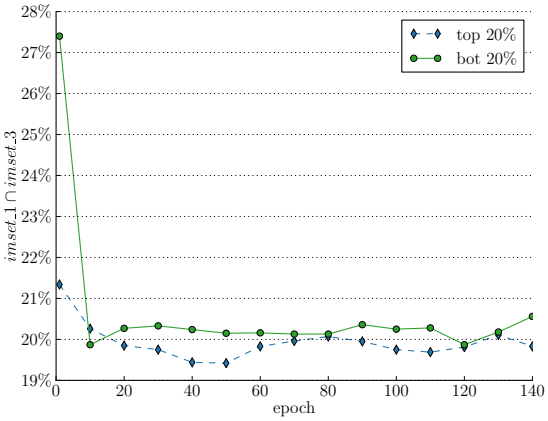
4. Intersection between the bottom sets from each epoch across any 3 training runs (image databases)

The results for items 1 and 2 above are shown on Figure 21, and the results for items 3 and 4 are shown on Figure 22. It appears that the set similarity hovers around 20% between the 2-database intersections, and drops sharply to $\approx 4\%$ when the sets are intersected across 3 image databases. This tells us that the formation of the top and bottom sets is specific to each image database (or training run), and that networks trained on different image sets have very different opinions on what constitutes the top and bottom sets.

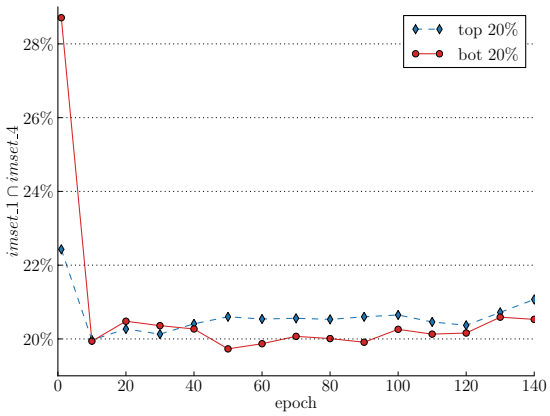
Figure 21: *Set intersections between top and bottom 20% sets across 2 CIFAR-10 databases (image sets)*



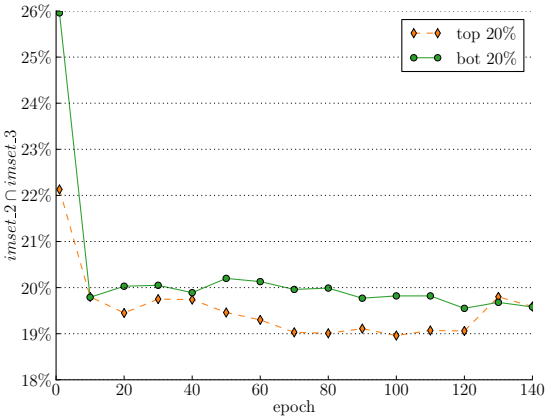
(a) imset_1 ∩ imset_2



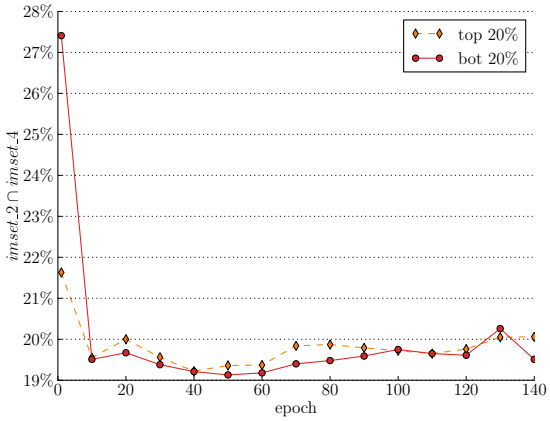
(b) imset_1 ∩ imset_3



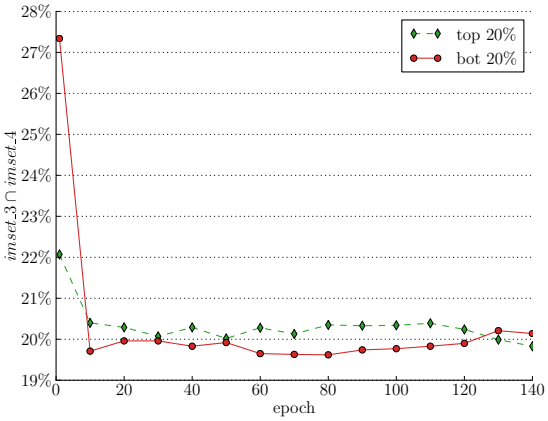
(c) imset_1 ∩ imset_4



(d) imset_2 ∩ imset_3

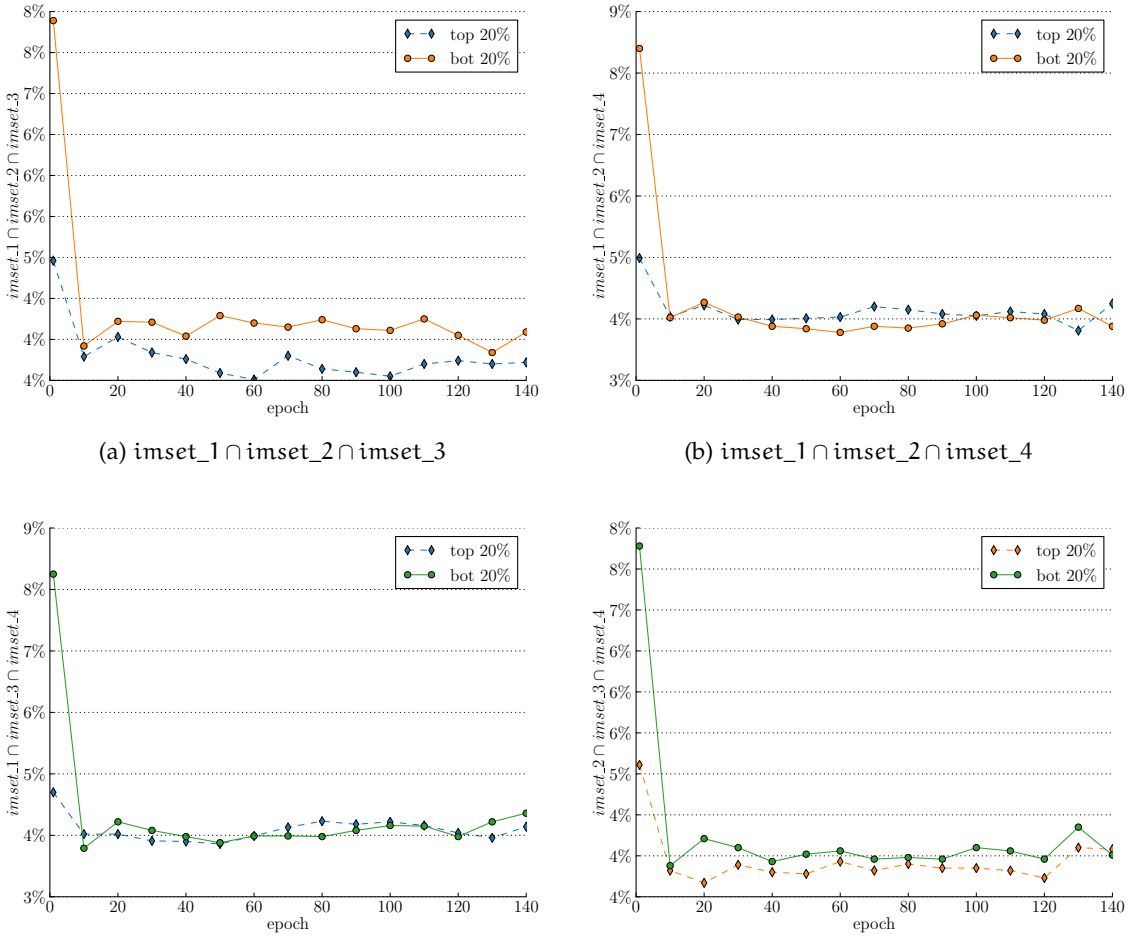


(e) imset_2 ∩ imset_4



(f) imset_3 ∩ imset_4

Figure 22: *Set intersections between top and bottom 20% sets across 3 CIFAR-10 databases*



(a) imset_1 ∩ imset_2 ∩ imset_3

(b) imset_1 ∩ imset_2 ∩ imset_4

(c) imset_1 ∩ imset_3 ∩ imset_4

(d) imset_2 ∩ imset_3 ∩ imset_4

These investigations into sample order persistence inspired the experiments defined in subsection Ei.2.

3.4 CURRICULUM LEARNING VARIANTS EXPLORED

The main focus of the project was to investigate how ConvNets trained with SPL and SPLD principles perform on image data. During the course of project, the experience with applying the two SPL and SPLD methods outlined in sections 2.3.1 and 2.3.2, respectively, gave rise to two alternative methods. The original SPL and SPLD algorithms place a strong emphasis on uniformity in difficulty levels in the epoch when selecting samples at each iteration. There is a sharp (in the case of SPL) and softer (in the case of SPLD) cut-off in difficulty level (or sample loss value) after which no sample is admitted in training. These cut-offs are dictated by the constants λ (13) and γ

(17). It was difficult to control these constants in a way that provides a smooth increase in sample inclusion after each epoch. To achieve this, an alternative implementation was devised that uses a simple ratio (or % from the current training batch size[1]) as a cut-off when including samples. It allows for simpler control over sample inclusion, and can be used to assure a smooth inclusion curve during training. The steepness of the inclusion curve is controlled by a $step$ parameter which increases the % included samples at each epoch. The notable difference is that while the original SPL and SPLD methods assure a uniform sample difficulty distribution at each epoch, this method can admit samples with differing levels of difficulty based on how the samples were distributed in each batch. For example, if one training batch happened to contain a high number of difficult samples, and another a relatively low number, the original SPL and SPLD methods would admit more samples from the second batch than from the first batch. The hereby suggested methods admit an equal amount of samples (the top m% easiest) from all batches. These variants were called p-SPL and p-SPLD.

3.4.1 *p-SPL*

The slightly modified cost function of p-SPL becomes:

$$(\mathbf{w}_{pSPL}, \mathbf{v}_{pSPL}; \tau) = \underset{\mathbf{w}\in\mathbb{R}^d, \mathbf{v}\in[0,1]^n}{\operatorname{argmin}} \left[\sum_{i=1}^{n} v_i L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \tau \sum_{i=1}^{n} v_i \right] \tag{25}$$

Optimizing this cost function is again performed in two steps, by alternatively solving subproblems (26) and (27):

$$\mathbf{v}_{t+1} = \begin{cases} 1 & \text{if } \frac{r}{N} < \tau \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

where r is the rank index of the sample based on its loss value among the current batch of samples processed during the stochastic gradient descent operation. N is the batch size, and $\tau \in (0, 1]$ is a percentage of the batch size.

$$\mathbf{w}_{t+1} = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left[\sum_{i=1}^{n} v_{t+1,i} L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \gamma \sum_{i=1}^{n} v_{t+1,i} \right] \tag{27}$$

1 Batch size here refers to the size of the mini-batches of samples processed during each stochastic gradient descent iteration to perform weight updates. For the experiments in this report, the batch size was set to 100. Thus, 1 training epoch (50,000 unique CIFAR-10 samples) equals 500 training iterations (of 100 samples each).

γ increases with each epoch by a step value, which controls the speed of the increase, until it reaches 1, at which point all samples are included. As noted, this variation places importance on a smooth inclusion of samples between epochs, instead of the equal difficulty level selection across batches. In fact, difficulty levels between included samples may vary depending on the individual batch composition. By contrasting this variation with the original SPL algorithm, we can also gain insight into the importance of keeping difficulty levels even when performing sample selection across batches.

3.4.2 *p-SPLD*

The cost function for p-SPLD is given in (28):

$$L_{pSPLD}(\mathbf{w}, \mathbf{v}; \tau, \gamma) = \sum_{i=1}^{n} v_i L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \lambda \sum_{i=1}^{n} v_i - \gamma \sum_{j=1}^{b} \|\mathbf{v}^{(j)}\|_2 \quad (28)$$

The function is optimized iteratively, by solving the following subproblems:

$$\mathbf{v}_{t+1} = \begin{cases} 1 & \text{if } \frac{j}{N} < \gamma \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

$$\mathbf{w}_{t+1} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \left[\sum_{i=1}^{n} v_{t+1,i} L(y_i, f(\mathbf{x}_i, \mathbf{w})) - \beta \sum_{j=1}^{b} \|\mathbf{v}_{t+1}{}^{(j)}\|_2 \right] \quad (30)$$

β carries the same meaning as in (25), and j is the rank of the sample *within its cluster* in the currently processed batch. The notable difference from the regular SPLD method is that the soft cut-off is removed, i.e. there is no "bonus" boundary past the hard cut-off where samples are given another chance for inclusion based on their rank within their class cluster. p-SPLD simply includes the easiest m% as specified by γ, evaluated within each cluster. So each cluster has the right to include the same % of samples as any other cluster. If the samples from each cluster are approximately evenly distributed across training batches, this should equate to approximately the same number of samples included from each cluster per batch, even if the difficulty levels between clusters vary wildly.

3.5 IMPLEMENTATION DETAILS

The various curriculum learning experiments for this report were implemented using the CAFFE [42] framework. CAFFE allows a variety

of neural network definitions to be simply expressed in a declarative model using the protobuf [43] protocol. The network structure is defined as a protocol buffer message types in .proto files. Each protocol buffer message is a small logical record of information, containing a series of name-value pairs that fully define the number and type of layers, inputs and outputs of the network. The forward inference and backward learning can also be defined at every layer. The input and output from each layer are accessible as Blobs. Blobs are 4-D arrays for storing and communicating information. They can hold data, derivatives, and parameters. In short, CAFFE provides the tools to define and train a great variety of neural networks, including ConvNets.

As previously noted, ConvNets learn the optimal model parameters $\mathbf{w}$ by minimizing the value of an objective (loss) function $L(\cdot)$ by iterative gradient descent. While CAFFE comes with a variety of possible pre-defined network loss layers out of the box, this experiment required implementing several custom loss layers that can dictate the order in which the training examples are presented.

3.5.1 *SPL and p-SPL Implementation*

Two custom CAFFE network layers were implemented to enable SPL and p-SPL:

SPL: The first variant follows equation (13). Samples are removed if their loss value is greater than $\lambda = \frac{1}{K}$, and K is decreased at each epoch by a constant step.

P-SPL: In the second variant, the samples are first sorted based on loss value. Then, only the easiest $m\%$ of the samples from each batch are included in training.

3.5.2 *SPLD and p-SPLD Implementation*

SPLD: This CAFFE layer implements the algorithm as outlined in (17).

P-SPLD: In this variant, the easiest $m\%$ samples from each class cluster in the current batch are selected for training.

Sample removal is facilitated by excluding a given sample's gradient from the backpropagation process.

Training ConvNets using stochastic gradient descent is a batch operation, so it is worth to note that sample selection is performed per batch, not per epoch. Both the K (or λ) value and the percentage m are updated per epoch, but the sample selection is performed at the batch level, not at the epoch level. For example, in the case of p-SPL,

the algorithm will not select 60% of the easiest samples based on the loss values from all samples in the current epoch. It will select 60% of the easiest samples in the current batch, based on the loss values from the samples in that particular batch. This operation is repeated for all batches in the epoch. In the case of SPL, all samples in all batches from the epoch will be examined against the same λ value.

For the p-SPLD implementation, all samples within a batch are sorted by loss value within their class cluster. A certain percentage m from each class cluster are included in training. The difficulty levels between samples from different classes may therefore not be consistent, or not as consistent as when using SPLD.

3.5.3 *ConvNet architecture*

A 9-layer ConvNet with 3 convolutional, 3 pooling, 2 normalization, and 1 fully connected, softmax layers was used as a baseline for all experiments, unless noted otherwise. The detailed diagram of this network's architecture is given on Figure 23. For parameter values used in the individual layers, please see Appendix, section A.1. This network will be referred to as the "Default network" further in the document.

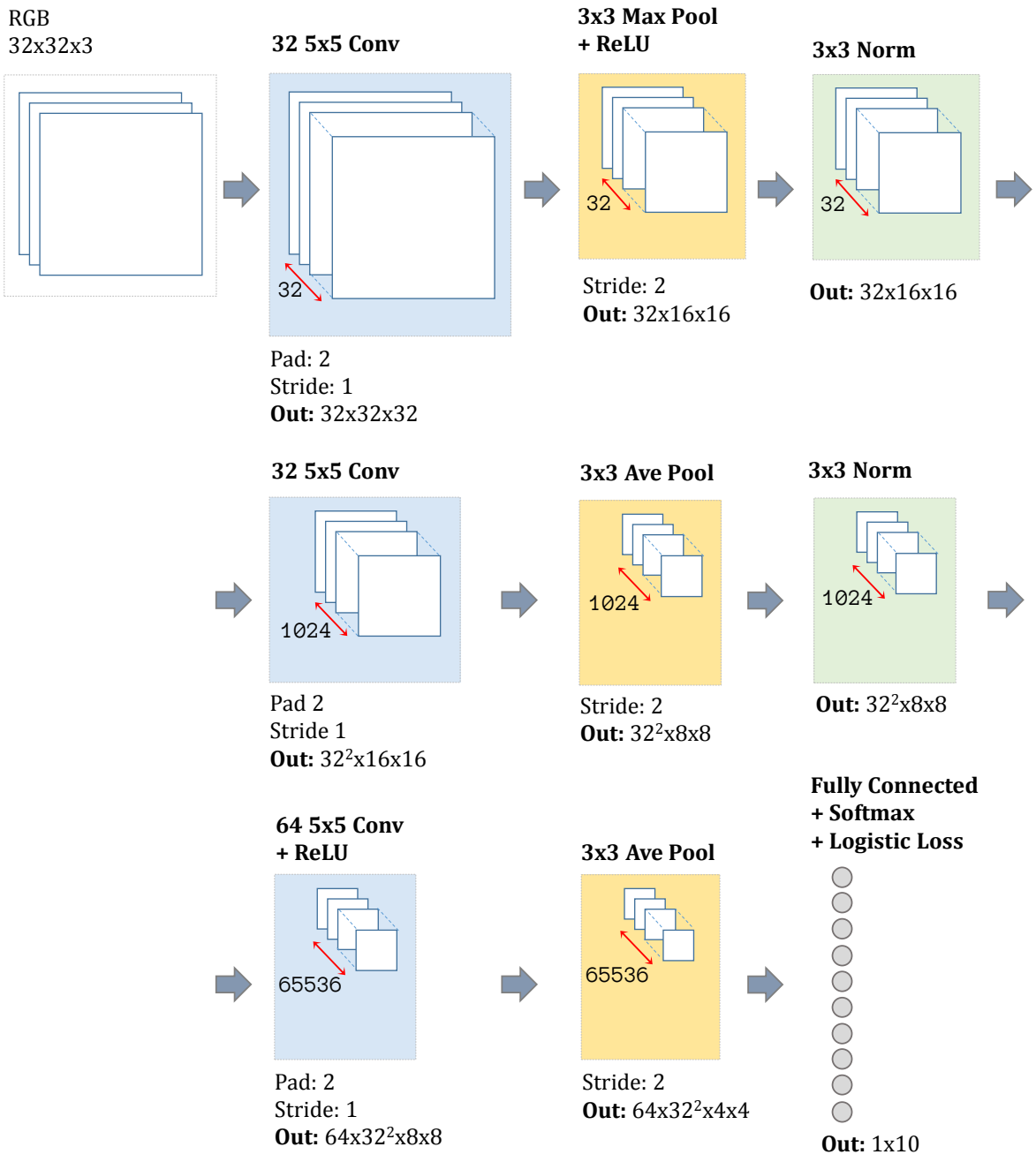


Figure 23: *Default network diagram.*

Note that the performance of this network on CIFAR-10 ($\approx$ 81% accuracy) is far from the best recorded for this dataset (which is currently over 95%), but it was chosen for the experiments because 81% accuracy leaves plenty of room for improvement for the new curriculum learning techniques.

All SPL and SPLD networks used in the experiments have the same architecture as the Default network, with only the last loss layer re-

placed with the customized SPL, p-SPL, SPLD or p-SPLD sotmax loss layers, respectively.

3.5.4 *Training parameters*

This section outlines the most important parameters used in training the Default, SPL and SPLD networks. As usual, all samples from the training set (50,000 for CIFAR-10) are examined in a single epoch. An epoch consists of multiple iterations, with one iteration performed over a single batch (subset) of training data. The batch size was set to 100, and training was performed for 70,000 iterations (for a total of 140 epochs) with learning rate parameters shown in table 1.

ITERATIONS	LEARNING RATE
$0 - 60,000$	0.001
$60,000 - 65,000$	0.0001
$65,000 - 70,000$	0.00001

Table 1: *Training parameters for Default, SPL and SPLD networks on CIFAR-10*

The momentum was set to 0.9, and weight decay to 0.004. Additionally, the following parameters were used in the various SPL and SPLD experiments:

`include_perc`	initial percentage samples to include in p-SPL and p-SPLD
`perc_step`	percentage step increase per epoch in p-SPL and p-SPLD
`warmup_epoch`	number of initial "warmup" epoch, that include all samples
`lambda`	value that sets difficulty level for SPL and SPLD
`lambda_step`	step increase for λ per epoch
`gamma`	value that extends a softer cutoff limit for SPLD
`exclude_global`	defines a predefined set of samples that are always excluded

3.6 EXPERIMENT DESIGN

Most of the included experiments are designed for comparing the different curriculum learning techniques outlined above. The parameters that were kept constant are the network architecture (with a different last layer for each network type), the training parameters, which are the ones listed in table 1, and the batch size, unless explicitly stated otherwise. All experiments involve 5 independent training runs on a set of 5 randomly generated CIFAR-10 databases, unless stated otherwise. The graphs usually show the average of these 5 training runs, and 1 standard deviation from the mean as a gray area.

EI *Experiment Set I: Default Network with Standard Training*

These experiments use the default network setup as described in 3.5.3. Accuracy and loss are averaged among 5 independent runs on 5 randomly generated CIFAR-10 databases.

EI.1 *Regular Run*

During this run, the network is trained the regular way, and randomly examines all 50,000 training samples from CIFAR-10 per epoch. The accuracy and loss numbers obtained from this experiment serve as the base point for comparison with the other 4 types of curriculum learning networks.

EI.2 *Partial Data*

This experiment explores how removing some of the training samples affects network performance. It was inspired by the ideas presented in "Are all training examples equally valuable?"[44] by Lapedriza, Pirsiavash, Bylinskii, *et al.* However, instead of determining the value of each sample by training a classifier as described in [44], the sample's difficulty is used as a measure of its importance during training. This is done based on the investigation described in section 3.3, where we observed that the top 20% sets of easy and difficult samples remain fairly consistent within the same run. What would happen to accuracy if the top m% was not included in training at all? How about the bottom m%? Does the difficulty levels matter at all when compared to random sample removal? For these experiments, a percentage of samples is removed permanently from the training set, and the effects on training are recorded. The % values explored are 10, 20, 40, 60 and 80. There are 3 variants of this experiment:

VARIANT 1: RANDOM REMOVAL The % samples are removed randomly. The same random set is removed permanently from training.

VARIANT 2: RANKED REMOVAL: EASY FIRST A training run is first performed using the usual setup. After the 70th epoch, the loss values for all samples are recorded. Then, these values are used to determine the % easiest samples to be permanently excluded from training in a new network run, so that this pre-determined set is effectively never seen by the network.

VARIANT 3: RANKED REMOVAL: DIFFICULT FIRST Same setup as Variant 2, but the top m% most difficult samples are removed instead of the top m% easiest ones.

EII *Experiment Set II: SPL Training*

These experiments are performed with two variants of the SPL network: SPL and p-SPL.

EII.1 *SPL*

5 runs on 5 random databases as usual, with the SPL network setup described in section 3.5.1. The parameters varied during the experiments are the base sample inclusion value, λ, $\lambda - step$, and warmup iterations.

EII.2 *SPL Inversed*

5 runs on 5 random databases as usual, with the p-SPL network setup described in section 3.5.1. Same setup as Eii.1, but what happens if instead of preferring easy samples, the network chooses difficult samples first? This experiment reverses the sorting order of the loss values per sample, so that most difficult samples are included first, and the difficulty decreases instead of increasing during training, until all samples are included.

EII.3 *p-SPL Quick Training, Normal and Reversed*

This experiment was designed to provide better statistics on the difference between a network with standard training, p-SPL and p-SPL Inversed networks by performing a greater number of experiments with each network. The experiment is based on the following setup: 5 CIFAR10 databases were used. Each database contains the $50,000$ training samples and $10,000$ test samples in different, random order. 3 types of ConvNets were trained with 4 runs on each of these 5 databases, for 20 runs per network:

1. A ConvNet like the Default network, but with the normalization layers removed, and one extra fully connected layer added before the last softmax loss layer. The architecture is given in listing A.2.
2. A p-SPL network, same architecture as 1, but with p-SPL layer instead of `SOFTMAX_LOSS` layer.
3. A p-SPL network like 2, but with the samples sorted by difficulty in ***decreasing*** order, i.e. the SPL layer is "inversed" to prefer samples of high difficulty first, with difficulty level decreasing as training progresses. This is equivalent to the setup in Eii.2.

20 runs were performed by each network: 4 runs through the 5 databases. This gives a total of 60 runs for the 3 network types. The

training parameters used for all runs are shown in table 2. The momentum was set to 0.9, and weight decay to 0.004, batch size to 100.

ITERATIONS	LEARNING RATE
$0 - 4,000$	0.001
$4,000 - 9,000$	0.0001

Table 2: *Training parameters for Experiment Set II: p-SPL Quick Training*

EII.4 *p-SPL*

5 runs on 5 random databases as usual, with the p-SPL network setup described in section 3.5.1.

EII.5 *p-SPL Inversed*

Same concept as Eii.2, but with the p-SPL network.

EIII *Experiment Set III: SPLD Training*

The experiments are performed with the SPLD and p-SPLD networks. The parameters varied during the experiments are the base sample inclusion value, λ, $\lambda - step$, and warmup iterations. γ is kept constant with a value of 0.2.

EIII.1 *SPLD*

The 5 usual runs with a regular SPLD network setup as described in 3.5.2.

EIII.2 *SPLD Inversed*

: Same network as Eiii.1, with inversed sample selection as described in Eii.2.

EIII.3 *p-SPLD*

Same experiment with the p-SPLD network as described in 3.5.2.

EIII.4 *p-SPLD Inversed*

Same network as Eiii.3, with inversed sample selection as described in Eii.2

3.7 DATA ANALYSIS

3.7.1 *Metrics*

The main metrics used to compare the performance of the different networks are outlined below:

- % classification accuracy, or % classification error on test set
- Learning "speed", measured in terms of % accuracy achieved per number of training iterations

3.7.2 *Hypotheses*

Using a comparative approach and the metrics outlined in section 3.7.1, the following hypotheses will be examined:

H1 *The accuracy decline of the Default ConvNet trained with a certain percentage* r_p *of training samples removed is less than 0.5%.*

This hypothesis implies that for some values of $r_p \in (0, 1)$, the overall effect on network performance is negligible. This offers an $O(N * r_p)$ improvement in training speed, if the network normally trains in $O(N)$, where N is the number of training samples.

H2 *A network trained with easiest samples removed performs worse than a network trained with the most difficult samples removed*

Does removing the easy samples up to a given percentage have a detrimental effect on training? How does it compare to removing the most difficult samples instead?

H3 *A network trained with the most difficult samples removed performs better than a network trained with random samples removed, and within* 0.2% *margin of the default network*

H4 *A network trained using SPL performs better than the Default network*

This hypothesis implies that a network trained using SPL while network architecture and learning parameters are otherwise kept the same achieves higher % accuracy than the Default network.

H5 *A network trained using SPLD performs better than the Default network*

Similarly, this hypothesis implies that a network trained using SPLD achieves higher % accuracy, (or converges faster) than the Default network.

H6 *A network trained using SPLD performs better than the same network trained using SPL*

These hypotheses will not be formally tested using standard statistical models, but merely used as a guideline when designing the experiments.

3.8 DELIMITATIONS

Since the experiments involve many parameters, which produce an exponential increase in the number of possible experiments, and the fact that the networks used take a relatively long time to train for each experiment, the following delimitations have been made in this project:

3.8.1 *Network architectures*

Due to time constraints, network architectures (various types and number of layers) other than the ones described in section 3.6 will not be explored.

3.8.2 *Training parameters: iterations, learning rate changes, λ, γ, m%, etc.*

Due to the infinite amount of possible combinations of these parameters, cuts had to be made on how many values can be explored so that the project can be completed within a reasonable time frame. The chosen experiment parameters may not have been optimal.

3.8.3 *Data sets*

The experiments are performed only on CIFAR-10, again due to time constraints.

3.8.4 *Batch sizes*

The batch size during stochastic gradient descent is an interesting parameter that was insufficiently explored in the experiments due to time constraints. For all reported results, the batch size is set to 100.

Part IV

RESULTS AND DISCUSSION

The following chapter details and discusses the results from the performed experiments.

4 RESULTS AND DISCUSSION

4.1 DIAGRAM DETAILS

Before venturing into the results section, some diagram explanations are in order. The diagrams of all curriculum learning variants include the Default network performance as a reference. Its label on the loss and accuracy charts is **"default"**. All accuracy and loss functions are given as the average of 5 runs (1 run on 5 different random databases), unless stated otherwise. The symbol `lr` represents the learning rate. The regions where the learning rate changes are labeled with the `lr` value, and colored differently. Under each training technique, *only* the best experiment result is shown, for brevity. The Sample Inclusion diagrams visualize how the value of included samples changes at each iteration. The total number of training samples in CIFAR-10 is 50,000, and the yellow bars at each epoch represent the number of samples currently included in training.

For the SPL and SPLD networks, the following parameters were varied:

1. λ (SPL and SPLD)
2. λ_step, (SPL and SPLD)
3. `perc_include_start`: 4 values: 20, 40, 60, 80 (p-SPL and p-SPLD)
4. `perc_step`: 4 values, to provide smooth sample inclusion during training (p-SPL and p-SPLD)
5. `warmup_epochs`: 2 values: no warmup and 2 warmup epochs
6. γ was set to 0.2 (SPLD)

A few more points of interest when reading the diagrams:

1 ITERATION represents one run through 100 samples (one batch).

1 EPOCH represents one run through all 50,000 samples. 1 epoch = 500 iterations.

4.2 EXPERIMENT RESULTS

EI *Experiment Set I: Default Network*

EI.1 *Regular run*

The results were obtained using the Default network, with the experiment setup described in section Ei.1. The average accuracy on the test set is shown on figure 24. The average loss on the test set is shown on figure 25.

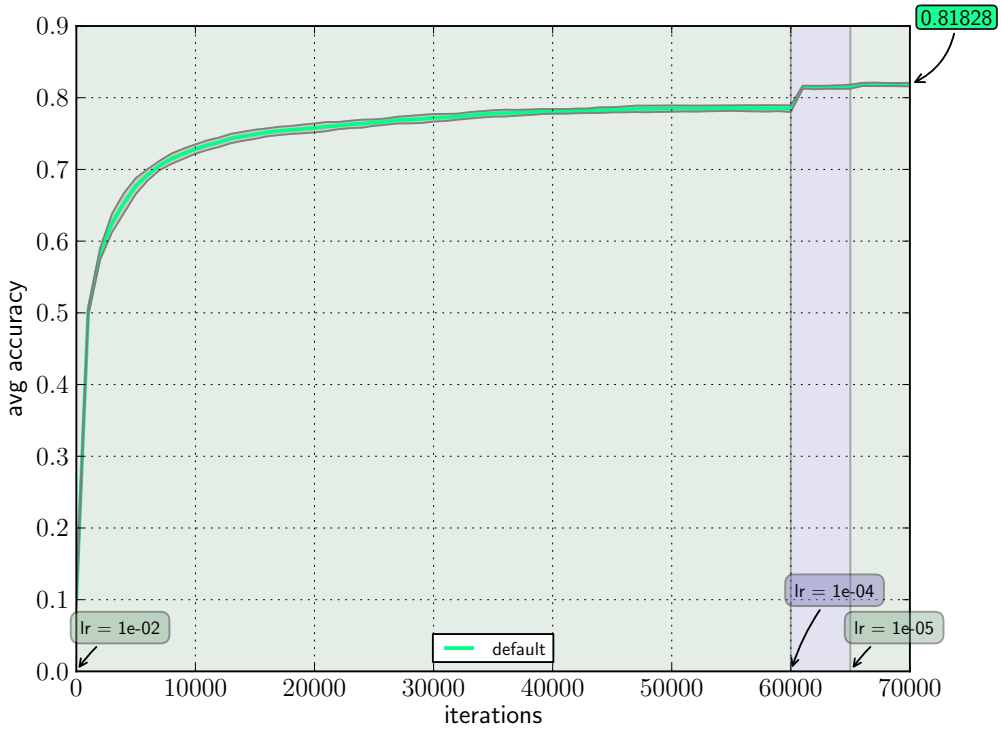


Figure 24: *Average accuracy on test set from 5 runs using Default network.*

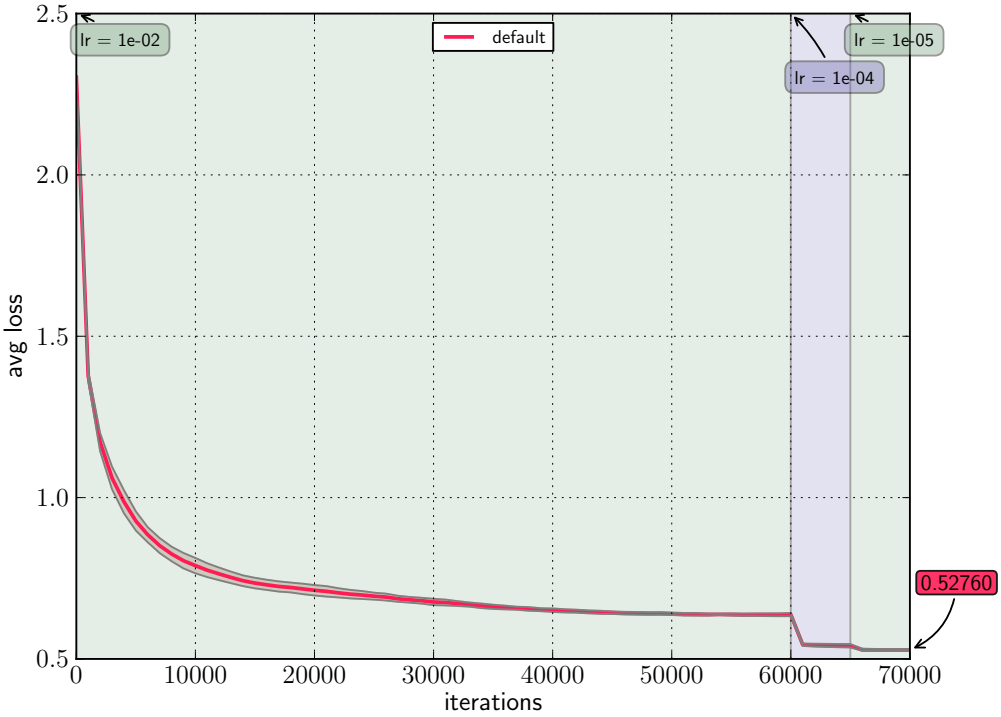


Figure 25: *Average loss on test set from 5 runs using Default network*

The accuracy and loss functions follow their usual course observed during ConvNet supervised training. The standard deviation is very small, so training proceeds largely in the same fashion across the different runs. The network achieves ≈ 82% accuracy. The results are summarized in table 3.

NETWORK	ACCURACY	LOSS
Default	0.81828	0.52760

Table 3: *Accuracy and loss results for the Default network*

EI.2 *Partial Data*

Ei.2.

VARIANT 1: RANDOM REMOVAL A randomly selected subset of samples is permanently removed from training. The experiment was performed with 10%, 20%, 40%, 60%, and 80% sample removal. The average accuracy from these runs is shown on figure 26. We can observe that difference between the Default network's performance and same network with 10% random sample removal is negligible at 0.0083. The network loses less than 2% in accuracy when 20% are removed, and less than 3% when 40% samples are removed, as shown in table 4.

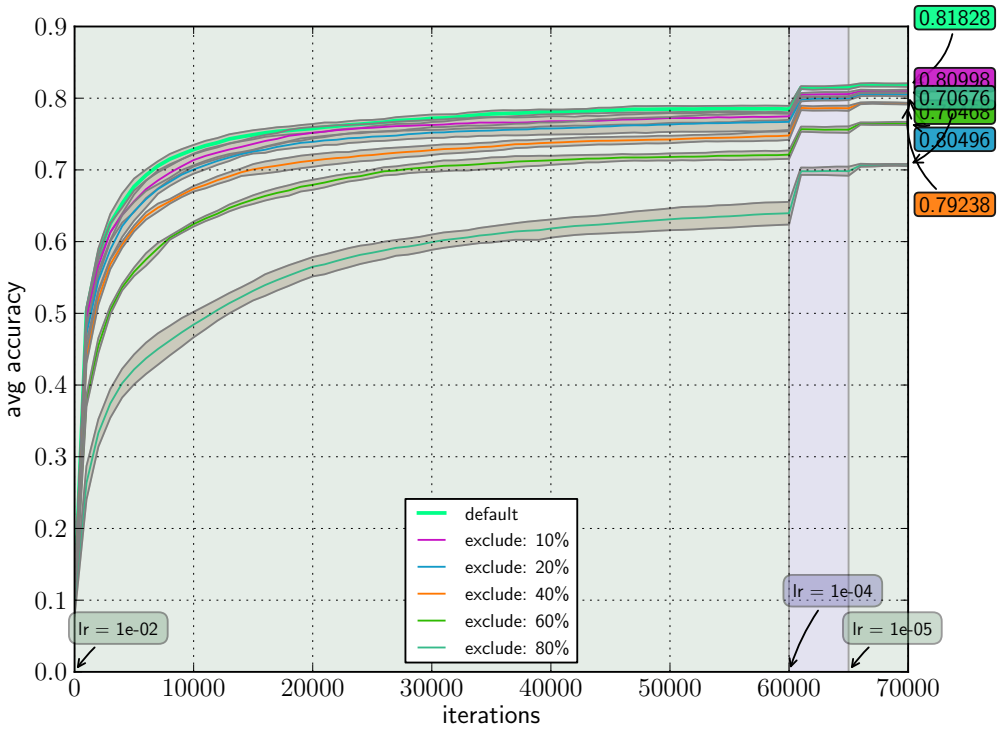


Figure 26: *Average accuracy on test set from 5 runs using Default network, with percentage random samples removed.*

PARAMETERS	ACCURACY	DIFF
default	**0.8183**	**0**
exclude: 10%	0.8100	0.0083
exclude: 20%	0.8050	0.0133
exclude: 40%	0.7924	0.0259
exclude: 60%	0.7647	0.0536
exclude: 80%	0.7068	0.1115

Table 4: *Random sample removal: effect on accuracy*

VARIANT 2: RANKED REMOVAL: EASY FIRST This experiment was performed under the condition described in section Ei.2. The set of samples to be removed (for both the easy first and difficult first variants) was determined based on the loss values of all 50,000 samples taken at epoch 70 from a regular network run. The training curves are shown on Figure 27.

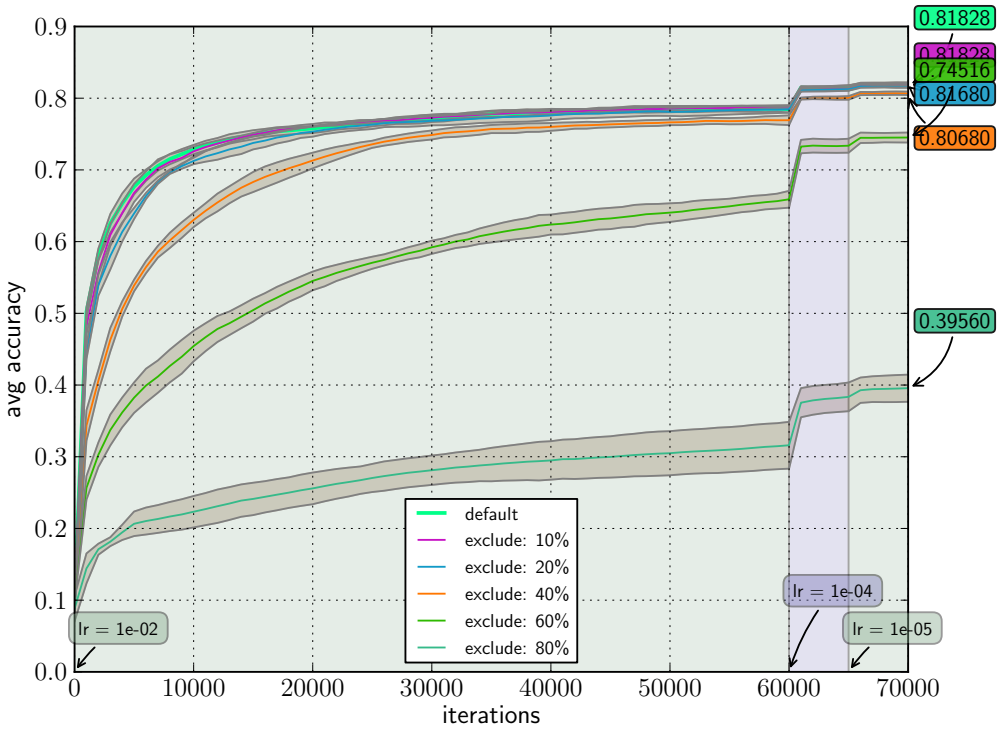


Figure 27: *Accuracy for different % of easiest samples removed*

The results are summarized in table 5. It is interesting that removing the easy samples improves performance compared to removing random samples. There is no difference in performance between the Default network, and a network trained with the top 10% easy samples removed! Also notable is the overall great performance of the network for up to 40% of easiest samples removed (only $\approx$ 1% accuracy drop), and the sharp deterioration at 80%, compared to random removal. There is also higher variance in the training curves at lower percentage points than the ones from the random removal network.

PARAMETERS	ACCURACY	DIFF
default	**0.8183**	**0**
exclude: 10%	0.8183	0
exclude: 20%	0.8168	0.0015
exclude: 40%	0.8068	0.0115
exclude: 60%	0.7452	0.0731
exclude: 80%	0.3956	0.4227

Table 5: *Ranked sample removal: accuracy for % easiest samples removed*

VARIANT 3: RANKED REMOVAL: DIFFICULT FIRST This experiment was performed under the condition described in section Ei.2. The training curves for different percentage points are shown on Figure 28.

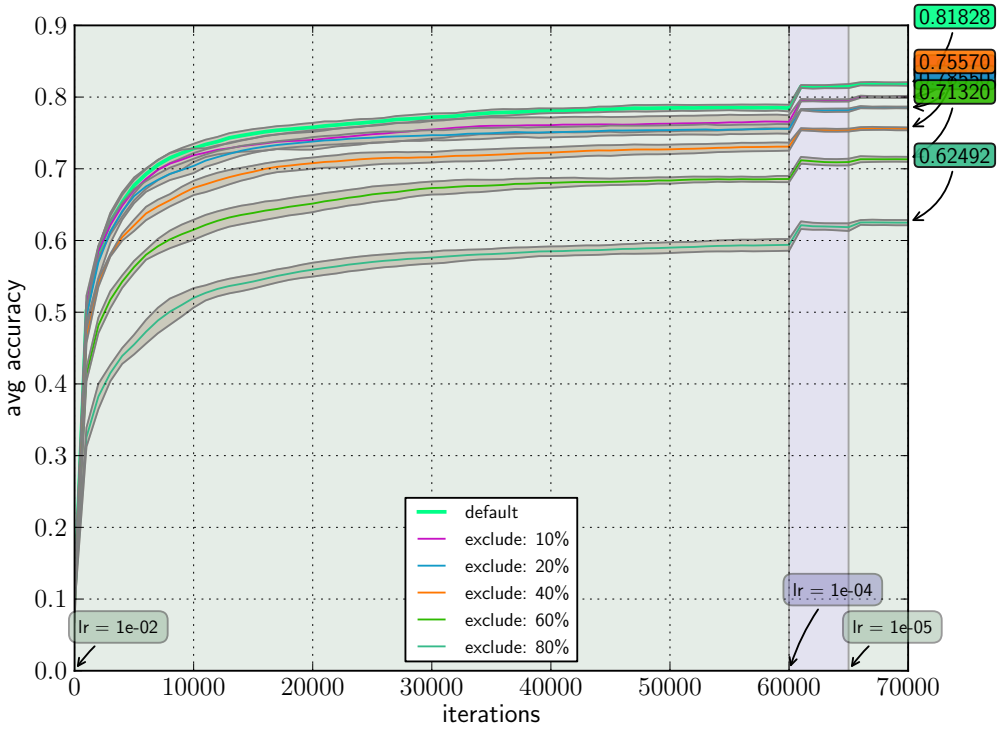


Figure 28: *Accuracy for different % of most difficult samples removed*

The results are summarized in table 6. Removal of the difficult samples causes a notably worse performance compared to the removal of the easiest samples. The difficult samples appear to be more valuable for the learning of the network than the easy ones.

PARAMETERS	ACCURACY	DIFF
default	**0.8183**	**0**
exclude: 10%	0.8002	0.0181
exclude: 20%	0.7855	0.0328
exclude: 40%	0.7557	0.0626
exclude: 60%	0.7132	0.1051
exclude: 80%	0.6249	0.1934

Table 6: *Ranked sample removal: accuracy for % most difficult samples removed*

Figure 29 shows a plot of the results for the 3 partial data experiments.

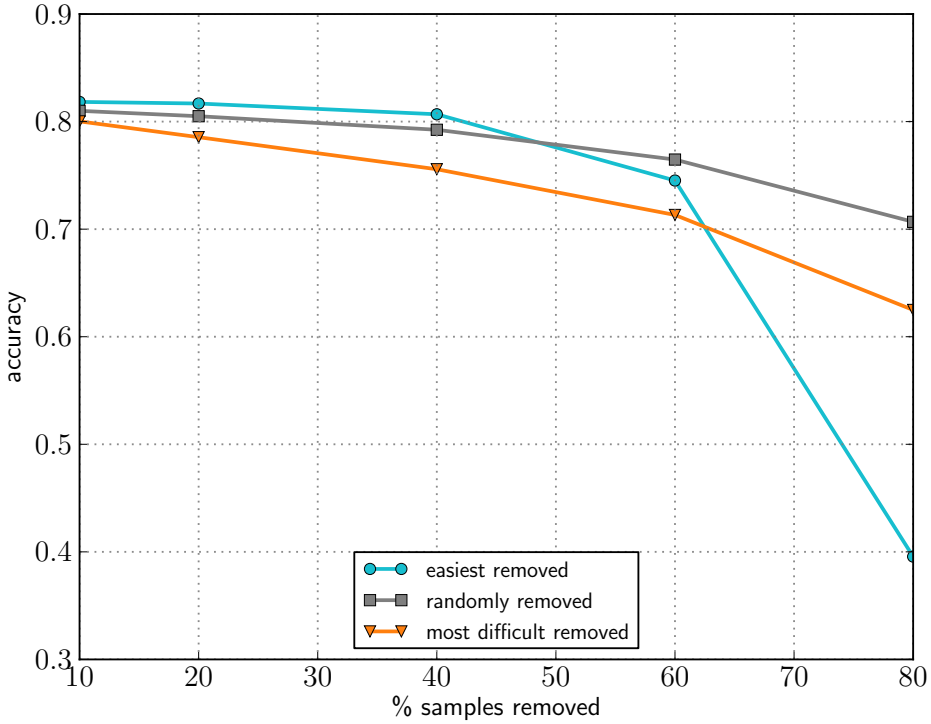


Figure 29: *Avg. accuracy vs. % sample removal plot for the random, most difficult first, and easiest first partial data experiments.*

Table 7 shows the accuracy results from all sample removal experiments. The best results per %-removed set is shown in bold.

% REMOVED	AVG. ACCURACY		
	random	easiest	most difficult
10	0.81	**0.8183**	0.8002
20	0.805	**0.8168**	0.7855
40	0.7924	**0.8068**	0.7557
60	**0.7647**	0.7452	0.7132
80	**0.7068**	0.3956	0.6249

Table 7: *Avg. accuracy results for partial data experiments.*

From Figure 29 and Table 7, it is clear that removing easiest samples first is least harmful to the performance of the network, up to a given point, after which performance rapidly deteriorates. This point may be significant when determining the quantity of viable, useful samples in a given data set, after which additional samples do not provide much benefit.

EII *Experiment Set II: SPL Training*

EII.1 *SPL*

The best achieved avg. accuracy results from 5 runs with two λ settings are shown on figure 30:

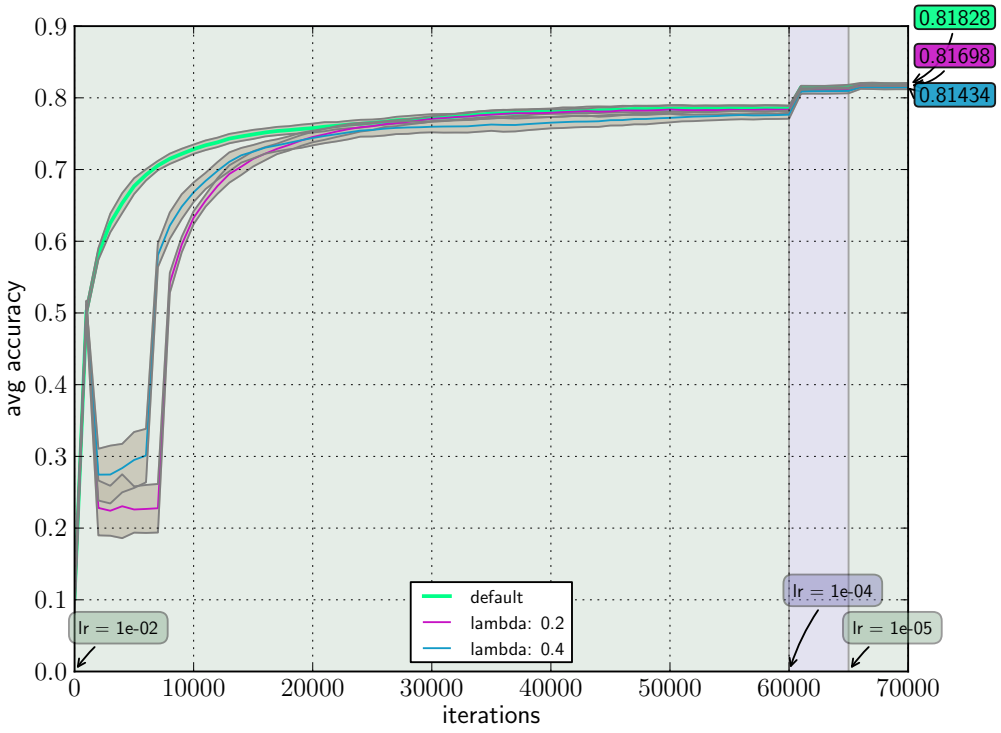


Figure 30: *Average accuracy on test set from 5 runs using SPL network.*

The best result was achieved with the following settings:

PARAMETER SETTINGS:

`lambda:`	0.2
`lambda_step:`	0.06
`warmup_epoch:`	2

Convergence speed is shown on figure 31:

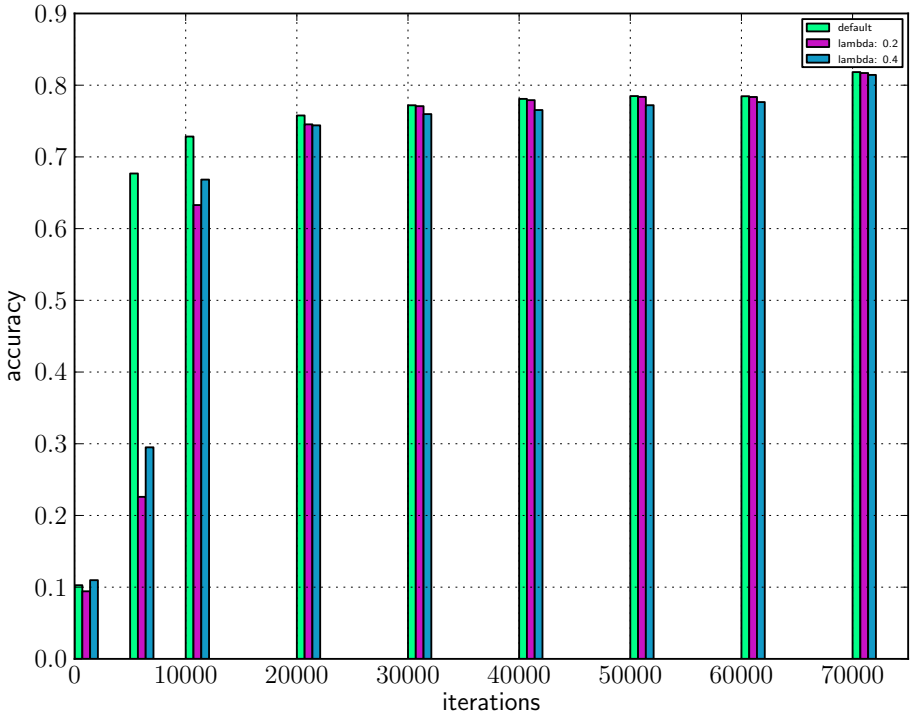
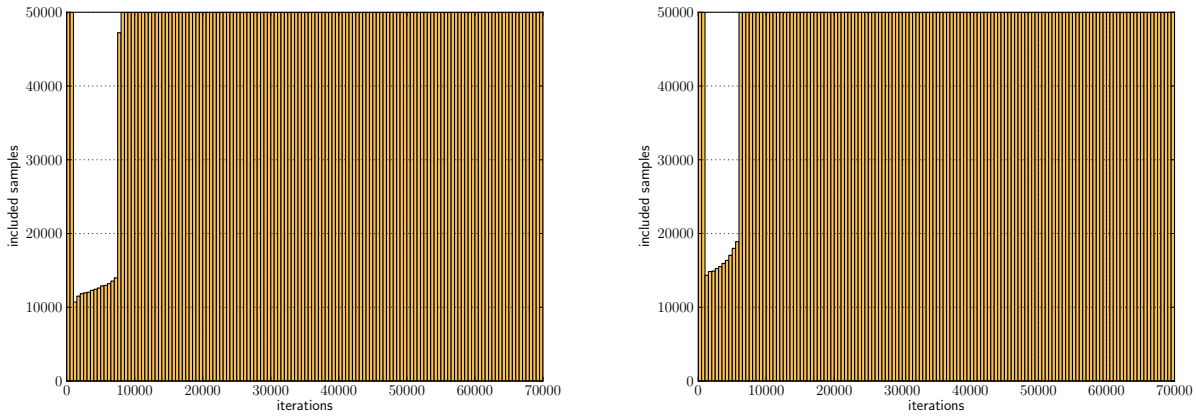


Figure 31: *Convergence speed on test set from 5 runs using SPL network.*

Included/excluded samples per iteration are summarized on figure 32.



(a) $\lambda = 0.2$ (b) $\lambda = 0.4$

Figure 32: *Sample inclusion for 4 different start percentage points*

The results are summarized in table 8.

PARAMETERS	ACCURACY	LOSS
default	**0.8183**	**0.5276**
lambda: 0.2	0.8170	0.5312
lambda: 0.4	0.8143	0.5431

Table 8: *Result summary from best experiment for SPL*

EII.2 *SPL Inversed*

This is the network setup as described in section Eii.2 This experiments were performed using combinations of 4 different start values for λ, 3 λ step increases, and 0 or 2 warmup epochs.

The best result was achieved with the following settings:

PARAMETER SETTINGS:

`lambda:`	0.4
`lambda_step:`	0.025
`warmup_epoch:`	0

Avg. accuracy results from 5 runs for the 4 different `lambda` settings are shown on figure 33. The reversed network performs better than the regular one, and in this case, better than the Default as well. Also of note is how tight the variance is around the training curves.

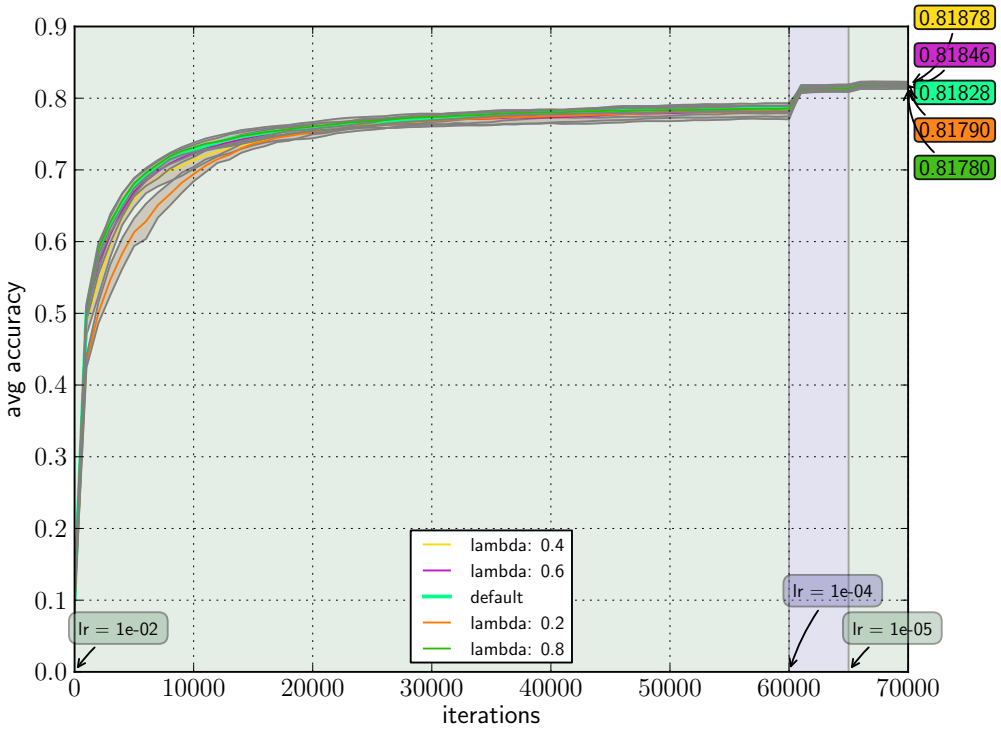


Figure 33: *Average accuracy on test set from 5 runs using SPL-Inversed network.*

Convergence speed is shown on figure 34. Accuracy is proportional to the amount of samples included at the beginning of training, but equalizes to a very narrow margin towards the end of training.

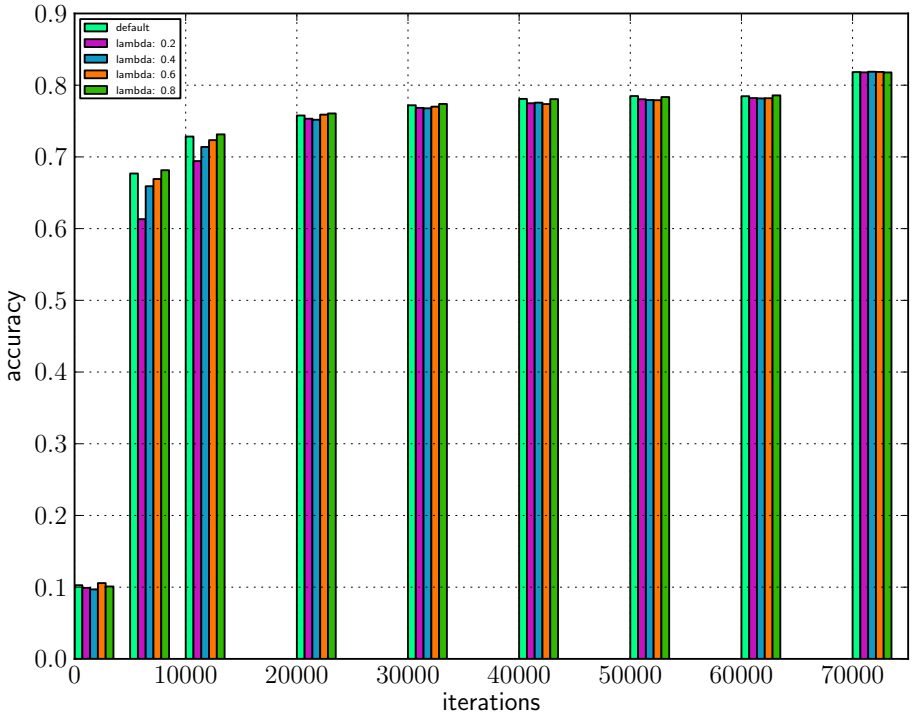
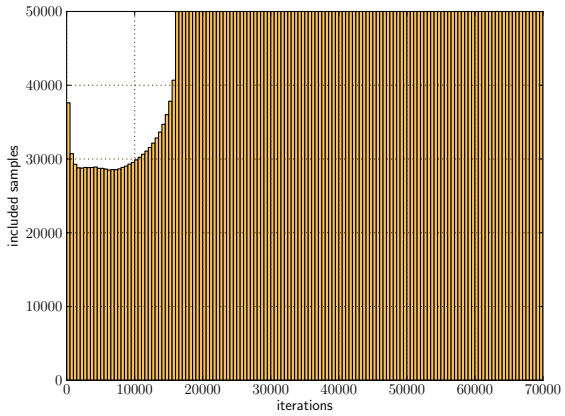
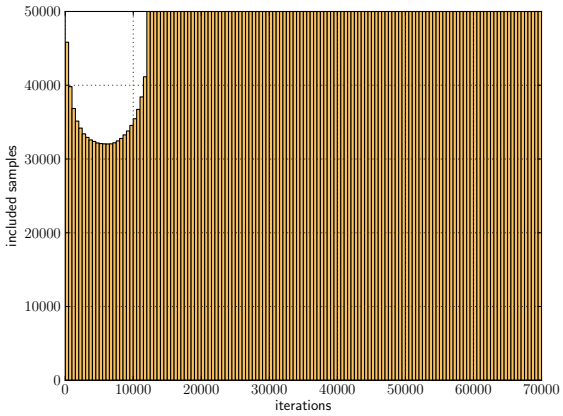


Figure 34: *Convergence speed on test set from 5 runs using SPL-Inversed network.*

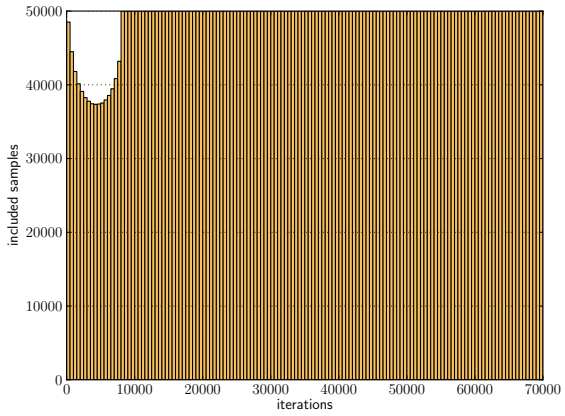
Included/excluded samples per iteration are shown on figure 35. We can observe that during this run, the majority of samples were included before the 20,000 iteration mark.



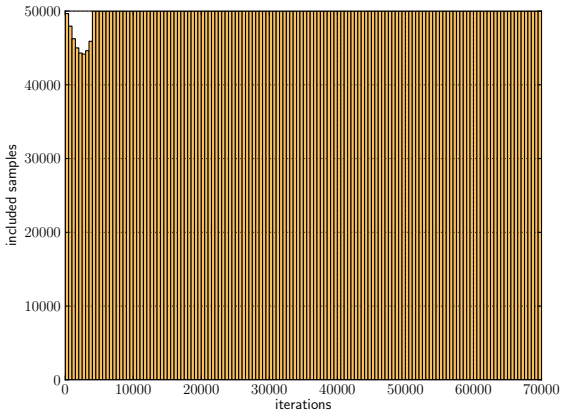
(a) $\lambda = 0.2$



(b) $\lambda = 0.4$



(c) $\lambda = 0.6$



(d) $\lambda = 0.8$

Figure 35: *Sample inclusion for 4 different start percentage points*

The results are summarized in table 9. It is interesting to observe that the most successful runs were in the mid-range of λ.

PARAMETERS	ACCURACY	LOSS
lambda: 0.4	**0.8188**	**0.5289**
lambda: 0.6	0.8185	0.5294
default	0.8183	0.5276
lambda: 0.2	0.8179	0.5274
lambda: 0.8	0.8178	0.5305

Table 9: *Result summary for SPL-Inversed experiment*

EII.3 *p-SPL Quick Training, normal and reversed*

These experiments were performed using the setup described in section Eii.3. Training only proceeds up to 9,000 iterations, to facilitate a larger number of independent runs for better statistics. For the p-SPL and p-SPL-Reversed networks, the sample include start value was set to 50%, and the step increase to 4%. Table 10 shows the included sample counts per epoch.

Epoch	Included Samples	Excluded Samples
1	25000	25000
2	27000	23000
3	29000	21000
4	31000	19000
5	33000	17000
6	35000	15000
7	37000	13000
8	39000	11000
9	41000	9000
10	43000	7000
11	45000	5000
12	47000	3000
13	49000	1000
14	50000	0
15	50000	0
16	50000	0
17	50000	0
18	50000	0

Table 10: *% samples included at each epoch for p-SPL and p-SPL-rev quick training experiment.*

Figure 36 shows the accuracy plot. The regularly trained network and the pSPL-Inversed network perform similarly, while the pSPL network lags behind.

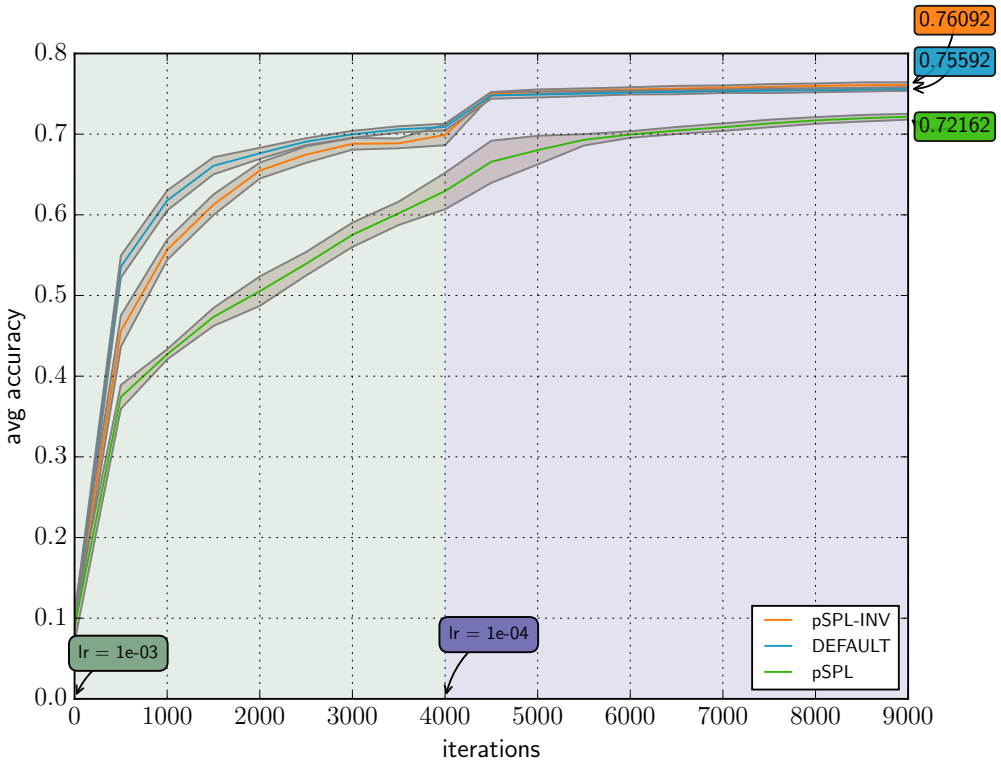


Figure 36: *Avg. accuracy for regular, p-SPL and p-SPL-Inversed networks.*

The results are summarized in the box plot shown on Figure 37. It is interesting to observe that the p-SPL network performs much worse than both the regular, and the p-SPL-inversed networks. The inversed network starts by including the most samples difficult samples first, and relaxes the difficulty requirement as training progresses, to include easier and easier samples. This is the opposite behavior from the concept of SPL, yet it seems that it is better for the performance of the network in this experiment. In fact, the reversed p-SPL network finishes with a slightly higher median than the regular network.

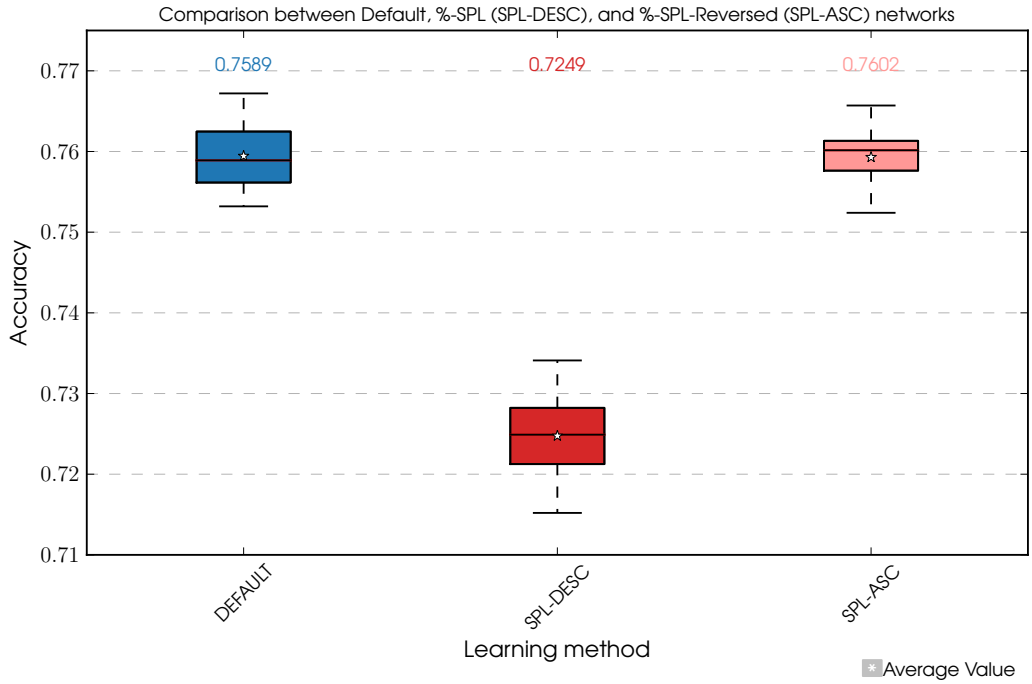


Figure 37: *Boxplot for the accuracy measured from 20 runs on randomized databases with CIFAR10 images with 3 types of networks. The median is represented by the horizontal line inside the box. The borders of the box are set at the 25th and the 75th percentiles. The whiskers extend to the most extreme data point within* $1.5 * (75\% - 25\%)$ *data range*

EII.4 *p-SPL*

This is the experiment setup as described in section Eii.4. The goal of this experiment was to keep a steady increase in included samples, starting with a certain percentage of included samples, and finishing with all samples getting included towards the last iteration. The start percentage values were varied between 0.2 and 0.8, and the percentage increase rate was determined using eq. (31):

$$\text{perc\_step} = \frac{1 - \text{include\_perc\_start}}{\text{total\_epoch} - \text{warmup\_epoch}} \tag{31}$$

The best result was achieved with the following settings:

PARAMETER SETTINGS:

include_perc_start :	0.8
perc_step :	0.0017
warmup_epoch:	0

Avg. accuracy results from 5 runs for the 4 different `include_perc` settings are shown on figure 38. The curves are notably jagged, as

the network is struggling to catch up with the Default for the entire duration of training. The failure of the 20% run was due to an integer division problem, so it is not significant for the experiment.

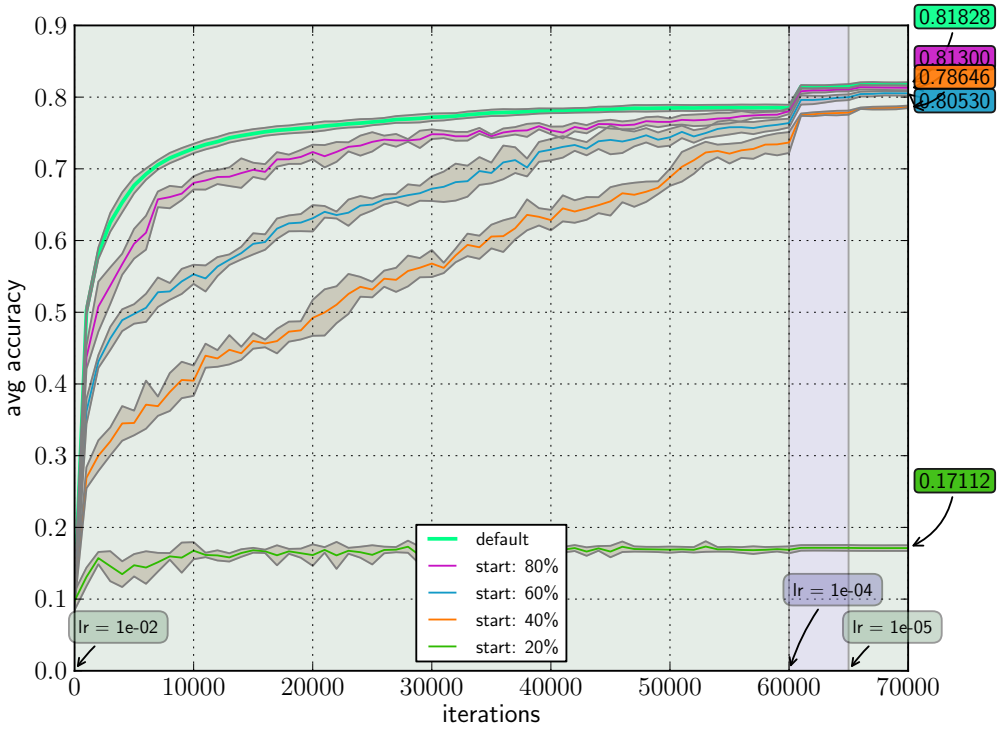


Figure 38: *Average accuracy on test set from 5 runs using p-SPL network.*

Convergence speed is shown on figure 39. The p-SPL network is far behind the Default during the majority of training.

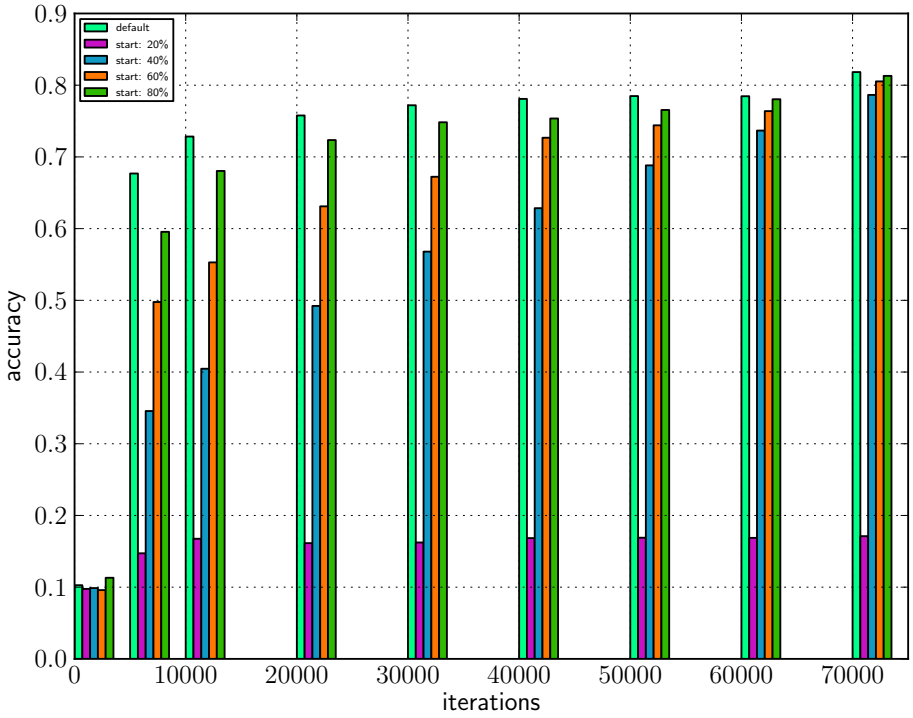
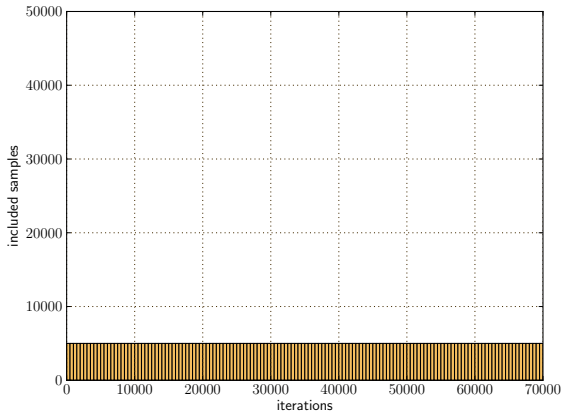
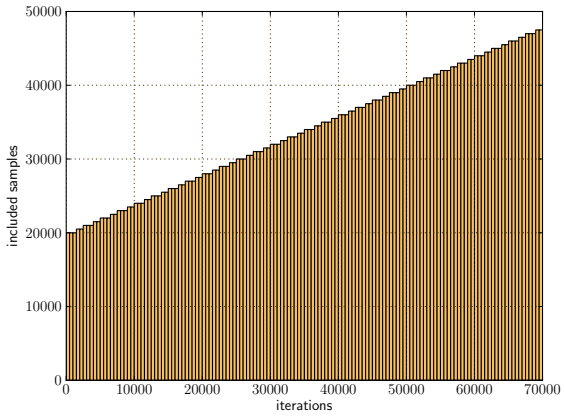


Figure 39: *Convergence speed on test set from 5 runs using p-SPL network.*

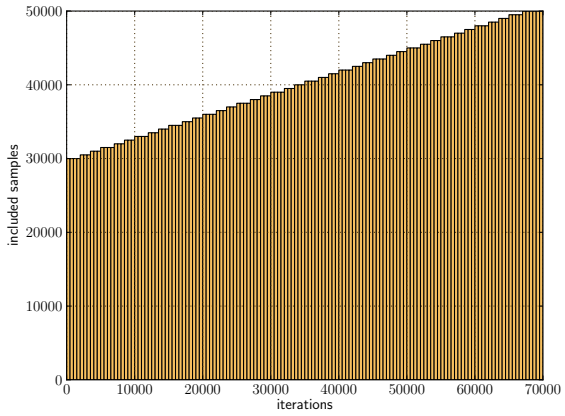
Included/excluded samples per iteration are summarized on figure 40. Except for the 20% run, all runs include samples gradually in order of increasing difficulty, in a more predictable fashion than SPL.



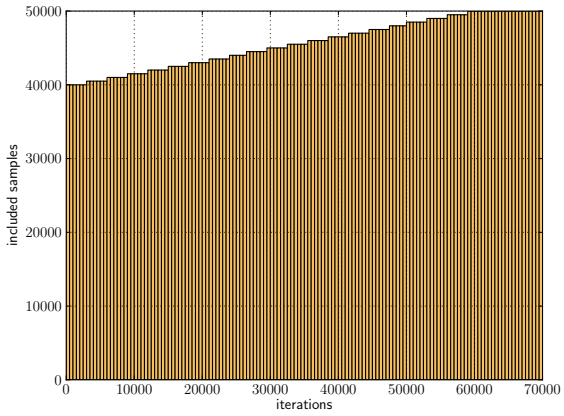
(a) start: 20%



(b) start: 40%



(c) start: 60%



(d) start: 80%

Figure 40: *Sample inclusion for 4 different start percentage points for p-SPL network*

The results are summarized in table 11. Overall, the results are unremarkable - very close to the Default network. However, it has to be pointed out that the p-SPL network saw much fewer samples during training than the Default network.

PARAMETERS	ACCURACY	LOSS
default	**0.8183**	**0.5276**
start: 80%	0.8130	0.5405
start: 60%	0.8053	0.5709
start: 40%	0.7865	0.6525
start: 20%	0.1711	6.9147

Table 11: *Result summary for best experiment from p-SPL network*

EII.5 *p-SPL-Inversed*

This experiment follows the setup described in section Eii.5. The best result was achieved with the following settings:

PARAMETER SETTINGS:

`include_perc_start` :	0.6
`perc_step` :	0.003
`warmup_epoch`:	2

Avg. accuracy results from 5 runs for the 4 different `include_perc` settings, with 2 warmup epochs, are shown on figure 41. The inversed network performs slightly better than the regular p-SPL network, but still worse than the Default network. The accuracy curves are also smoother, and exhibit less variance.

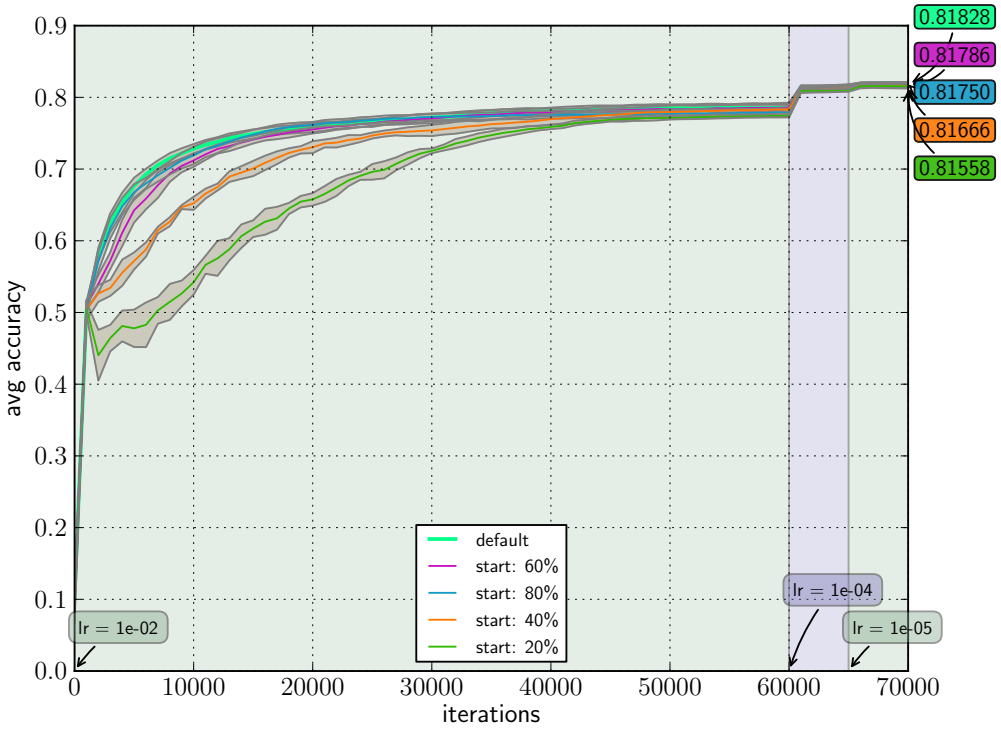


Figure 41: *Average accuracy on test set from 5 runs using p-SPL-Inversed network.*

Convergence speed is shown on figure 42. While the 4 different runs start with very uneven performance, again they equalize towards the end of training.

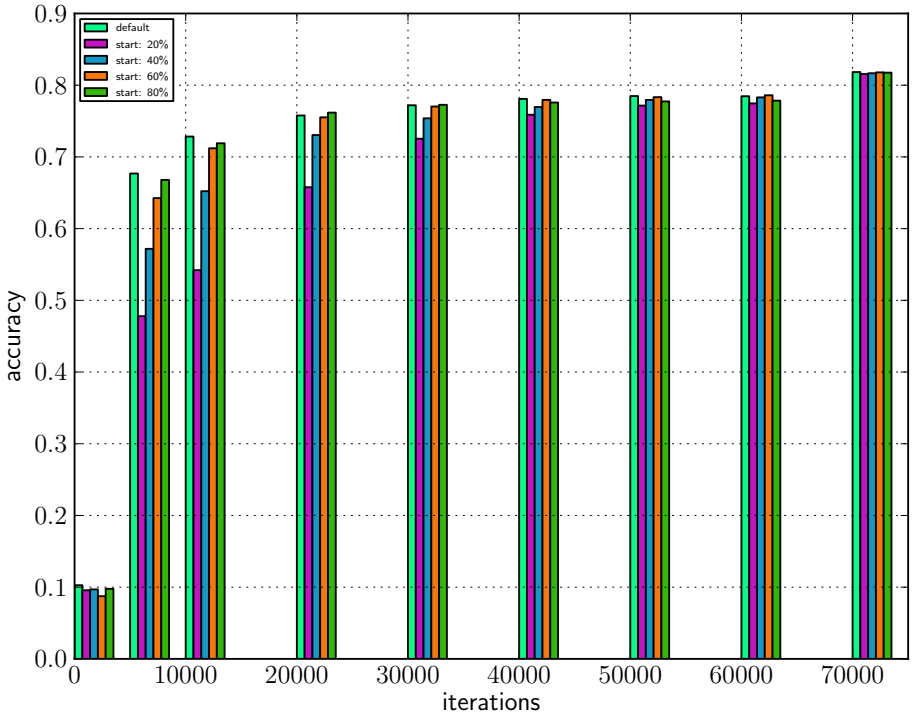
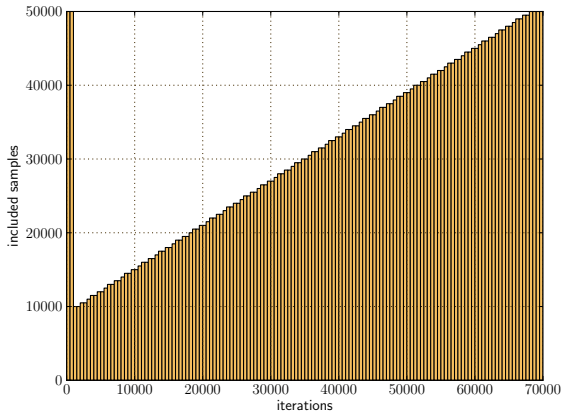
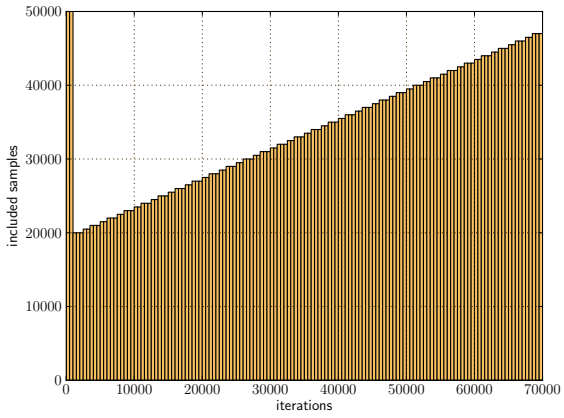


Figure 42: *Convergence speed on test set from 5 runs using p-SPL-Inversed network.*

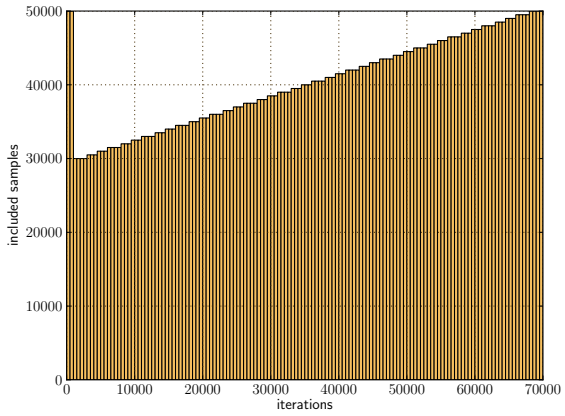
Included/excluded samples per iteration are summarized on figure 43. Again these runs exhibit the smooth sample increase characteristic for the p-SPL variant. Note that the network spends very little epochs (if any) training with all samples.



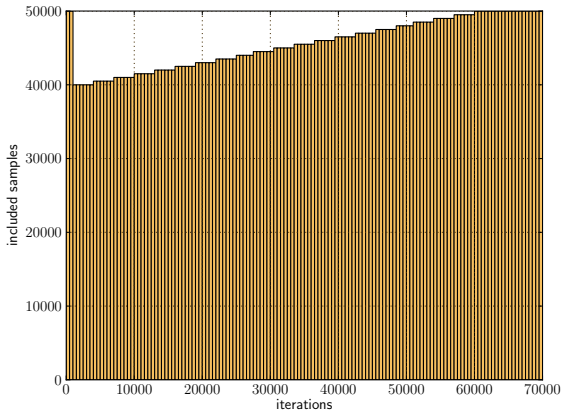
(a) start: 20%



(b) start: 40%



(c) start: 60%



(d) start: 80%

Figure 43: *Sample inclusion for 4 different start percentage points for p-SPL-Inversed network*

The results are summarized in table 12. It is interesting that the 60% start curve performs better than the 80%.

PARAMETERS	ACCURACY	LOSS
default	**0.8183**	**0.5276**
start: 60%	0.8179	0.5265
start: 80%	0.8175	0.5290
start: 40%	0.8167	0.5274
start: 20%	0.8156	0.5351

Table 12: *Result summary for p-SPL-Inversed*

EIII *Experiment Set III: SPLD Training*

EIII.1 *SPLD*

Avg. accuracy results from 5 runs for 2 different λ settings are shown on figure 44. The network slightly outperforms the Default network in one of the runs.

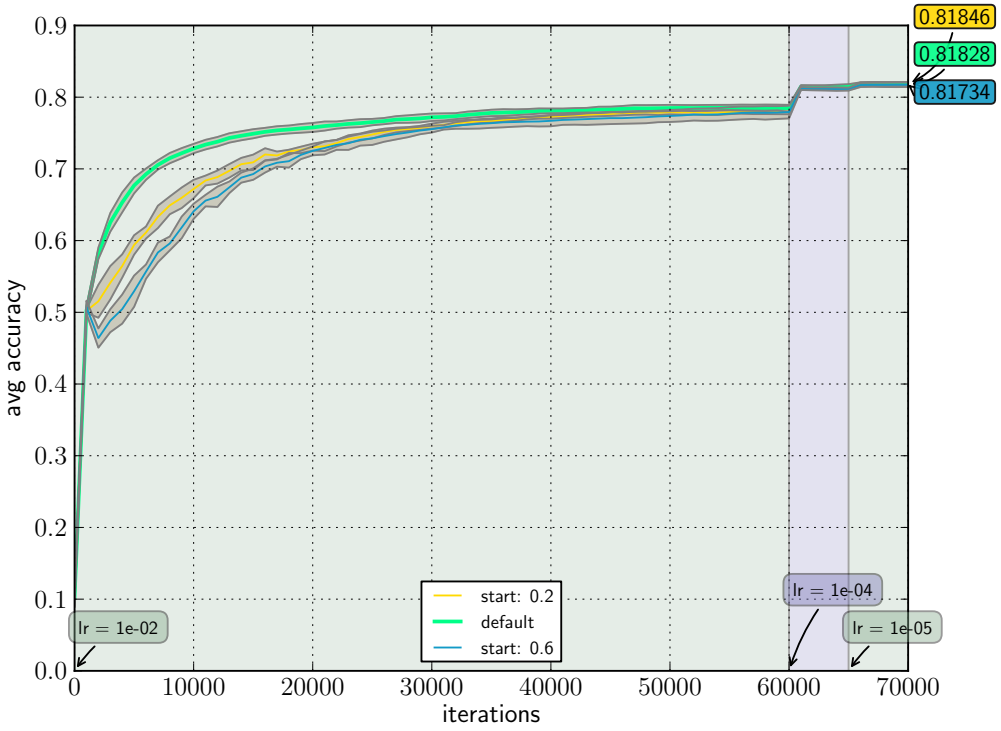


Figure 44: *Average accuracy on test set from 5 runs using SPLD network.*

Convergence speed is shown on figure 45:

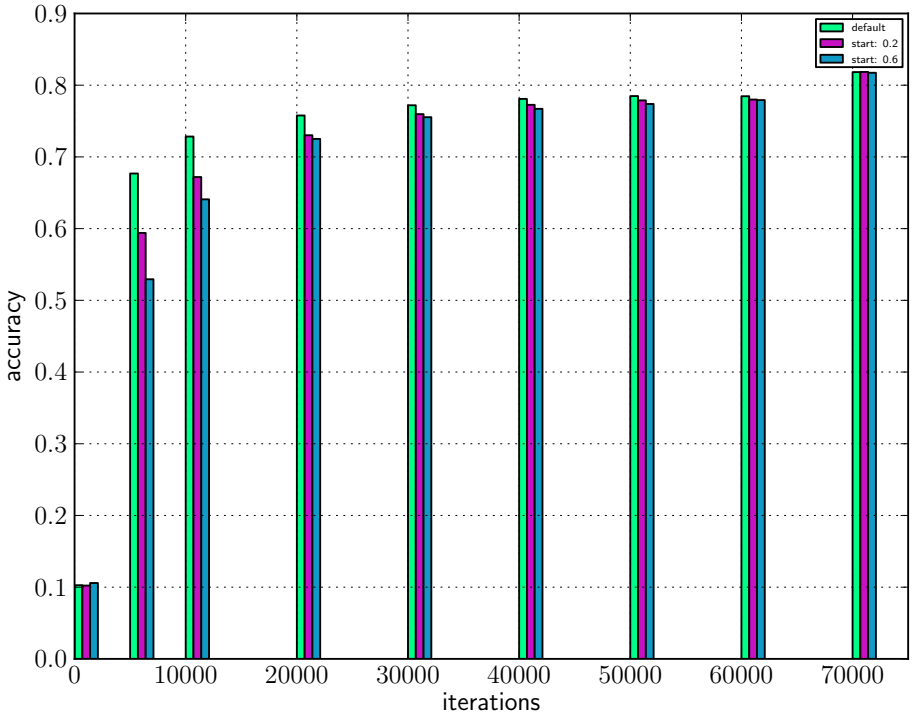
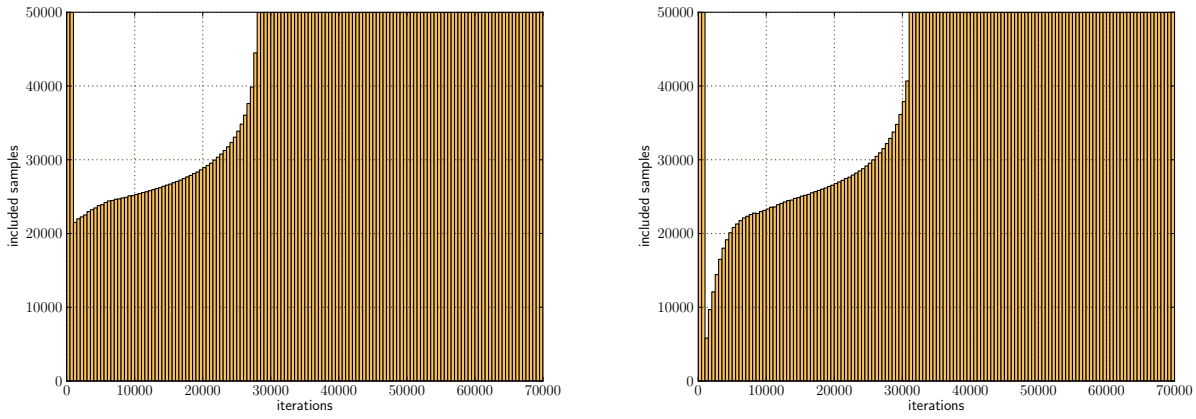


Figure 45: *Convergence speed on test set from 5 runs using SPLD network.*

Included/excluded samples per iteration are summarized on figure 46.



(a) λ: 0.2 (b) λ: 0.6

Figure 46: *Sample inclusion for 2 different start λ values for SPLD network*

The results are summarized in table 13.

PARAMETERS	ACCURACY	LOSS
start: 0.2	**0.8185**	**0.5297**
default	0.8183	0.5276
start: 0.6	0.8173	0.5275

Table 13: *Result summary for SPLD network*

EIII.2 *SPLD-Inversed*

This experiment was performed using the setup described in section Eiii.2. The best result was achieved with the following settings:

PARAMETER SETTINGS:

```
lambda:         0.8
lambda_step:    0.02
warmup_epoch:   0
```

Avg. accuracy results from 5 runs for 2 different λ settings are shown on figure 47. It is noticeable how tight the accuracy curves are again very tight, similarly to the SPL-Inversed network. The $\lambda = 0.8$ run is the best network performance recorded among all curriculum learning experiments.

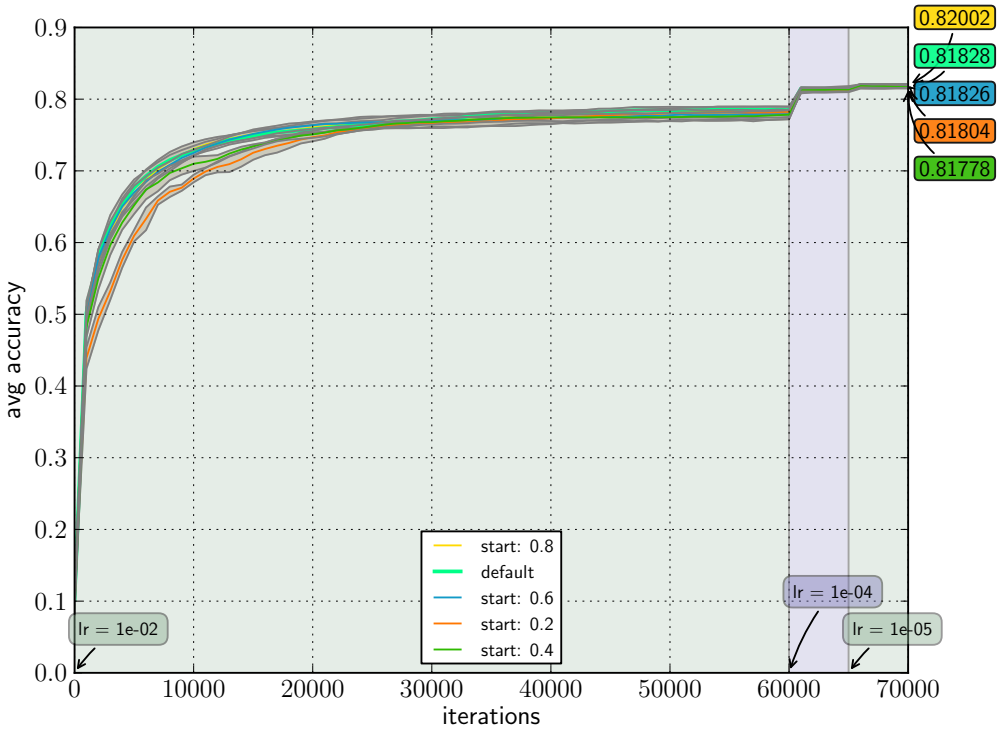


Figure 47: *Average accuracy on test set from 5 runs using SPLD-Inversed network.*

Convergence speed is shown on figure 48. It is fairly even across iterations.

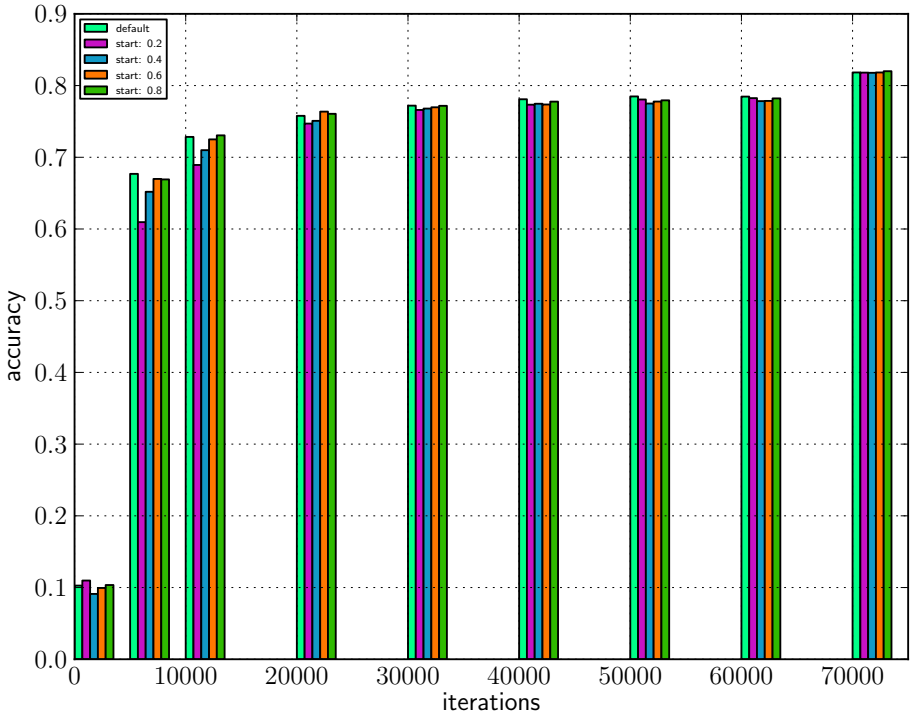
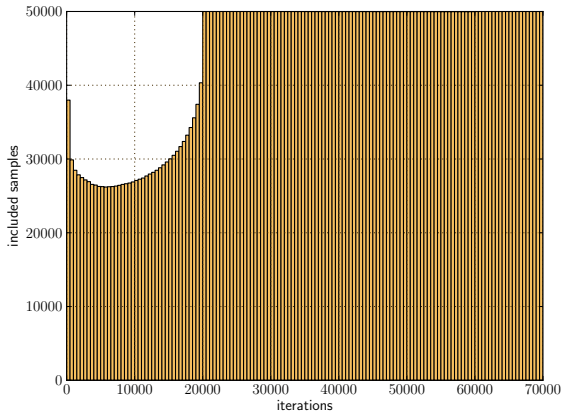
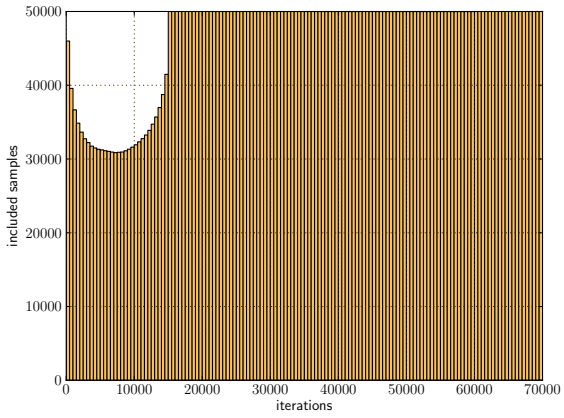


Figure 48: *Convergence speed on test set from 5 runs using SPLD-Inversed network.*

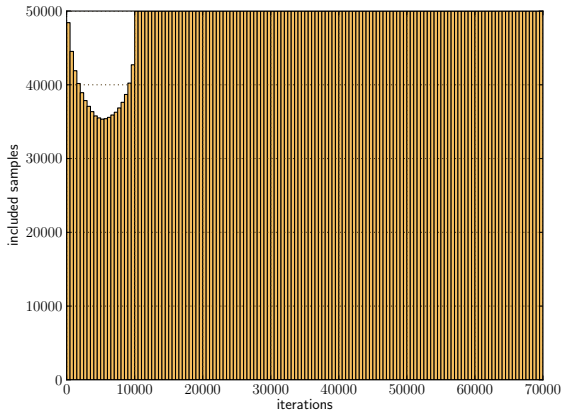
The included/excluded samples per iteration are summarized on figure 49. The overall inclusion shapes are similar to the ones from the SPL and SPL-Inversed networks.



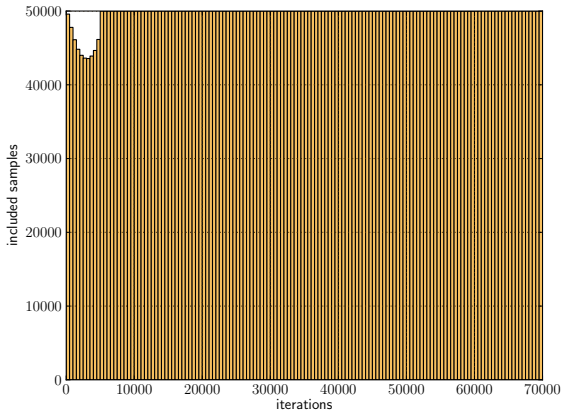
(a) start: 20%



(b) start: 40%



(c) start: 60%



(d) start: 80%

Figure 49: *Sample inclusion for 4 different start percentage points for SPLD network*

The results are summarized in table 14.

PARAMETERS	ACCURACY	LOSS
start: 0.8	**0.8200**	**0.5296**
default	0.8183	0.5276
start: 0.6	0.8183	0.5304
start: 0.2	0.8180	0.5284
start: 0.4	0.8178	0.5279

Table 14: *Result summary for SPLD-Inversed network.*

EIII.3 *p-SPLD*

This experiment was performed using the setup from section Eiii.3. The best result was achieved with the following settings:

PARAMETER SETTINGS:

include_perc_start :	0.8
perc_step :	0.015
warmup_epoch:	2

Avg. accuracy results from 5 runs for the 4 different include_perc settings, and 2 warmup epochs, are shown on figure 50. After the warmup epochs, there is a notable drop in accuracy (because the network starts removing samples). However, the 4 runs come together for a very tight finish. Performance is very close to the Default network.

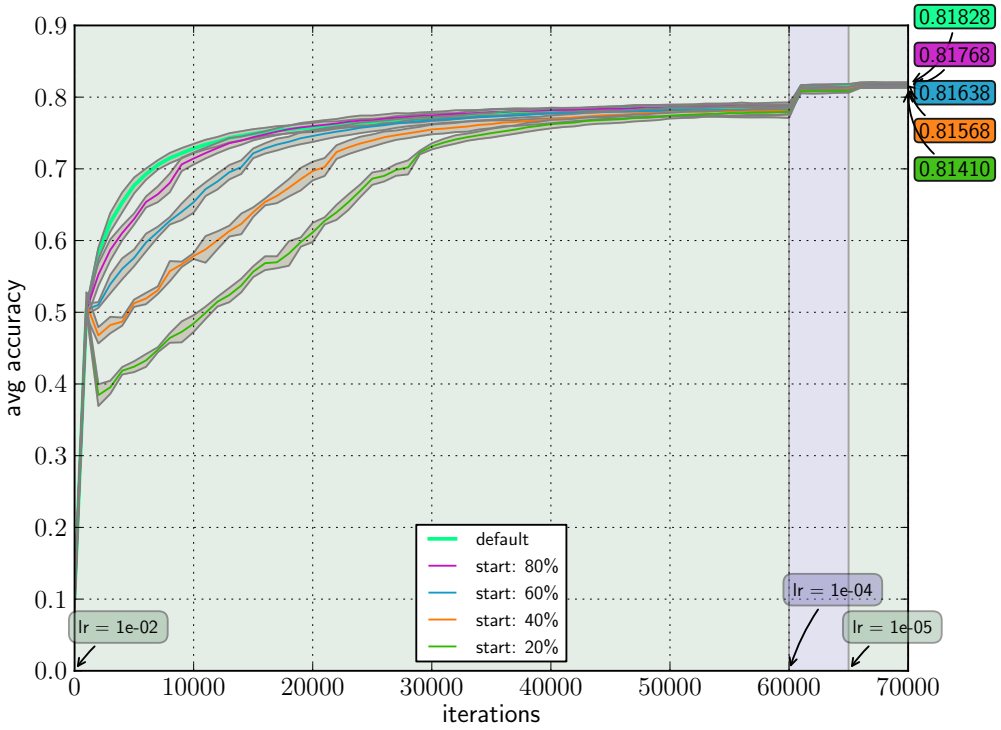


Figure 50: *Average accuracy on test set from 5 runs using p-SPLD network.*

Convergence speed is shown on figure 51. The runs start out very uneven, and equalize towards the end of training.

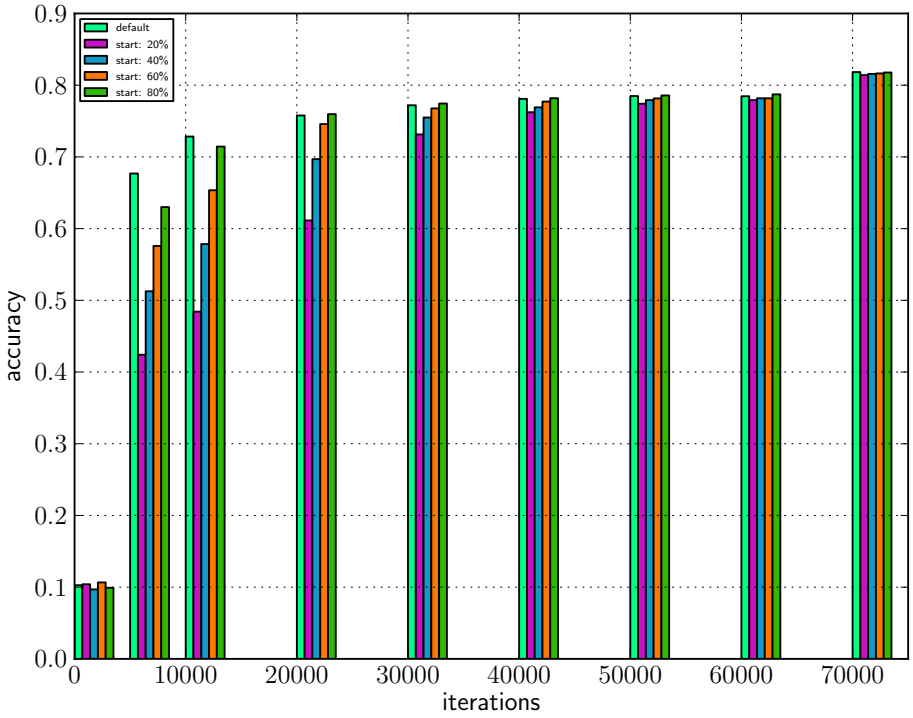
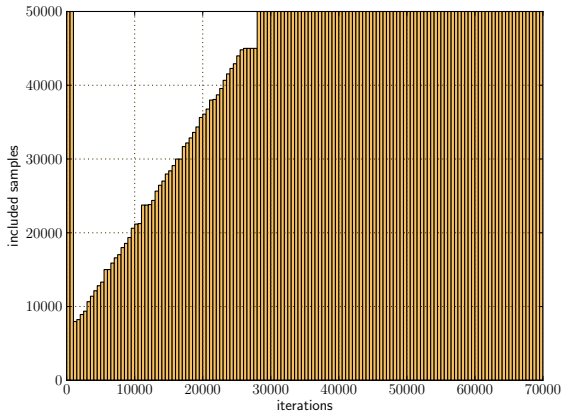
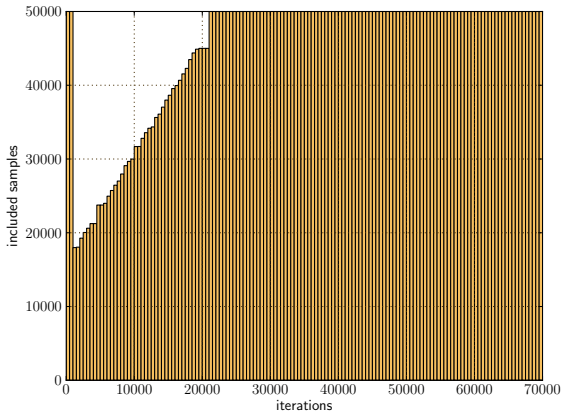


Figure 51: *Convergence speed on test set from 5 runs using p-SPLD network.*

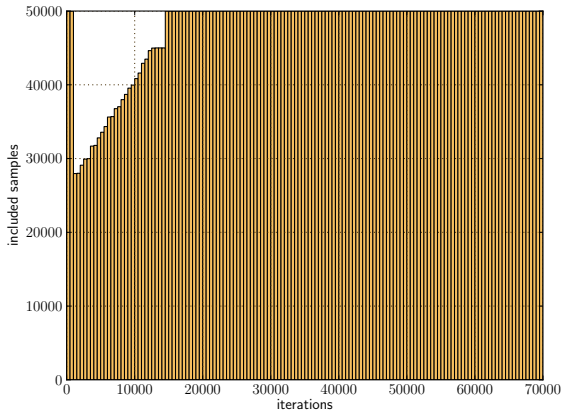
Included/excluded samples per iteration are summarized on figure 52. All samples get included before mid-point of training in this experiment.



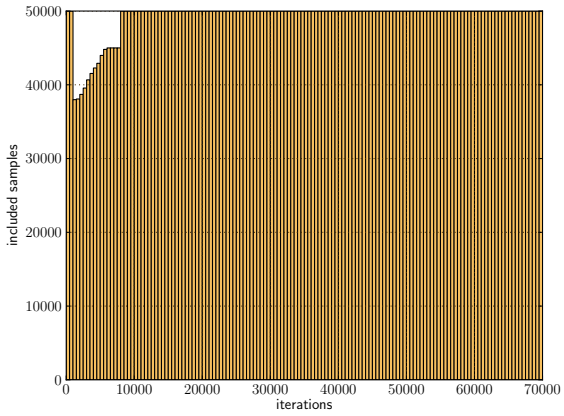
(a) start: 20%



(b) start: 40%



(c) start: 60%



(d) start: 80%

Figure 52: *Sample inclusion for 4 different start percentage points for p-SPLD network*

The results are summarized in table 15. Performance is again very close to the Default network, and otherwise unremarkable.

PARAMETERS	ACCURACY	LOSS
default	**0.8183**	**0.5276**
start: 80%	0.8177	0.5297
start: 60%	0.8164	0.5325
start: 40%	0.8157	0.5381
start: 20%	0.8141	0.5456

Table 15: *Result summary for p-SPLD network*

EIII.4 *p-SPLD Inversed*

The best experiment result was achieved with the following settings:

PARAMETER SETTINGS:

include_perc_start :	0.6
perc_step :	0.015
warmup_epoch:	0

Avg. accuracy results from 5 runs for the 4 different `include_perc` settings are shown on figure 53. It is very interesting to notice here now the 20% run starts off very slowly, but is able to catch up to the rest of the runs for a very tight finish. The results are very close to the performance of the Default network. Note that the mid-point starting points again perform better than the end-points.

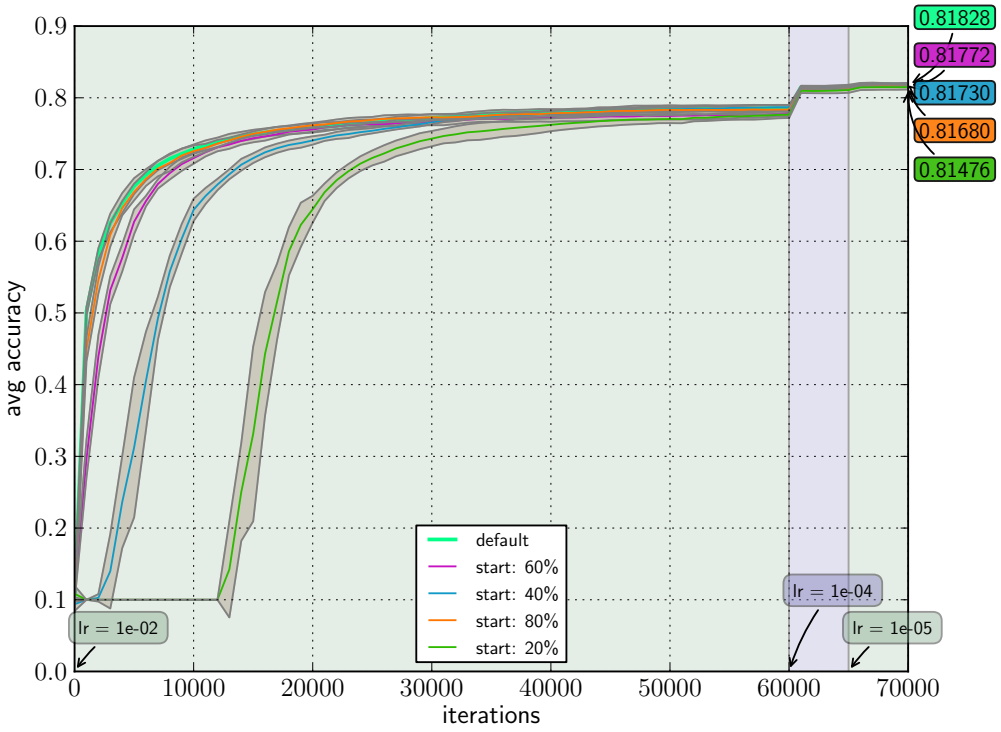


Figure 53: *Average accuracy on test set from 5 runs using p-SPLD-Inversed network.*

Convergence speed is shown on figure 54. This diagram is interesting primarily because it demonstrates how a very poor start can be overcome by the network with enough iterations.

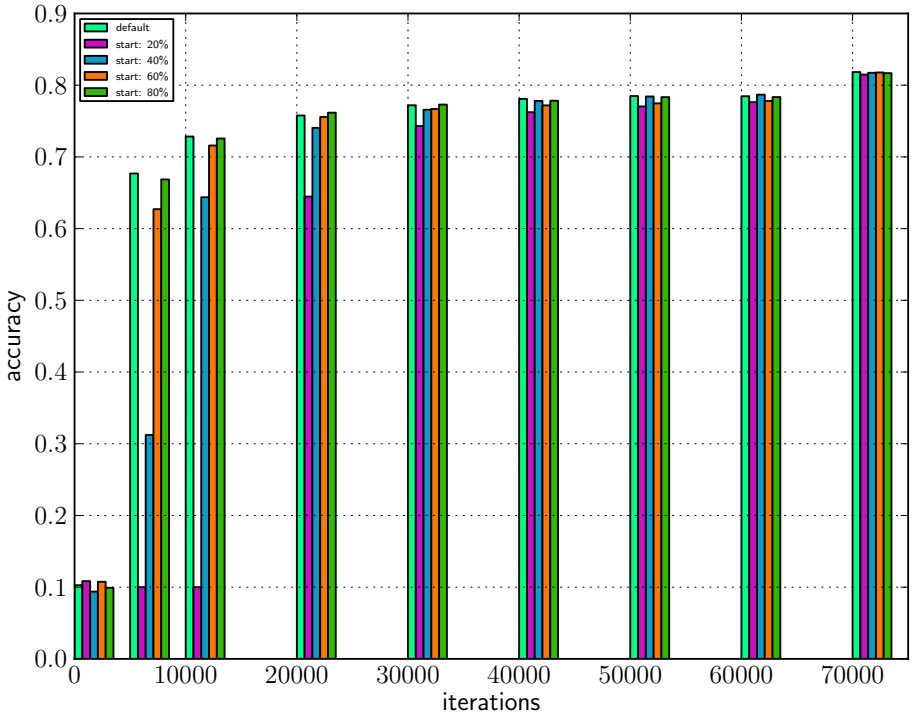
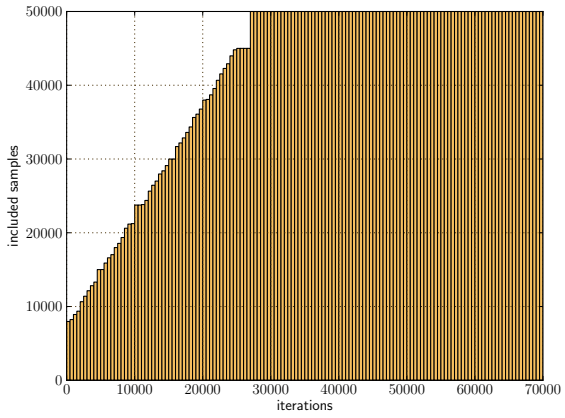
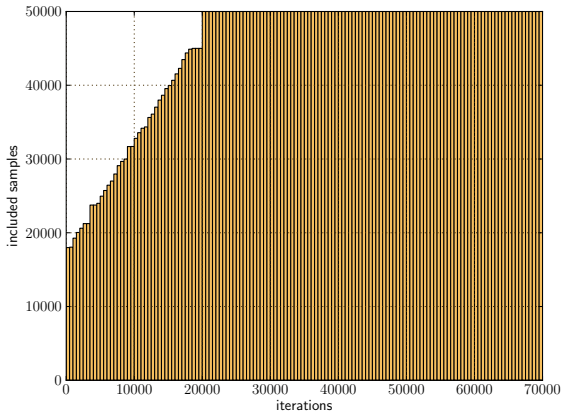


Figure 54: *Convergence speed on test set from 5 runs using p-SPLD-Inversed network.*

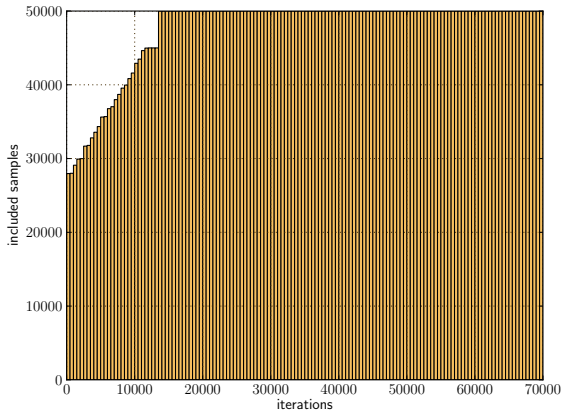
Included/excluded samples per iteration are summarized on figure 55.



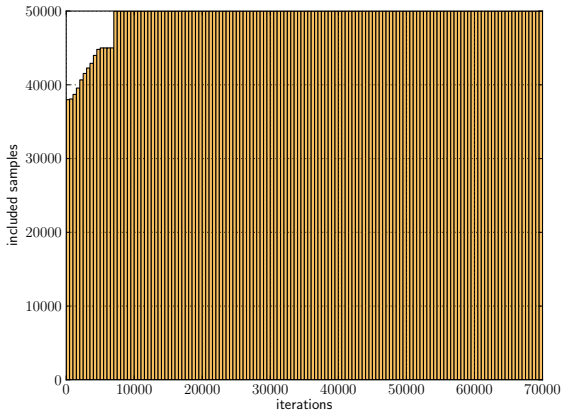
(a) start: 20%



(b) start: 40%



(c) start: 60%



(d) start: 80%

Figure 55: *Sample inclusion for 4 different start percentage points on p-SPLD-Inversed network.*

The results are summarized in table 16.

PARAMETERS	ACCURACY	LOSS
default	**0.8183**	**0.5276**
start: 60%	0.8177	0.5292
start: 40%	0.8173	0.5318
start: 80%	0.8168	0.5327
start: 20%	0.8148	0.5363

Table 16: *Result summary for p-SPLD-Inversed*

4.3 SUMMARY AND DISCUSSION

The performance results (in terms of accuracy) from all 8 explored curriculum learning variants compared to the Default network are shown in table 17. A box and whisker plot is shown on Figure 56.

METHOD	AVG ACC	ACC STD	AVG LOSS	LOSS STD
SPLD-Inv	0.82002	0.00090	0.52961	0.00272
SPL-Inv	0.81878	0.00386	0.52891	0.00843
SPLD	0.81846	0.00271	0.52965	0.00572
Default	0.81828	0.00249	0.52760	0.00310
p-SPL-Inv	0.81786	0.00330	0.52646	0.00497
p-SPLD-Inv	0.81772	0.00178	0.52919	0.00366
p-SPLD	0.81768	0.00262	0.52965	0.00309
SPL	0.81698	0.00212	0.53123	0.00608
p-SPL	0.81300	0.00328	0.54054	0.00418

Table 17: *Avg. accuracy and avg. loss result summary from best experiment results for all explored methods. The results are sorted by avg. accuracy in decreasing order.*

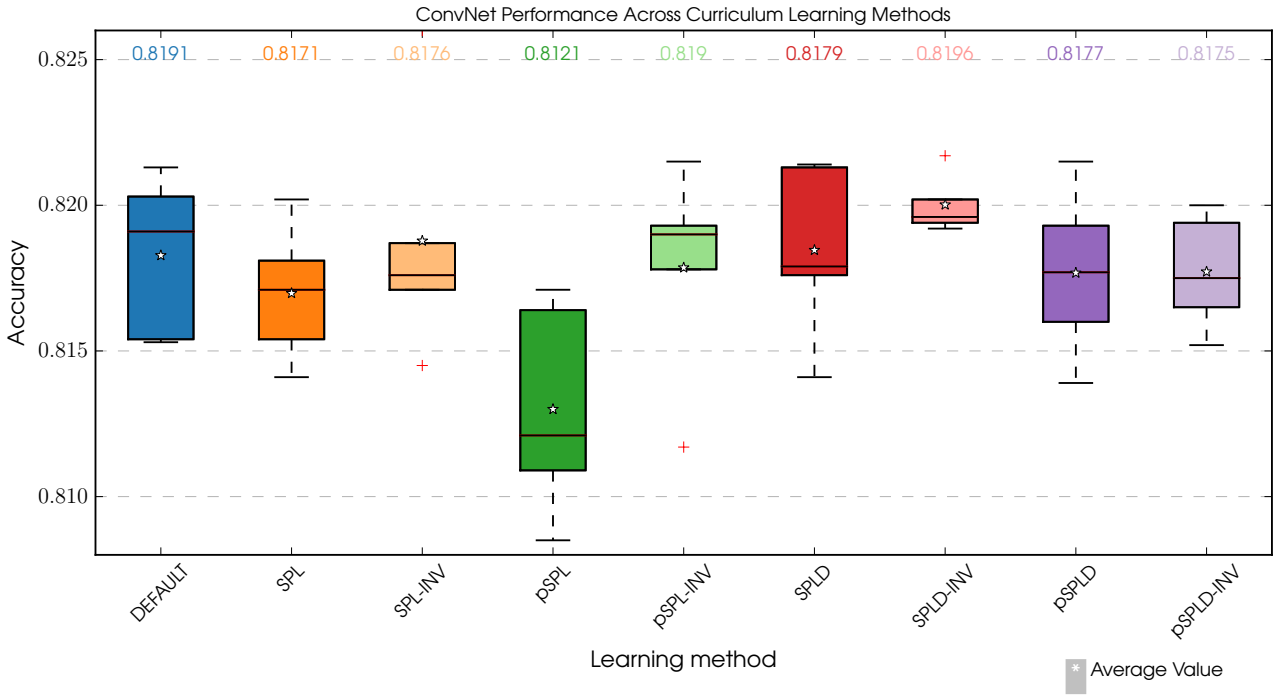


Figure 56: *A box and whisker plot of all explored methods at iteration* 70,000. *The median is represented by the horizontal line inside the box. The borders of the box are set at the 25th and the 75th percentiles. The whiskers extend to the most extreme data point within* $1.5 * (75\% - 25\%)$ *data range*

First we have to note that for all performance comparisons, the networks perform very similarly. We are splitting hairs with the performance measure, but the most important outcome is that the Inversed versions of the SPL and SPLD networks overall performed better than the regular versions. This is interesting, because it implies that the assumption that the progression from easy to difficult samples is beneficial may not hold for ConvNets trained on image datasets. Based on the experiments, the opposite progression performs very similarly, and slightly better. The order of sample presentation may not be that significant, compared to keeping the level of difficulty of accepted samples even across an epoch or batch. Also of note is how tight the performance variance of the Inversed networks is. More experiments on more data sets are needed to confirm these finding. If we step back to the hypothesis defined in section 3.7.2, based on experiment data, the following conclusions can be made:

H1 *The accuracy decline of a ConvNet trained with a certain percentage x_p of training samples removed is less than* 0.5%.

This hypothesis is true for removing up to 40% of the training samples. This finding is significant because large gains in training speed can be made by dropping a large set of the samples with relatively negligible effect on performance. It also hints that after a certain point, adding more samples to a data set does not correlate to significantly better performance. There is a point at which the performance improvement per added samples stalls. More experiments are needed to determine how this point can be figured out based on the particular details and nature of each data set.

H2 *A network trained with easiest samples removed performs worse than a network trained with the most difficult samples removed*

The results for this experiment were a surprise, since the opposite statement was shown to be true: a network trained with the most difficult examples removed performed notably worse than the one trained with easiest samples removed.

H3 *A network trained with the most difficult samples removed performs better than a network trained with random samples removed, and within* 0.2% *margin of the default network*

The hypothesis, as stated, is not true: because of the unexpected results for hypothesis H3. The hypothesis needs to be flipped: a network trained on the easiest samples removed performs better than the random removal network, and within 0.2% margin of the Default network for up to 40% sample removal.)

H4 *A network trained using SPL performs better than the default network*

This hypothesis is true for the SPL-Inversed network, by a very small margin. All other variants perform worse.

H5 *A network trained using SPLD performs better than the default setup*

This hypothesis is true for the SPLD and SPLD-Inversed networks, again by a very small margin.

H6 *A network trained using SPLD performs better than the same network trained using SPL*

This hypothesis is true for all SPLD variants except p-SPLD-Inversed, which seems to perform slightly worse than p-SPL-Inversed.

Part V

CONCLUSION

This chapter summarizes the most important findings of the project and their implications. It also provides suggestions and guidance for possible future work directions.

5 CONCLUSION

5.1 SUMMARY OF FINDINGS

The project investigated two variants of SPL, two variants of SPLD, and their inversed counterparts – networks who prefer difficult samples first. Their performance was compared to that of a ConvNet with exactly the same structure and learning parameters, but trained with the customary random sample presentation. All networks performed very similarly, within less than a 1% avg. accuracy margin. Within this margin, the following observations can be made:

- Presenting the samples in order of decreasing difficulty, i.e. starting with the most difficult first (as done in the Inversed networks), generally has a more positive effect on performance compared to presenting the samples in increasing order of difficulty (Fig. 37 and Fig. 56). This effect is more pronounced for networks trained using SPL than SPLD. Inversed networks also tend to exhibit lower variance in accuracy score curves than the Default and regular SPL and SPLD networks. This finding is interesting because it demonstrates that while the order of sample presentation does affect the network's ability to learn, the direction of the order (ascending or descending difficulty) may not be fixed, and is likely data set, learning task and algorithm specific. The ConvNets trained in the experiments seemed to derive more value from the difficult examples, which is possibly due to a fine balance between the network's ability to learn from difficult examples (e.g. they do not lie too far away from the regular samples as to be outliers that confuse the network), and their ability to provide the largest gradient adjustment per sample in the backpropagation phase.
- In all eight instances, when training the networks for $70,000$ iterations, there was no clear improvement in achieved accuracy over the Default network which used random sample presentation (Fig. 56). The SPLD-Inversed network achieved best performance, with a median difference of 0.05% over the Default Network. The SPLD-Inversed network also registered the highest average accuracy score of 82.002%, a difference of 0.174% from the Default network's average score. This approximately equates to 17 more images (out of the $10,000$ test images) classified correctly by the SPLD-Inversed network. It has to be noted, however, that the SPL and SPLD networks saw fewer samples

during the training cycle, so if accuracy is measured with number of examined samples taken into account, then the performance of the SPL and SPLD networks would be considered better. Finally, there was a notable performance difference between the p-SPL and p-SPL-Inversed networks from the quick train experiment Eii.3, likely due to an earlier stop in network training. Again, the p-SPL-Inversed version of the network performed slightly better than the default network, and in this instance, notably better than the regular p-SPL network.

- Warm-up iterations had no notable effect on SPL and SPLD training. The networks who start without warm-up iterations are able to compensate during the course of training.

- p-SPL and p-SPLD networks perform worse than the regular SPL and SPLD networks. This points out the fact that keeping difficulty levels consistent across an entire epoch (or in general, for a longer period during training) is important.

The "teaching value" potential of data samples according to their difficulty level was also investigated, by performing step-wise sample removal of the easiest, most difficult, or random samples from the CIFAR-10 training data. The results are summarized below.

- A large number of samples in rich training sets may be redundant. Certain percentage of samples can be removed randomly without having a noticeable effect on final performance, thus speeding up the training process significantly. For CIFAR-10, removing up to 10% of the easiest samples made no difference in network performance.

- Removing the difficult samples hurts network training the most (Fig. 29). Pruning easy samples is the best option: removing up to 40% of the easiest samples from CIFAR-10 reduced accuracy by only $\approx 1\%$. Removing 40% of the most difficult samples reduced accuracy by $\approx 6\%$. Finally, removing 40% samples randomly reduced accuracy by $\approx 3\%$.

- Removing the easiest samples first ("Ranked Removal: Easy First) does yield better performance in all instances over the random and difficult first removals, up to 60%. In the exclude 80% case, "Ranked Removal: Easy First" performs 31% worse than random removal, as a result of a rapid deterioration (Fig. 29). This is possibly due to the fact that the sample removal % crossed a data-set specific threshold where "Ranked Removal: Easy First" is out of enough representative (easy) samples, and left with the most difficult, divergent samples from which it can no longer establish a good class representation. It is possible that such inflection point exists in each data set: a threshold

up to which "Ranked Removal: Easy First" performs better, and after which random removal performs better. This threshold is probably significant, and can be used to establish a heuristic on what percentage of the data can be dismissed from a data set randomly (or "easiest first") without losing performance.

- As it was demonstrated in section 3.3, the set of top 20% easiest samples stays fairly consistent during training iterations, so they can be identified early with relatively high confidence, and a percentage of them potentially removed with minimal effect on training.

5.2 FUTURE WORK

The findings in this report raise interesting questions about the ideal order in which samples should be arranged in a curriculum for ConvNets. More experiments on different data sets are needed to replicate the results outlined in this work. Further, trying the removal experiments on more data sets may help pinpoint the validity of the point of inflection determined by the performance of the easiest-first removal set falling below the random removal set, and whether a percentage close to this point can be used as a determining factor of when a data set becomes "saturated", when adding further examples will have little or no impact on classification accuracy.

In the performed experiments, the difference in performance between curriculum learning variants was minor. This could be due to the fact that the network was given a long time (number of epochs) to learn, or that the network was already so capable that using different methods of sample presentation made no major difference in performance. Removing layers from the network architecture could potentially "weaken" the network and better illustrate the effect of curriculum learning techniques.

Finally, other metrics of difficulty can be explored in order to shape the curriculum. Currently, difficulty is only measured as the difference between the actual label distribution (1 for true label) and the predicted label probability. This does not take into effect for example, if the network strongly believes in a wrong label, or the probability is more equally distributed among the possible classes. In the first scenario, it is more unlikely that the network will adjust from a strong belief in the wrong label to the correct label, while in the second scenario, it is more likely that the correct label may yet be reached. Such subtle differences in the prediction softmax vector can be explored in order to provide a finer metric of difficulty for each sample.

BIBLIOGRAPHY

[1] D. Adams, *The Hitchhikers Guide to the Galaxy*. Del Rey, 1979.

[2] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., Curran Associates, Inc., 2010, pp. 1189–1197. [Online]. Available: http://papers.nips.cc/paper/3923-self-paced-learning-for-latent-variable-models.pdf.

[3] 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*, 2014, pp. 2078–2086.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for visual recognition," in *CVPR workshop of DeepVision*, vol. 1, 2014, p. 3.

[6] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: delving deep into convolutional nets," *arXiv preprint arXiv:1405.3531*, 2014.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," DTIC Document, Tech. Rep., 1985.

[8] Y. Le Cun, "A learning scheme for asymmetric threshold networks," *Proceedings of Cognitiva*, vol. 85, pp. 599–604, 1985.

[9] 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)*, 2010, pp. 807–814.

[10] 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*, vol. 15, no. 1, pp. 1929–1958, 2014.

[11] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09, Montreal, Quebec, Canada: ACM, 2009, pp. 41–48, ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553380.

[12] J. A. Anderson and E. Rosenfeld, *Neurocomputing*. MIT press, 1993, vol. 2.

[13] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[14] M. L. Minsky, *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[15] D. Hebb, "The organization of behavior; a neuropsychological theory.," 1949.

[16] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[17] O. G. Selfridge, "Pandemonium: a paradigm for learning in mechanisation of thought processes," 1958.

[18] B. WIDROW, M. E. HOFF, *et al.*, "Adaptive switching circuits.," 1960.

[19] R. Winter and B. Widrow, "Madaline rule ii: a training algorithm for neural networks," in *Neural Networks, 1988., IEEE International Conference on*, IEEE, 1988, pp. 401–408.

[20] M. Minsky and P. Seymour, "Perceptrons.," 1969.

[21] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral sciences," 1974.

[22] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[23] K. Fukushima, "Cognitron: a self-organizing multilayered neural network," *Biological cybernetics*, vol. 20, no. 3-4, pp. 121–136, 1975.

[24] ——, "Neocognitron: a hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, Citeseer, 1990.

[27] (). Cs231n: convolutional neural networks for visual recognition, [Online]. Available: http://cs231n.github.io/convolutional-networks/ (visited on 09/01/2015).

[28] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[29] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, p. 310, 1995.

[30] B. F. Skinner, "Reinforcement today.," *American Psychologist*, vol. 13, no. 3, p. 94, 1958.

[31] G. B. Peterson, "A day of great illumination: bf skinner's discovery of shaping.," *Journal of the Experimental Analysis of Behavior*, vol. 82, no. 3, p. 317, 2004.

[32] J. L. Elman, "Learning and development in neural networks: the importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[33] K. A. Krueger and P. Dayan, "Flexible shaping: how learning in small steps helps," *Cognition*, vol. 110, no. 3, pp. 380–394, 2009.

[34] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of artificial intelligence research*, 1996.

[35] L. E. Atlas, D. A. Cohn, and R. E. Ladner, "Training connectionist networks with queries and selective sampling," in *Advances in neural information processing systems*, 1990, pp. 566–573.

[36] S. D. Whitehead and D. H. Ballard, *A study of cooperative mechanisms for faster reinforcement learning*. University of Rochester, Department of Computer Science Rochester, NY, 1991.

[37] S. B. Thrun and K. Möller, "Active exploration in dynamic environments," in *Advances in neural information processing systems*, 1992, pp. 531–538.

[38] A. Linden and F. Weber, "Implementing inner drive by competence reflection," in *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior, MIT Press, Cambridge, MA*, 1993.

[39] J. Storck, S. Hochreiter, and J. Schmidhuber, "Reinforcement driven information acquisition in non-deterministic environments," in *Proceedings of the International Conference on Artificial Neural Networks, Paris*, Citeseer, vol. 2, 1995, pp. 159–164.

[40] Y. Yang, H. T. Shen, Z. Ma, Z. Huang, and X. Zhou, "L2, 1-norm regularized discriminative feature selection for unsupervised learning," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Citeseer, vol. 22, 2011, p. 1589.

[41] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Computer Science Department, University of Toronto, Tech. Rep*, vol. 1, no. 4, p. 7, 2009.

[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[43] (). Protocol buffers - google's data interchange format, [Online]. Available: https://developers.google.com/protocol-buffers/.

[44] A. Lapedriza, H. Pirsiavash, Z. Bylinskii, and A. Torralba, "Are all training examples equally valuable?" *arXiv preprint arXiv:1311.6510*, 2013.

Part VI

APPENDIX

APPENDIX

A.1 DEFAULT CONVNET ARCHITECTURE

Listing 1: CAFFE network definition for Default Network

```
name: "CIFAR10_full"
layers {
  name: "cifar"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "examples/cifar10/cifar10_train_leveldb"
    batch_size: 100
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  include: { phase: TRAIN }
}
layers {
  name: "cifar"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "examples/cifar10/cifar10_test_leveldb"
    batch_size: 100
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  include: { phase: TEST }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
```

```
    weight_filler {
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "pool1"
  top: "pool1"
}
layers {
  name: "norm1"
  type: LRN
  bottom: "pool1"
  top: "norm1"
  lrn_param {
    norm_region: WITHIN_CHANNEL
    local_size: 3
    alpha: 5e-05
    beta: 0.75
  }
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "norm1"
  top: "conv2"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
```

```
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu2"
  type: RELU
  bottom: "conv2"
  top: "conv2"
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "norm2"
  type: LRN
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    norm_region: WITHIN_CHANNEL
    local_size: 3
    alpha: 5e-05
    beta: 0.75
  }
}
layers {
  name: "conv3"
  type: CONVOLUTION
  bottom: "norm2"
  top: "conv3"
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
```

```
    }
  }
}
layers {
  name: "relu3"
  type: RELU
  bottom: "conv3"
  top: "conv3"
}
layers {
  name: "pool3"
  type: POOLING
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool3"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 250
  weight_decay: 0
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "accuracy"
  type: ACCURACY
  bottom: "ip1"
  bottom: "label"
  top: "accuracy"
  include: { phase: TEST }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip1"
```

```
  bottom: "label"
  top: "loss"
}
```

A.2 QUICK TRAINING CONVNET ARCHITECTURE

Listing 2: CAFFE network definition for Experiment Set II: %-SPL Quick Training Network

```
name: "CIFAR10_quick"
layers {
  name: "cifar"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "examples/cifar10/cifar10_train_leveldb"
    batch_size: 100
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  include: { phase: TRAIN }
}
layers {
  name: "cifar"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "examples/cifar10/cifar10_test_leveldb"
    batch_size: 100
  }
  transform_param {
    mean_file: "examples/cifar10/mean.binaryproto"
  }
  include: { phase: TEST }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
```

```
      type: "gaussian"
      std: 0.0001
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "pool1"
  top: "pool1"
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 32
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu2"
  type: RELU
  bottom: "conv2"
  top: "conv2"
}
```

```
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "conv3"
  type: CONVOLUTION
  bottom: "pool2"
  top: "conv3"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 64
    pad: 2
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu3"
  type: RELU
  bottom: "conv3"
  top: "conv3"
}
layers {
  name: "pool3"
  type: POOLING
  bottom: "conv3"
  top: "pool3"
  pooling_param {
    pool: AVE
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
```

```
  bottom: "pool3"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 64
    weight_filler {
      type: "gaussian"
      std: 0.1
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "gaussian"
      std: 0.1
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "accuracy"
  type: ACCURACY
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include: { phase: TEST }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```