

Improvements in Learning Capacities of Convolutional Neural Networks

A thesis submitted by

Bishshoy Das

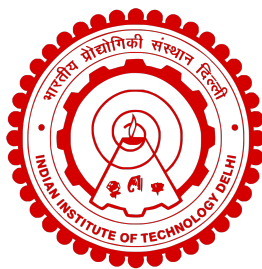
2016EEZ8482

under the guidance of

Prof. Sumantra Dutta Roy

*in partial fulfillment of the requirements
for the degree of*

Doctor of Philosophy



Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY DELHI

December 2022

THESIS CERTIFICATE

This is to certify that the thesis entitled “*Improvements in Learning Capacities of Convolutional Neural Networks*” being submitted by Bishshoy Das to the Department of Electrical Engineering, Indian Institute of Technology Delhi, for the award of the degree of Doctor of Philosophy is the record of the bonafide research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted either in part or in full to any other university or institute for the award of any degree or diploma.

Prof. Sumantra Dutta Roy

Professor, Dept. of Electrical Engineering

Indian Institute of Technology Delhi

Hauz Khas, New Delhi, Delhi - 110016

DECLARATION

I hereby certify that :—

1. The work contained in the thesis is original and has been done by me under the guidance of my supervisor.
2. The work has not been submitted to any other institute for any other degree or diploma.
3. I have followed the guidelines provided by the institute in preparing the thesis.
4. I have conformed to ethical norms and guidelines while writing the thesis.
5. Whenever I have used materials (data, models, figures, text) from other sources, I have given due credit by citing them in the text of the thesis, and giving their details in the references, and taken permission from the copyright owners of the sources whenever necessary.

Bishshoy Das

Dept. of Electrical Engineering

Indian Institute of Technology Delhi

Hauz Khas, New Delhi, Delhi - 110016

ACKNOWLEDGEMENT

I would like to thank Professor Sumantra Dutta Roy for his assistance and guidance during my PhD. I learned a great deal from him in the classroom, but I learned much more from our interactions and talks outside of the classroom.

I would also want to thank Professors Shiv Dutt Joshi and Brejesh Lall for their insightful recommendations and methodical analyses, which helped me advance my thoughts when I was stuck.

I would also want to thank Professor Manas Kamal Bhuyan of Indian Institute of Technology Guwahati, who inspired and encouraged me to pursue an academic career.

Finally, I would like to thank my colleagues and friends Milton Mondal, Martin Cheerangal, Gopala Rao D, Siranjeevi Gurumani, Karthikeya G S, Himanshu Padole, and Santosh Kumar Gedela for their intellectual and emotional support, without which I would not have been able to endure my long PhD trip. During the course of my PhD, I also served as a mentor to numerous M. Tech students, who offered me the opportunity to instruct them on fundamental ideas and help them through their projects. Regarding this, I would like to thank Srijeet Chatterjee, Hareesh Kumawat, Sakshi Agrawal, Asmita Nandkumar Patil, Abhinav Gaur, and Panchal Sanket Hitendrakumar for their stimulating discussions.

*To my beloved mother
and my beloved father*

ABSTRACT

In this thesis, we explore different methods of improving the utilization of a convolutional neural network’s learning capacity. We define what we mean by the learning capacity of a neural network, since no standard definition of learning capacity exists in the literature hitherto. Based on our definition of learning capacity, we formulate two scenarios in which the learning capacity of a CNN can get significantly affected, (i) Intra-network learning and (ii) Inter-network learning. In both cases, we explore in-depth as to what is the cause behind the lack of utilization of learning capacities. Then, we formulate targeted solutions to solve for each of the discovered lacunae. For example in our explorations with intra-network learning, we discover that in non-BatchNormalized networks, some filters or even some layers completely goes to zero during training, a phenomena which we call zeroing out. This leads to a drop in the utilization of the network’s trainable parameters and hence an under-utilization of learning capacities. We address this phenomena by proposing **PowerGrad Transform** that modifies the gradient before the backward pass and with that we mitigate the zeroing out phenomena and improve final accuracies of non-BatchNormalized convolutional neural networks. We also apply PowerGrad Transform to BatchNormalized networks. We observe that PowerGrad transform can help the network learn parameter that are outside of the scope of standard gradient descent. Even with high loss values, networks can achieve higher final convergence accuracies in both training and test datasets. This leads us to believe that transforming the gradient through PowerGrad transform leads the network to better utilize its learning capacity.

In our explorations of inter-network learning, we see that when two or more models are trained from random initialization, often the final converged weights have significant overlaps leading to a lower utilization of pool learning capacity. Ensembles of such networks do not improve the performance (or marginally improves) the performance over any one single network in the same pool. This points to the observation that the total learning capacity of the pool as a whole is severely under-utilized. In order to remedy this, we propose adversarial training through **Feature Difference Losses**, a method in which all feature vectors are shared amongst worker networks in a pool, and each worker adversarially pushes each other to learn different feature maps, leading to a forced outcome where at convergence each network arrives at vastly different optima on the loss landscape. This in turn improves ensemble performance as it leads to a diversity in the networks' features, which directly translates to an increased utilization of the pool's learning capacity.

TABLE OF CONTENTS

1	Introduction	1
1.1	Learning Capacity	2
1.2	Motivation	6
1.3	Contributions	7
1.4	Objective Formulation	8
1.4.1	Inter-network learning capacity	9
1.4.2	Intra-network learning capacity	10
1.5	Organization of the Thesis	12
2	Mathematical Preliminaries	14
2.1	Convolutional layers	14
2.2	Pooling layers	17
2.3	Residual connections	18
2.4	Activation layers	19
2.5	Normalization layers	20
2.6	Fully-connected layers	21
2.7	Softmax layer	22
2.8	Loss functions	22
2.9	Hyperparameters & Regularization	23
2.10	Backpropagation & Optimization	24
3	Related Works	29

4	Knowledge Diversification	35
4.1	Motivation	35
4.2	Introduction	37
4.3	Method	38
4.3.1	Feature Difference Loss functions	40
4.3.2	Training Phases	42
4.3.3	Phase 0	42
4.3.4	Phase 1 & 2	43
4.3.5	Phase 3	44
4.3.6	Phase 4	46
4.4	Experiments	46
4.5	Ablation studies	52
4.6	Discussions	54
5	PowerGrad Transform	57
5.1	Motivation	57
5.2	Introduction	58
5.3	Method	63
5.4	Analysis	65
5.4.1	Properties of PGT	65
5.4.2	Effect on Logits	68
5.4.3	Effect on Weight Gradients	70
5.4.4	Effect on Class Separation	72
5.5	Experiments	73
5.5.1	BatchNormalized networks	73
5.5.2	Non-BatchNormalized networks	75
5.5.3	Grid search and Effects on Loss	82
5.6	Ablation studies	84

5.6	Implementation	85
5.7	Discussions	86
6	Conclusion	89
6.1	Knowledge Diversification	90
6.2	PowerGrad Transform	92
7	List of Publications	95
8	References	97

LIST OF FIGURES

1	Problem 1: Improving inter-network learning capacity	9
2	Problem 2: Improving intra-network learning capacity	11
3	Architecture of the ensemble head network	38
4	Architecture of four networks trained with FDL	39
5	Training pipeline of two networks trained with FDL	42
6	Networks used for FDL experimentation.	46
7	Filters vs Accuracy of simple ConvNet with and w/o FDL	47
8	Multi-network FDL ensemble accuracies	49
9	Several plots for FDL ablation studies	52
10	Feature map differences with and without FDL	53
11	Gradient alteration types	59
12	Effect of PGT on probability distributions	63
13	Analysis of PGT at the fully connected layer	70
14	PGT Training/Test accuracy vs. baseline for ResNet-50	73
15	ResNet-18 architecture with layer indices	76
16	The Zeroing Out phenomena	77
17	Norm vs. Iteration plot of layer 5 of non-BN ResNet-18	79
18	Experiment demonstrating the efficacy of PGT	80
19	Effects of PGT: various statistical measures	83
20	PGT Grid search for various values of α	84
21	Python code of PowerGrad Transform based on PyTorch	86

LIST OF TABLES

1	FDL: Experiments on MNIST	47
2	Comparisons of ensemble methods vs. FDL on CIFAR datasets . . .	48
3	Comparisons of ensemble methods vs. FDL on ImageNet-1K datasets	48
4	Multi-network FDL ensembles with CIFAR-10	50
5	Multi-network FDL ensembles with CIFAR-100	50
6	FDL experiment hyperparameters	51
7	Ablation studies using FDL (1)	52
8	Ablation studies using FDL (2)	53
9	PGT: Comparison table for various CNNs	75
10	PGT: Results for non-BN ResNets	81
11	PGT: Ablation studies	85

1 INTRODUCTION

Convolutional neural networks form the basis of many computer vision tasks including image classification [Deng et al. (2009)], object detection [Ren et al. (2015)], instance segmentation [Lin et al. (2014)], image generation [Goodfellow et al. (2020)], 3D reconstruction [Li et al. (2022)], super-resolution [Ledig et al. (2017)], pose estimation [Wang et al. (2019)] to name a few. With hundreds of neural networks that are being proposed every year, the necessity of learning better feature representations arise naturally. The task of learning better feature representations require exploring the objective landscape for more optimal solutions. In this space, techniques that exist in the literature hitherto are, namely more appropriate initialization strategies such as Kaiming initialization [He et al. (2016)] or Xavier initialization [Glorot & Bengio (2010)], and optimization addons such as momentum based stochastic gradient descent [Kiefer & Wolfowitz (1952)] [Robbins (2007)], Nesterov approximation [Nesterov (1983)], AdaGrad [Duchi et al. (2011)], RMSProp [Hinton et al. (2012)] and Adam (adaptive momentum) [Kingma & Ba (2014)]. The aforementioned optimizers uses first order gradients while optimizers such as BFGS solvers (Broyden-Fletcher-Goldfarb-Shanno algorithm) such as L-BFGS [Liu & Nocedal (1989)] are second order methods that estimate the Hessian matrix to steer the search through the variable space with limited memory and compute. Other techniques that also tries to improve the learning capacity of neural networks are data augmentation [Cubuk et al. (2020)], dropout [Srivastava et al. (2014)], batch normalization [Ioffe & Szegedy (2015)], skip connections [He et al. (2016)] and attention [Hu et al. (2018)]. Loss function

design has also been used to improve the learning capacity of neural networks [Goodfellow et al. (2014)]. Training tricks such as early stopping [Prechelt (1998)], learning rate scheduling [Loshchilov & Hutter (2016)] [Smith (2017)] and curriculum learning [Bengio et al. (2009)] are also used to improve the learning capacity of neural networks. Ensemble learning [Dietterich (2000)] is a technique that combines multiple models to improve the performance of a single model. In this thesis, we explore two techniques that are disjoint from the aforementioned techniques to improve the learning capacity of neural networks. The first one is how to alter the gradients generated by a neural network during training, and force the network to explore parts of the loss landscape that are inaccessible to the traditional way of training. With this we find that there are indeed solutions that are better than ones that neural networks automatically reach with normal training methods. The second method is improving the joint performance of many networks operating in a shared pool using adversarial training. Both of these technique are aimed to maximally utilize the inherent learning capacity of a network.

1.1 LEARNING CAPACITY

We first discuss learning capacities of neural networks in general, and what is meant by improving the learning capacity of neural networks. Unlike the capacity of communication channels that is used in communication theory, the learning capacity of a neural network does not have a mathematically rigorous definition as of writing this thesis. Hence, in this section we try to develop a notion of learning capacity that is used in formulating the problem statement of this thesis, and is subsequently used as a basis for defining the various cases of “unused” capacity that can occur in neural network training.

First we develop the notion of learning capacity of a single neural network trained in isolation. In this setting, there is only one neural network that is being

trained to achieve a particular objective. Since there is no direct notion of learning capacity, we use a proxy measure for it. We propose that when a neural network is able to achieve a higher *training accuracy*, it is said to have utilized a higher fraction of its inherent learning capacity. For example, if we have a ResNet-50 [He et al. (2016)] model that is trained on ImageNet [Deng et al. (2009)] dataset, and if it achieves a training accuracy of 75%, and another identical ResNet-50 model that is trained on the same dataset and achieves a training accuracy of 80%, then the second model is said to have a higher utilization of its inherent learning capacity. In this example, the learning capacity of the ResNet-50 model is the measured difference between the training accuracies of the two models. The accuracy metric is used as a proxy measure to observe how much the model has utilized its parameters to achieve the objective, which is image classification in this case. Note, that the accuracy metric is not a direct measure of learning capacity, but we use it with the assumption that a higher training accuracy implies that the model has utilized its parameters to achieve the objective better than a model with lower training accuracy. Therefore, our first notion of learning capacity is the difference in training accuracies of two models that are trained on the same dataset and architecture. We call this the intra-network learning capacity. Now we define inter-network learning capacity.

The notion of learning capacity is much broader than just the training accuracy metric. In a single network scenario, it is exactly the training accuracy of the model, but a notion of learning capacity can also be developed for scenarios where multiple networks operate in a pool. For example, if many satellites scan the surface of the earth for calamities and if each neural network onboard each of these satellites have the same network (with the same weights), then the total knowledge of the pool is no more than the knowledge of one single network. In this thesis, we also consider the notion of learning capacity of an ensemble of models. When multiple models are combined to achieve a particular objective, a concept of joint learning

capacity of the ensemble is also something that comes into the picture. We develop the notion of the learning capacity of an ensemble in the following way. When an ensemble of models achieves a higher ensemble training accuracy, it is said to have a higher utilization of the combined learning capacity of each of the base models in the ensemble. We call this the inter-network learning capacity.

The main difference between the intra and inter-network learning capacity is that, the first one is defined for a single unified architecture, while the other one is defined for a split architecture. Connections (neurons) within a single neural network pertain to intra-network learning as with each connection, the model gains a new dimension to optimize upon and gain the ability perform the objective in a better fashion. However, in inter-network learning (i.e. when we have many networks in a pool), we do not have connections amongst the slave networks. Thus we cannot analyze nor can we employ the same solutions or tricks that we usually employ for single network cases to improve the joint learning capacity of a pool. However, the joint network capacity is something that is still a valid concern. As per our previous example, if we have a swarm of satellites performing weather prediction by scanning images of the earth and let's say if all networks onboard each of these satellites are one and the same (with same model architecture and same model weights), then the total knowledge of the pool is no higher than the knowledge of one single satellite. This is similar if there are multiple cameras trying to scan objects on a conveyer belt in a factory application. Or if there are multiple autonomous self-driving vehicle assessing the same local scene to decide on a complex decision. In each of these cases, having neural networks onboard these systems helps in bringing disjunct information to the table. Similar to diverse knowledge that human bring to the table when people with different opinions work on the same problem, a pool of artificial neural networks working towards a shared objective can also benefit from having a disjoint / diverse knowledge by having different weights.

Thus our main motivation of defining learning capacity has been portrayed. The idea of having training accuracies as learning capacities is simple for a single neural network but what we want to emphasize is that the term **learning capacity** is a broader one, one that also captures the notion of the shared knowledge in a pool. We list out a set of points to reiterate the above:

- *Inter-networks* refer to the relationships and connections between different neural networks, such as in an ensemble where multiple models work together. On the other hand, *Intra-networks* pertain to the connections and relationships within a single neural network, like the connections between individual neurons or layers.
- The learning capacity of intra-networks focuses on how well a single model can learn from data and utilize its inherent potential. It's determined by factors such as the architecture's depth and breadth, the nature of the activation functions, and most importantly the optimization method. In this thesis, we propose a method of altering a network's gradient before the update step in the backward pass and thus force the network to look at different parts of the loss landscape that was previously inaccessible by the network. To our surprise, we find new optimas in the higher valleys of the loss landscape (with higher convergent loss), but which are more robust to a variety of network issues that can inhibit a network from maximally utilizing its learning capacity.
- Inter-network learning capacity, however, is concerned with how different models in an ensemble complement and enhance each other. It's determined by how diverse the individual models are and how their outputs are combined. We propose learning ensembles with adversarial training to directly emphasize diversity in the shared representations learned by networks in a pool.

1.2 MOTIVATION

Initialization strategies aim to put the starting state of the weights of the neural network in a favourable region, and by doing so it tries to control the flow of gradients in the early phases of training that hopefully leads to us to local minima. On the other hand, optimization strategies try to control the flow of gradients by automatically overriding hyperparameters such as the learning rate by augmenting it with additional hyperparameters such as momentum, first order moment and second order moments in Adam. Data augmentation tries to improve the learning capacity of neural networks by increasing the amount of data that is available for training. Dropout tries to improve the learning capacity of neural networks by randomly dropping neurons in the network during training. Batch normalization tries to improve the learning capacity of neural networks by normalizing the activations of the network to have zero mean and unit variance. Skip connections try to improve the learning capacity of neural networks by providing a shortcut path for gradients to flow through. Attention and self-attention tries to improve the learning capacity of neural networks by providing a mechanism to attend to important features in the input. With custom loss functions, neural networks can be trained towards a specific objective. Other training tricks such as early stopping, learning rate scheduling and curriculum learning tries to improve the learning capacity of neural networks by controlling the flow of gradients in the network. Ensemble learning tries to improve the learning capacity of neural networks by combining multiple models to improve the performance of a single model.

These techniques have been heavily explored and limited progress has been seen in recent times on these fronts. In this thesis, we aim to explore other techniques of improving learning capacities that are separate from the two aforementioned strategies.

1.3 CONTRIBUTIONS

The two strategies that we explore in this thesis are the following:

1. Inter-network adversarial learning to enhance joint knowledge of neural networks operating in a pool.
2. Intra-network gradient modification to force networks to choose stronger solutions on the loss landscape of the objective function.

In each of the two scenarios, we achieve our goal of improving the learning capacities of convolutional neural networks. However, each of the two topics we mention approaches to attack this problem in very different ways. First, we attack the problem of global knowledge in a swarm (or a pool) of neural networks. If there are N networks in a pool and all networks have both identical network architectures and identical weights, then the total knowledge of the pool is severely limited. Our first work aims to improve on this situation by incorporating shared knowledge in the training pipeline. With adversarial training, we allow each network to explore a much different part of the objective landscape than their peers (other networks in the pool). At convergence, this results in each network in the pool obtaining an optimum that is much different from the optimas achieved by peer networks existing in the same pool. With a prediction combiner head network, we can take the predictions of each network as an array of experts and combine them to form a superior prediction.

Secondly, we attack the problem of finding better optimas within a single neural network itself. As we mentioned earlier, the most common approaches towards solving this problem has been to either initialize the networks with better initial states and hope for smooth gradient flow, while the other strategy is to speed up and modify gradient descent through optimization. However, both of these methods and most traditional methods of training in general have relied on the gradients

that are generated by the traditional rules of calculus to train deep neural networks. In our second work, we aim to break away from the traditional way of updating a network's weights with the first order gradients and instead focus on modifying and transforming the gradients in such a way to force network weights to arrive at higher quality optimum. We do so by making the networks train with gradients that pushes it towards higher class separation. We show that our method, PowerGrad Transform, can push a network towards attaining states that are not possible with traditional training methods. In our analysis with PowerGrad Transform, we find that many local optimas exist in the loss landscape that has higher loss values, yet it yields a better classification performance on both the training and the test datasets. Such an optimum is out of reach through traditional means of updating network weights through gradient values derived from the first order rules of calculus.

1.4 OBJECTIVE FORMULATION

In this part, we elaborate on the structure of the problem that we aim to attack in this thesis, and then we discuss the scope of how we attempt at solving it. Primarily, we aim to improve the learning capacity of a convolutional neural network in different ways. A concrete rigorous definition of learning capacity is non-existent in the literature. So we first try to develop a sense of it first. To imagine what the capacity of a network architecture can be, it is often beneficial to think from the perspective of how the learning capacity or the learning capabilities of a network can be hindered. One way in which the learning capabilities of a neural network can get impeded is when a neural network gets stuck in a very sharp valley (local minima) of the loss landscape, during the gradient based iterative weight updation process. Another way in which a neural network can underperform, is when the final convergent optima is of a lower quality (even though the regions around the optima is quite flat). This means that the neural network could have been trained

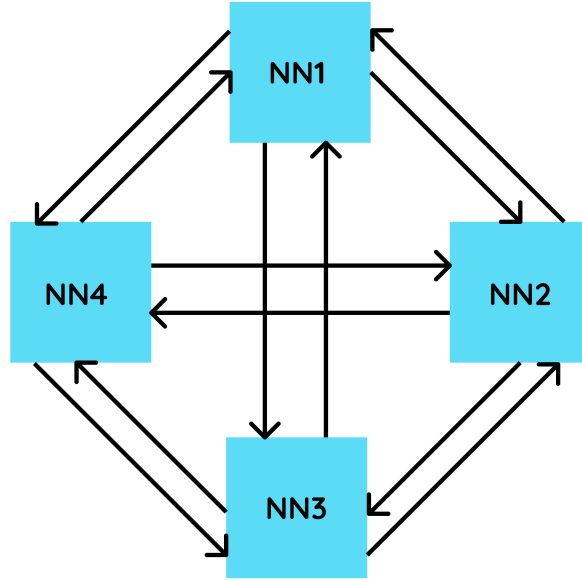


Figure 1: **(Problem 1)** — Inter-network setting for improving learning capacities of convolutional neural networks. The figure shows four networks working in an ensemble with information sharing represented as arrows.

to acquire a different convergent optima, but it wasn't able to get there. These are examples of ways in which the learning capacity can get impeded within a single network structure. However, the concept of learning capacity can also come into place when we have a pool of many neural networks. Let's say we have $N \geq 2$ neural networks working at a common task. The total capacity of a pool depends on the variety of information that each neural network captures. Here the total capacity is not just a measure of individual performance, but rather the combined knowledge of all entities in the pool.

1.4.1 INTER-NETWORK LEARNING CAPACITY

We attack both of these problems in two different chapters of our thesis. First, we attack the problem of pool knowledge. In this scenario, we have multiple neural networks working in tandem as an ensemble. In traditional ensemble techniques, knowledge variety is not learnt during training, i.e. each neural network

participating in the pool do not automatically get to know the expanse of the shared knowledge that other members of the pool has already acquired at any given iteration of training. We bring in knowledge sharing to make every member network of the pool know about the shared knowledge base that has already been acquired by the pool, and then with adversarial training, we try to dissuade every network from arriving at similar representations during the course of training. Doing so, we observe that each neural network diverts away from their normal course of learning trajectory on the loss landscape and arrive at far apart optimas at convergence. This brings in diversity in the learnt representations and enhances the total learning capacity of the pool. We show a figurative representation of this problem statement in Fig. 1.

1.4.2 INTRA-NETWORK LEARNING CAPACITY

Next we try to attack the problem of enhancing the learning capacity within a convolutional neural network itself. Traditionally, it is observed that lower loss values acquired during training leads to improved accuracies on the training dataset. This means that high quality optimas in the loss landscape that exist within higher loss surfaces are ignored. We aim to selectively modify gradients during the backpropagation process in such a way that forces the neural network to explore regions of high loss values. We observe that there exists optimal points in high loss surface regions that can give higher training accuracy than optimas located in low loss surface regions. These optimas are not only better than those obtained with traditional means of training, but are also flatter (meaning that they perform better on the test set as well). We show a visual representation of this problem in Fig. 2.

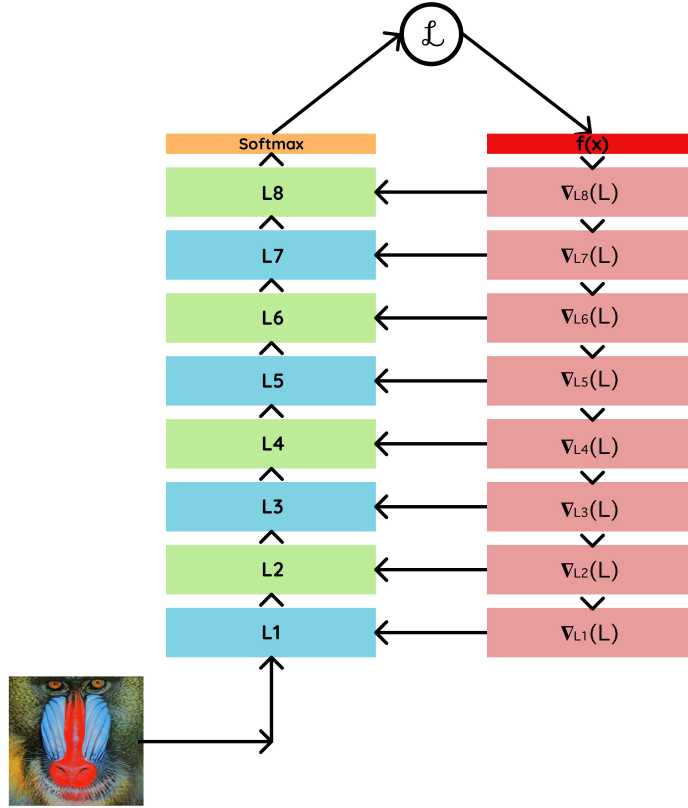


Figure 2: **(Problem 2)** — Intra-network setting for improving learning capacities of convolutional neural networks. The figure shows the forward pass on the left and the backward pass on the right. We modify the gradients at the junction marked as $f(x)$. This modification automatically modifies all other gradients dependent on it.

1.5 ORGANIZATION OF THE THESIS

The thesis is organized in the following manner. We split our thesis in **6** chapters, listed as follows:

Chapter (1) — *Introduction*: We have provided the motivational grounds for our work. Our work ties several different strategies for optimizing a convolutional neural network’s learning capacity. In several different scenarios, **(i)** Inter-network learning and **(ii)** Intra-network learning, we discuss our strategies for attacking the problem. In addition to that we discuss a way of improving learning from an architectural point of view. We concretize our problem statements and explore the landscape of possibilities within which we aim to explore solving the problem. We also delineate the organizational structure of this document.

Chapter (2) — *Mathematical Preliminaries*: We discuss about the basics of convolutional neural networks in a general sense. We introduce the fundamentals of a convolutional neural network (aka *ConvNet*), namely the convolution layers, pooling layers, activation layers, the choice of loss functions, etc. We also discuss the properties of ConvNets that make them stand out in vision processing tasks.

Chapter (3) — *Related Works*: We discuss prior relevant attempts that have been made towards improving both inter-network and intra-network learning capacity.

Chapter (4) — *Knowledge Diversification*: We discuss a novel method of training multiple convolutional neural networks in a pool with knowledge sharing and adversarial loss functions that leads each network to arrive at superior optimas. Each network is made aware of other networks in the pool and together they enhance group knowledge.

Chapter (5) — *PowerGrad Transform*: We discuss another novel technique, PowerGrad Transform, which we use to separate the forward pass from the

backward pass and use the modified gradients to update a neural network's weights in such a way that make it separate classes more strongly. We show a remarkable result, that even at higher cross-entropy loss surfaces, it is possible to obtain superior accuracies (on both training and test datasets). The effect is stronger for deeper networks trained on larger datasets. This runs counter-intuitive to traditional approaches of neural network training, where it is expected that loss minimization leads to better performance.

Chapter (6) — *Conclusion*: We make overall conclusion about the entire thesis and also discuss future work that can be built on top of the current thesis.

2 MATHEMATICAL PRELIMINARIES

In this chapter, we discuss some of the building blocks of convolutional neural networks. In deep learning architecture, many building blocks are used for various purposes. However, not all of them are used in the vision domain for tasks such as image classification and object detection. Many blocks are suitable for use in other domains, such as time-series data, or text based data sources. In the following sections, we only discuss about the types of blocks are used in vision architectures.

2.1 CONVOLUTIONAL LAYERS

The convolution operation between two functions f and g is denoted by $f * g$ is defined as:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x - y)dy \quad (1)$$

Images are discrete signals. The discrete convolution operation is defined as:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] \quad (2)$$

To define the convolution of two finite sequences, we can extend the sequences to finitely supported functions of the set of integers. The Cauchy product of the coefficients of the sequences is the set of coefficients of the ordinary product of two

polynomials, and it is the same as the convolution of two sequences. Thus, if g has a finite support $-M, -M + 1, \dots, M - 1, M$, then the finite summation is used:

$$(f * g)[n] = \sum_{m=-M}^M f[n - m]g[m] \quad (3)$$

However, a discrepancy exists in the nomenclature of the term convolution that is in traditional signal processing and the one that is used in deep learning literature. In signal processing, the definition (as defined above) requires one of the signal to be flipped before performing the Cauchy product, but in the deep learning literature, no signal (or kernel) is flipped. This makes the convolution operation in deep learning similar to cross-correlation, which is defined as:

$$(f \star g)(x) = \int_{-\infty}^{\infty} f(y)g(y + x)dy \quad (4)$$

Similarly, the discrete version of the cross-correlation is used in case of discrete signals such as images.

$$(f \star g)[m] = \sum_{n=-\infty}^{\infty} f[n]g[n + m] \quad (5)$$

with the finite sequence definition being:

$$(f \star g)[m] = \sum_{n=-M}^M f[n]g[n + m] \quad (6)$$

Here onwards, we refer to this definition of cross-correlation as the convolution operation in deep learning. Often one of the functions (say f) is referred to as the

signal (image, sound or video) and the other function (say g) is referred to as the filter or the kernel.

We now define what a convolutional layer is in the context of deep learning [LeCun et al. (1998)]. Images are multi-dimensional discrete signals having several channels (2 for grayscale images, 3 for RGB and any $n > 1$ channels for feature maps). Hence in most cases, we represent images or features maps as tensors of rank 3. Furthermore, in most deep learning training scenarios, we used mini-batches to train the neural networks. Mini-batches are nothing but a collection of several multi-channel images stacked on top of each other. This makes the tensor representation of a mini-batch of images or features, a tensor of rank 4. We denote such an entity with the symbol $X(N, C_{in}, H, W)$, where N is the number of images/features in a mini-batch, C_{in} denotes the number of channels present in each image or feature (aka input channels), while H and W represents the height and width of each image/feature respectively. For the purpose of convolution, we have kernels (or filter banks) that is also a tensor of rank 4, which we denote as $W(C_{out}, C_{in}, k_1, k_2)$ where C_{out} are the number of channels present in the convolved output of W and X . k_1 and k_2 is the height and width of each kernel respectively. We also have a bias vector for each filter in the filter bank (denoted by $B(C_{out})$). With this setup, we can now define the convolution operation for a batch of multi-channel images as:

$$Y(N_i, C_{out_j}) = B(C_{out_j}) + \sum_{k=0}^{C_{in}-1} W(C_{out_j}, k) \star X(N_i, k) \quad (7)$$

We leave out the height and width (spatial dimensions) for clarity. Several other variables are used as hyperparameters on top of the above definition of convolution, like zero-padding, stride, dilation, groups, etc. Padding zeros around the spatial dimensions of an input image or an intermediate feature tensor is done to ensure

maximum overlap between the kernel and the feature tensor. If padding is not used, then the spatial shape of the Y tensor reduces by $(k_1 - 1)/2$ and $(k_2 - 1)/2$ in the vertical and horizontal directions respectively. This is avoided by increasing the spatial size of the input tensor X by the same amount, in order to compensate for the reduction in spatial shape. Strided convolutions with stride factor (S) involve shifting the kernel by S amounts after each multiply-and-add operation. This reduces the output spatial dimensions by a factor of S in each direction. Dilated convolutions ignores certain pixels in the kernel, often in a grid like manner. In grouped convolutions, certain channels are packed together as a group and filters dedicated to each individual group are kept separate from other groups.

2.2 POOLING LAYERS

A pooling layer is used to discard unimportant information present in a feature tensor. There are several ways to pool a feature tensor, the most common being the *maxpool* operation [Krizhevsky et al. (2017)]. A 2D maxpool operator is a kernel of shape $k_1H \times k_2W$ that operates on an input feature tensor in strides of S_1 and S_2 . The operation is defined as follows:

$$Y(N_i, C_j, h, w) = \max_{m=0, \dots, k_1H-1} \max_{n=0, \dots, k_2W-1} X(N_i, C_j, S_1h + m, S_2w + n) \quad (8)$$

The maxpool operator basically selects the maximum value from a local patch of the feature map, and discards the rest of the non-maximum values. Doing so, makes the neural net propagate only the highest activation regions of an incoming feature tensor. Both zero-padding and dilated variants of maxpool is possible.

Another form of pooling that reduces the amount of available information without outright discarding any, is the *average pooling* operation. In similar fashion to the previous definition, it is defined as:

$$Y(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} X(N_i, C_j, S_1h + m, S_2w + n) \quad (9)$$

Similar to maxpool, the average pool operator selects a patch of pixels from the feature tensor, but instead of discarding the non-maximum values, it averages out all the values contained in that region. A variant of average pooling is the *global average pooling* operation. This operator is same as average pooling but having a kernel shape equal to that of the input feature tensor. All pooling layers reduce the spatial size of the input feature maps.

2.3 RESIDUAL CONNECTIONS

In feedforward networks (with no branching), each layer is connected only to the next layer. The functional form of a layer in such a network can be described as:

$$Y = F(X) \quad (10)$$

where X is the input tensor to the layer and Y is the output tensor. Here we denote F as the function that a layer learns during the training process. Layers with residual connection modify the above formulation as:

$$Y = F(X) + X \quad (11)$$

The extra term indicates the presence of a *residual connection* (also called a *skip connection*) [He et al. (2016)] that connects the input tensor to the output tensor through the process of addition. The benefit of such a formulation is that

it relieves the layer from having to learn the identity i.e. the output tensor equals the input tensor even with zero filter weights. Residual connections can be used to connect multiple layers instead of just one layer, in which case the formulation becomes:

$$Y = F_1(F_2(F_3(\dots F_n(X)))) + X \quad (12)$$

2.4 ACTIVATION LAYERS

Activation layers are used to induce non-linearity in neural networks. The most common activation layer is the *Rectified Linear Unit* (or ReLU for short) [Nair & Hinton (2010)], the definition of which is as follows:

$$ReLU(x) = \max(0, x) \quad (13)$$

The ReLU layer completely suppresses all information that has a value less than zero. A variant of ReLU that is also used as activation layer in scenarios where the negative values need to be preserved is the *Leaky ReLU* [Maas et al. (2013)], defined as:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases} \quad (14)$$

Other frequently used activation layers is the *sigmoid* activation:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \quad (15)$$

and the *hyperbolic tangent* (or *tanh*) activation [Bengio et al. (2007)]:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (16)$$

Both the sigmoid and the tanh activation functions squashes the range of the input tensor's values to the interval $(-1, 1)$.

2.5 NORMALIZATION LAYERS

Convolutional layers comprise many multiplication and addition operations. Often the results of the multiply-and-add operations can add up to very large numbers, leading to signal explosion in the forward pass. In the backward pass, this can lead to gradient explosion. This makes for poor training in neural network architectures that are very deep. Also, in the backward pass of deep network architectures, the increased depth can lead to vanishing gradients in the earlier layers of the network, which in turn leads to stalled updation of network weights, followed by poor convergence and increased training time. Another difficulty that commonly arises in convolutional networks is from the constant shifts in input distribution with each mini-batch. When neural networks are trained in mini-batches, the distribution of pixel values keep on changing with each mini-batch. This leads to the arbitrary shifts in the distribution of all feature tensor values as well. Any shift in the input propagates through the network, and this leads to poorer estimation of network weights. Normalization layers aim to solve this problem. The most common normalization layer used in convolutional neural networks is the *Batch Normalization* layer [Ioffe & Szegedy (2015)]. The operation of a 2D BatchNorm layer is defined as follows:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta \quad (17)$$

Here x and y represents the *channel-wise* random variable that describes input distribution and output distribution respectively. γ and β are trainable parameters that are learnt during training via gradient descent. One can see that the above operation does two things in succession. First, it whitens the existing distribution. Then it recolors the distribution to make it have a fixed mean and variance.

Other forms of normalization layers are *Layer Normalization* [Ba et al. (2016)], that performs the same operation as BatchNorm but is applied *feature-wise* instead of channel-wise. *Group Normalization* [Wu & He (2018)] is an extension of LayerNormalization, in that it coalesces groups of channels instead of taking all the channels as a single group. *Instance Normalization* [Ulyanov et al. (2016)] separates both channels and features dimensions, and applies normalization to per-channel, per-feature. Methods that apply normalization to weight tensors instead of applying it to feature tensors include *Weight Normalization* [Salimans & Kingma (2016)] and *Weight Standardization* [Qiao et al. (2019)].

2.6 FULLY CONNECTED LAYERS

Fully connected layers (aka the *linear layer*) often come at the end of a convolutional neural network architecture. It is simply the matrix multiplication of the input with a bias term:

$$y = xA^T + b \quad (18)$$

Here x is the vectorized version of any feature tensor that interfaces with the fully connected layer at its input. The parameters of the layer are A and b which are learnt during training, through gradient updates. The fully connected layer takes in a fixed size input and produces a fixed size output. The output vectors are commonly termed as *logits*. The linear layer is responsible for drawing decision boundaries (hyperplanes) in a high dimensional space. Most convolutional neural networks contain a single linear layer.

2.7 SOFTMAX LAYER

The logits produced by a fully connected layer have a range of $(-\infty, \infty)$, often the range is squashed into an interval of $(0, 1)$ by a softmax layer:

$$p_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (19)$$

where x_i 's are the logits of a preceeding fully connected layer. This prepares the vector to be treated as probability distribution p_i , which can then be compared to a one-hot representation of the ground truth probability values for the purpose of computing various loss functions.

2.8 LOSS FUNCTIONS

To compare probability distributions, the *cross-entropy* loss function is often employed. It is defined as:

$$\mathcal{L}_{Xent} = \sum_{i=0}^{C-1} -q_i \log(p_i) \quad (20)$$

where p_i denotes the predicted probability values, q_i denotes the ground truth probability values and C is the number of unique classes present in the classification dataset.

Other commonly used loss functions are, the *mean-squared error* loss function:

$$\mathcal{L}_{MSE} = \sum_{i=0}^{M-1} (\theta_i - \phi_i)^2 \quad (21)$$

where θ is the predicted value (such as a pixel in an image) and ϕ is the target ground-truth value at the same index i , $0 \leq i \leq M - 1$.

2.9 HYPERPARAMETERS & REGULARIZATION

We have discussed several hyperparameters in previous sections, namely the amount of zero-padding, stride, dilation and kernel shapes. Two other important hyperparameters are the width and depth of a network. The depth of a network is one of the single most important parameter in choosing a network architecture. With BatchNorm and residual connections, it has become possible to train very deep networks (north of 100 layers). Also, classification accuracy generally tends to increase monotonically with depth, though the returns tend to diminish gradually at very high depths. However, compute and memory requirements increase significantly with deeper networks, leading to lower efficiency at the high end.

It is possible to have a single layer network approximate any function, if the layer has sufficient width. This is courtesy of the *Universal Approximation Theorem* [Cybenko (1989)]. However, it is difficult to train such a network with large width and unit depth. Deeper networks can extract much more variety of information with fewer parameters. This is because, repeated convolutions of convolutions can construct complex features that a single layer network cannot.

Regularization is often one of the most important factors to ensure good convergence in neural networks, especially with architectures bearing very high depth. *Data augmentation* [Krizhevsky et al. (2017)] randomly applies image processing techniques to an existing dataset to produce more examples of the same dataset. *Weight decay* disallows a network's weights to change by a large value with each iteration of gradient descent update. Mathematically, this is akin to the formulation of $\mathcal{L}_2$ regularization. Other regularization techniques include *stochastic depth* (dropping different layers randomly during training) [Huang et al. (2016)], *dropout* (dropping linear layer units randomly to promote the learning of independent features) [Srivastava et al. (2014)] and *label smoothing* (computing loss with respect to smooth distributions of target labels) [Müller et al. (2019)].

2.10 BACKPROPAGATION & OPTIMIZATION

Backpropagation [Rumelhart et al. (1986)] (short for backward propagation of errors) is a method which applies the chain rule of calculus to recursively calculate gradient of the loss w.r.t to any parameter in the a deep neural network. The setup is as follows. We have input vector I (a mini-batch of images). We have a neural network with D layers. The weights at layer l is w_l . The collection of all weights is denoted by $\theta = \{w_1, \dots, w_l, \dots, w_D\}$. Each layer takes in the feature map of the previous layer and computes an output feature map, which formally is as follows:

$$A_l = F_{w_l}(A_{l-1}) \quad (22)$$

The loss function of the neural network $\mathcal{L}(X, \theta)$ is a function of all parameters of the network. To update the network's parameters with gradient descent, we need to find the gradient of the loss w.r.t each parameter, i.e. $\frac{\partial \mathcal{L}}{\partial w_l}$, which we do so by using the backpropagation algorithm. The backpropagation algorithm starts at the

final layer and recursively progresses towards the first layer, calculating the gradient in each step using the chain rule. First we calculate the derivative of the loss itself $\frac{\partial \mathcal{L}}{\partial w_D}$. Therefore by chain rule, the gradient of the loss w.r.t any parameter w_l is given by:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{\partial A_l}{\partial w_l} \frac{\partial \mathcal{L}}{\partial A_l} \quad (23)$$

which can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{\partial A_l}{\partial w_l} \frac{\partial w_{l+1}}{\partial A_l} \frac{\partial \mathcal{L}}{\partial w_{l+1}} \quad (24)$$

which can further be expressed as:

$$\frac{\partial \mathcal{L}}{\partial w_l} = \frac{\partial A_l}{\partial w_l} \frac{\partial w_{l+1}}{\partial A_l} \cdots \frac{\partial A_D}{\partial w_{D-1}} \frac{\partial \mathcal{L}}{\partial A_D} \quad (25)$$

This process continues until exhaustion. In the end of this process, we get all the values of the gradient of each and every parameter of the network. We can then proceed to update the network's weights using *gradient descent* which is given as:

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \quad (26)$$

where θ_t represents the state of the network's parameters at the t 'th iteration of gradient descent, and λ is a hyperparameter known as the learning rate. This method of updating the network's weights is known as *Stochastic Gradient Descent* optimizer (or *SGD* for short).

Stochastic gradient descent [Kiefer & Wolfowitz (1952)] can be quite slow to converge and is often prone to getting stuck at sharp local minimas and saddle points. A modification to the traditional SGD based update is the inclusion of *momentum* hyperparameter in optimization. *SGD with momentum* is a variant of SGD that is formulated as:

$$\begin{aligned} v_{t+1} &= \rho v_t + \frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \\ \theta_{t+1} &= \theta_t - \lambda v_{t+1} \end{aligned} \quad (27)$$

Here v is an intermediate variable that keeps track of the velocity vector of a parameter's state during the optimization proces. Then it uses ρ to build up that velocity vector as running mean of gradients. Finally, it calculates the position vector using the velocity vector and completes the optimization step. SGD + momentum is leads to faster convergence and is robust to poor local optimas and saddle points.

A slight modification to the formulation of SGD + momentum is the *Nesterov momentum* optimizer [Nesterov (1983)], which approximates the future position of the state vector (in a “lookahead” fashion) and uses that to compute the gradient instead of the old position. The formulation is as follows:

$$\begin{aligned} v_{t+1} &= \rho v_t - \frac{\partial \mathcal{L}(X, \tilde{\theta}_t)}{\partial \theta} \\ \tilde{\theta}_{t+1} &= \tilde{\theta}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned} \quad (28)$$

where $\tilde{\theta}$ represents the position vector approximation.

The aforementioned optimizers applies hyperparameters globally. Optimizers that apply the optimization update on a per-parameter basis are *Adagrad* [Duchi et al. (2011)], *RMSprop* [Hinton et al. (2012)] and *Adam* [Kingma & Ba (2014)].

Adagrad tracks the sum of squared gradients (per-parameter), and then the square-root of it is used to normalize the learning rate in the update step.

$$\begin{aligned} G_{t+1} &= G_t + \left(\frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \right)^2 \\ \theta_{t+1} &= \theta_t - \frac{\lambda}{\sqrt{G_{t+1} + \epsilon}} \frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \end{aligned} \quad (29)$$

RMSprop builds on top of Adagrad, in which it addresses the slow monotonic decreasing learning rate phenomena of Adagrad. It introduces another hyperparameter called the decay rate (β). The formulation is as:

$$\begin{aligned} G_{t+1} &= \beta G_t + (1 - \beta) \left(\frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \right)^2 \\ \theta_{t+1} &= \theta_t - \frac{\lambda}{\sqrt{G_{t+1} + \epsilon}} \frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \end{aligned} \quad (30)$$

The Adam optimizer is a sort of RMSprop with induced momentum. It consists of two additional hyperparameters, β_1 and β_2 which it uses to calculate the first moment μ and second moment ν . The formulation goes like:

$$\begin{aligned} \mu_{t+1} &= \beta_1 \mu_t + (1 - \beta_1) \frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \\ \nu_{t+1} &= \beta_2 \nu_t + (1 - \beta_2) \left(\frac{\partial \mathcal{L}(X, \theta_t)}{\partial \theta} \right)^2 \end{aligned} \quad (31)$$

We then calculate two intermediate variables, b_1 and b_2 :

$$\begin{aligned} b_1 &= \frac{\mu_{t+1}}{1 - \beta_1^t} \\ b_2 &= \frac{\nu_{t+1}}{1 - \beta_2^t} \end{aligned} \quad (32)$$

Finally, we calculate the updated weights:

$$\theta_{t+1} = \theta_t - \lambda \frac{b_1}{\sqrt{b_2 + \epsilon}} \quad (33)$$

All of the above optimizers are first order optimizers, meaning that they use only the first order gradient to update the weights of the neural network. However, most first order optimizers are associated with a lot of hyperparameters. Second order optimizers can perform optimization without the need for any hyperparameters such as learning rate, or momentum, etc. It is given as:

$$\theta^* = \theta_0 - H^{-1} \nabla_{\theta}(J(\theta_0)) \quad (34)$$

where $J(\theta_0)$ is the second-order Taylor series expansion of θ , which is given by:

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_{\theta}(J(\theta_0)) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0) \quad (35)$$

and H is the Hessian matrix. This is quite a problematic formulation because the update equation requires the Hessian matrix to be inverted ($O(n^3)$). The *BFGS* optimizer (*Broyden-Fletcher-Goldfarb-Shanno*) [Broyden (1965)] employs a quasi-Newton method to perform an $O(n^2)$ approximation of the inverse Hessian by collecting rank 1 updates over time, while the *L-BFGS* optimizer (*Limited Memory BFGS*) [Liu & Nocedal (1989)] does it with reduced memory cost.

3 RELATED WORKS

First we explore the existing literature in the context of inter-network learning. Forming a group of neural networks, or an ensemble, is a widely accepted approach to enhance the efficiency of learning-driven algorithms. The winning residual network from ILSVRC 2015, [He et al. (2016)], utilized six distinct base ResNet models. Traditionally, the training of base models has been conducted without considering the interdependence between them. The more diverse the features learned by the base models, the more effective an ensemble can be. This diversity means each model introduces a distinct set of features to the ensemble, thus encompassing a wider spectrum of information and enhancing generalization. The Random Initialization Ensemble approach changes the starting states of base models, leading to varied optimization paths. Deep Ensembles [Lakshminarayanan et al. (2017)] is a technique that follows this principle. However, these methods do not actively promote diversity during training. They typically require multiple experiments with varied initializations, after which a selection of the most diverse models is included in the ensemble. There are other ensemble techniques as well. SnapShot Ensemble [Huang et al. (2017)] and [Wen et al. (2019)], integrates checkpoint gathering with cyclic learning rate strategies like SGD with warm restarts [Loshchilov & Hutter (2016)]. Fast Geometric Ensemble [Garipov et al. (2018)], creates ensembles by merging modes as it navigates high-accuracy paths between them. Checkpoint ensembles [Chen et al. (2017)] use training checkpoints as ensemble candidates, merging them towards the training’s conclusion. Adaptive Ensemble [Inoue (2019)] lowers computational load by eliminating ensemble

processes for inputs that display high softmax outputs, leveraging confidence intervals. Auto-Ensemble [Yang & Wang (2020)] forms ensembles by gathering checkpoints and employing a learning rate scheduling mechanism.

Several methods exist for learning ensembles such as [Hansen & Salamon (1990)], [Krogh & Vedelsby (1994)] and [Dietterich (2000)]. Negative correlation learning [Alhamdoosh & Wang (2014)] and error independent ensembling [Giacinto & Roli (2001)] has also been utilized to create ensembles. In many cases, the dataset is often divided into many overlapping or non-overlapping subsets and fed to the base models and with enough hardware, the models can be simultaneously trained [Ge et al. (2020)] [Zhang et al. (2018)] and their predictions aggregated. Often, the base models are trained with different hyperparameters to reach different solution points. The base models are combined using model averaging, various types of voting (majority voting, soft voting or plurality voting) or weighted consensus (boosting voting) [Wu et al. (2020)]. In collaborative learning [Song & Chai (2018)], many ensemble head networks are used on the same network to improve robustness and generalization. A similar idea is explored in [Zhou et al. (2002)] and from the perspective of pruning in [Le et al. (2020)].

Some well-known techniques for obtaining ensembles from base models include Random Forest [Breiman (2001)] [Ho (1995)], Bagging [Breiman (1996)], Boosting [Schapire (1999)], and AdaBoost [Schapire (2013)], Random Layer Sampling [Lee et al. (2020)]. EnsembleBench [Wu et al. (2020)] tries to minimize the number of possible ensemble candidates required for evaluation while maximizing overall ensemble performance. It is widely accepted that maximizing diversity is critical to maximizing ensemble benefits. It is obvious that if all the base models are exact replicas of each other, the ensemble's performance is no better than any of its base models. Correlation between diversity and ensemble performance is explored in [Kuncheva & Whitaker (2003)] and [Partridge & Krzanowski (1997)].

Co-training losses ensure that all base models eventually perform similarly with the ensemble output being an average of all predictions. Student-teacher based ensembles has been explored in [Zhang et al. (2018)] [Lyu et al. (2021)] using distillation [Hinton et al. (2015)]. Co-training has been used to achieve diversification [Ge et al. (2020)] [Blum & Mitchell (1998)] [Qiao et al. (2018)] [Batra & Parikh (2017)] [Yang et al. (2020)]. Neural Architecture Search (NAS) based methods have also been used to create ensembles [Elsken et al. (2019)] [Weill et al. (2019)] [Fang et al. (2019)] [Jin et al. (2019)] [Zoph & Le (2016)] [Cortes et al. (2017)] [Zaidi et al. (2021)]. Loss functions and training routines that directly target model diversity has been explored in [Liu & Yao (1999)] [Lee et al. (2015)] [Zhou et al. (2018)] [Webb et al. (2019)] [Jain et al. (2020)] [Pearce et al. (2020)].

For intra-network learning, that is improving the learning capacity of a single neural network in isolation we have the following works:

Gradient Clipping (GC): Gradient Clipping [Pascanu et al. (2013)], often used in natural language processing methods [Merity et al. (2017)], is a technique that involves changing or clipping the gradients with respect to a predefined threshold value during backward propagation through the network and updating the weights using the clipped gradients [Zhang et al. (2019); Smith et al. (2020)]. By rescaling the gradients, the weight updates are likewise rescaled, significantly reducing the risk of an overflow or underflow [Pascanu et al. (2012)]. GC can be used for training networks without batch-normalization.

The formulation is as follows: If G is a gradient vector which is $\partial L / \partial \theta$ (the gradient of the loss L with respect to the parameters θ). In clip by value, the gradients are modified as:

$$G \rightarrow \begin{cases} \lambda \frac{G}{\|G\|} & \text{if } \|G\| > \lambda, \\ G & \text{otherwise} \end{cases} \quad (36)$$

At larger batch sizes, the clipping threshold in GC becomes highly sensitive and requires extensive finetuning for various models, batch sizes, and learning rates. As we demonstrate later in our studies, GC is not as effective in improving the performance of non-normalized networks. AGC performs better than GC in non-normalized networks.

Adaptive Gradient Clipping (AGC): Adaptive Gradient Clipping [Brock et al. (2021)] is developed to further enhance backward pass gradients than what is performed by GC. It takes into account the fact that the ratio of the gradient norm to the weight norm can provide an indication of the expected change in a single step of optimization. Here the normalized gradient is clipped by λ , which means that large weights can have large gradients and still the normalized gradient can be within λ . If it is not within the threshold, then their gradient is normalized by the ratio of norm of the gradient and norm of the weight in order to avoid gradient explosion. If W^l is the weight matrix of the l^{th} layer, and G^l is its gradient in the backward pass, then the following equation is used to modify the gradient of the i^{th} filter before its update:

$$G_i^l \rightarrow \begin{cases} \lambda \frac{\|W_i^l\|_F^*}{\|G_i^l\|_F} G_i^l & \text{if } \frac{\|G_i^l\|_F}{\|W_i^l\|_F^*} > \lambda \\ G_i^l & \text{otherwise} \end{cases} \quad (37)$$

The hyperparameter λ of AGC is less sensitive to changes in batch size and depth as observed in [Brock et al. (2021)]. However, when applied to vanilla residual networks (ResNet-18), the performance is still afar from normalized variants as demonstrated by our experiments. In most cases, AGC is used for training networks without batch-normalization.

Label Smoothing: Label smoothing, introduced by Szegedy et al. [Szegedy et al. (2016)], utilizes smoothing of the ground truth labels as a method to impose regularization on the logits and the weights of the neural network.

The formulation is as follows. If q_i is the value at the i^{th} index of the one hot encoded vector of the ground truth label, then the transformed distribution of the labels q'_i is:

$$q'_i = \begin{cases} 1 - \epsilon & \text{if } i = y, \\ \frac{\epsilon}{K-1} & \text{otherwise} \end{cases} \quad (38)$$

where ϵ is a hyperparameter with a value in $[0, 1]$ and y denotes the index of the correct class.

Müller et al. investigates how label smoothing work in [Müller et al. (2019)]. Using visualizations of penultimate layer activations, they demonstrate that label smoothing calibrates the networks predictions and aligns them with the accuracies of the predictions. Parallel works in label smoothing include Pairwise Confusion [Dubey et al. (2018)], combatting Label Noise [Reed et al. (2014)], achieving regularization through intentional label modification [Xie et al. (2016)].

Knowledge Distillation: The formulation of PowerGrad Transform is akin to temperature based logit suppression of [Hinton et al. (2015)]. Knowledge distillation [Ba & Caruana (2014)] is a process in which two networks are trained. First a teacher network is trained on a given dataset and then the soft-labels (the predicted probabilities) of the teacher network are used to train another network, called the student network. In similar style of label smoothing, knowledge distillation aims to generate smooth gradient behaviour by forcing the network not to overpredict on any single image. As in case of label smoothing, the student network's weights are automatically penalized if the network assigns a probability value higher than the soft labels generated by the teacher network. Variants of knowledge distillation include self-distillation [Zhang et al. (2019)] [Wang et al. (2021)], identical student network distillation [Furlanello et al. (2018)], channel distillation [Ge et al. (2019)], regularizing wrong predictions [Yun et al. (2020)] and representation or embedding based knowledge distillation [Aguilar

et al. (2020)] [Yao et al. (2018)] [Passalis & Tefas (2018)]. Distillation is applied to various tasks such as model compression [Wang et al. (2020)], face recognition [Ge et al. (2018)], enhancing ensemble performance [Zhang et al. (2018)], interpreting neural networks [Liu et al. (2018)]. Efficacy of distillation is extensively studied in [Cho & Hariharan (2019)] [Yuan et al. (2020)]. Applications of logit suppression include: deep metric learning [Zhai & Wu (2018)], non-parametric neighborhood component analysis [Wu et al. (2018)], sequence-to-sequence modelling [Chorowski & Jaitly (2016)], reinforcement learning [He et al. (2018)] and face recognition [Liu et al. (2017)] [Wang et al. (2018)].

Self-knowledge distillation is used in many different ways, which mainly focuses on improving the performance of the model by either creating multiple ensemble student models [Zhang et al. (2018)], using different distorted versions of same data with two different versions of a single network which guides one another [Xu & Liu (2019)] or injecting multiple sub-modules in the network to improve its capacity [Zhang et al. (2019)]. However none of these methods use normal traditional training without introducing sub-modules in the architecture.

Other techniques of weight update include slot machines [Aladago & Torresani (2021)], where random weights are generated and carefully selected during backpropagation in a way that minimizes the loss function.

4 KNOWLEDGE DIVERSIFICATION

In typical neural network ensemble methods, multiple networks are independently trained, and their outcomes are merged either through voting or linear combination. However, this approach doesn't promote information sharing among the ensemble's base networks. To boost the performance of neural network ensembles, it's crucial to diversify their representations. Therefore, we suggest a technique that enhances the learned representations within an ensemble by promoting feature sharing during the training goal. Our Feature Difference Loss (FDL) encourages the networks within the ensemble to acquire distinct features, thus directly enhancing model diversity. Testing with ensembles of two to four networks reveals marked improvements over other ensemble methods. These improvements are more pronounced for datasets with fewer samples per category, like MNIST, CIFAR-10, and CIFAR-100. Even on extensive datasets like ImageNet, we noted positive results. Furthermore, the benefits are consistent across various architectures, from basic ConvNets to advanced networks such as VGG and ResNets.

4.1 MOTIVATION

In section 1.1, we introduced the notion of the learning capacity of an ensemble of networks. We define it as the joint performance that a pool of networks can achieve in a given task. We call each individual network in a pool, as base models. It is known that when base models in an ensemble are trained from a different

initial condition, they almost always arrive at a different local optima. This, often, serves as a source of additional knowledge. Thus when the predictions of N such base models ($N > 1$) are taken, we find that we can combine their predictions using some form of voting scheme and achieve performance higher than what each individual model in the ensemble could have achieved. This is often known in the literature of ensemble learning as diversity. However, to the best of our knowledge, as of writing this thesis, there isn't any way to force diversity into a pool of base models. With this work, we introduce a method to force base models to oppose each other in an adversarial manner and arrive at very different optimal points on the loss landscape, guaranteeing high diversity in the base models.

The fact that diverse models leads to better ensemble performance is also directly connected to our notion of inter-network learning capacity. If diversity between ensembles is low or zero, then each network do not bring anything new to the table and consequently their ensemble performance suffers. If the parameter count of a single base model is P , then the parameter count of N such identical base models is NP , and hence it is expected that the learning capacity of the cohort of N base models should at least surpass the base models themselves. However, this is contingent to the diversity of learnt representations that each base model learn during training. In traditional neural network training, there is no way to train multiple neural networks in a connected manner (in the sense that gradients from one network do not flow into another network during training because they are treated as separate computation graphs). Consequently, there is no way for any one of the base models in the cohort to know what other base models are learning. In this work, we allow features from one network to travel to other networks in the ensemble and by doing so allow all networks in the ensemble to be aware of all other networks in the pool. Then we invoke adversarial training to force each network to oppose each other and explore the loss landscape to search disjoint information and learn diverse feature representations.

4.2 INTRODUCTION

In various applications, identical neural networks are used. For instance:

1. Self-driving vehicles in a local area each using identical neural networks.
2. Factories utilizing neural networks to detect fractures in components moving on a conveyor belt.
3. Satellites monitoring the climate analyzing the earth.

In these contexts, the same overall objective is assigned to multiple instances of identical neural networks. However, if these networks possess the same weights, the collective knowledge remains equivalent to that of a single network, restricting enhanced performance. To overcome this limitation, our goal is to diversify the knowledge within a pool of neural network (with identical structure but different weights). We have introduced a training strategy designed to enrich the collective knowledge of the pool by encouraging diverse feature representations among the networks. This is achieved using our Feature Difference Loss (FDL), a loss function that promotes diversity by making each network learn unique features. Our novel training regimen requires sharing feature tensors among the networks during the training process. This sharing ensures the cumulative knowledge of the pool is disseminated throughout the networks, optimizing for diversity. As networks gain insights into the collective knowledge of a pool, they can be driven towards different optimal solutions through the FDL, effectively making the base models compete in a minimax style. This training approach doesn't introduce new hyperparameters beyond the usual ones. We have conducted thorough tests on various models and datasets, ranging from simple ConvNets to deeper architectures like VGG, ResNet-18, and ResNet-50. These networks were trained on datasets like MNIST, CIFAR-10, CIFAR-100, and ImageNet-1K with up to four networks. Our results indicate that FDL consistently enhances performance across all tested scenarios.

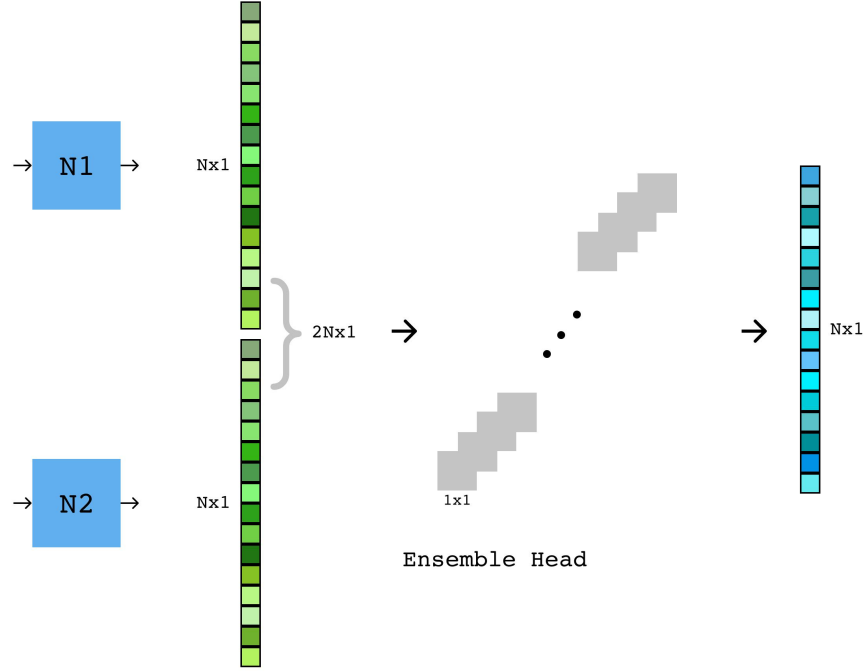


Figure 3: The architecture of the ensemble head network. Predictions of two identical networks are combined into a single prediction vector using $2N$ linear units or $2N$ (1×1) convolution.

4.3 METHOD

We start with N identical base networks. Without loss of generality, we portray the $N = 2$ case. In Fig. 3, we show networks N_1 and N_2 . The two networks produce two sets of prediction vectors independently, and then those two vectors are concatenated and fed to an ensemble head network, N_E . The ensemble head comprises a single layer of 1×1 convolution kernels and it learns the optimal linear combination to combine the prediction vectors at its input. All networks N_1 , N_2 and N_E are trained using cross-entropy loss. However, we also have other losses during training which we portray in next section.

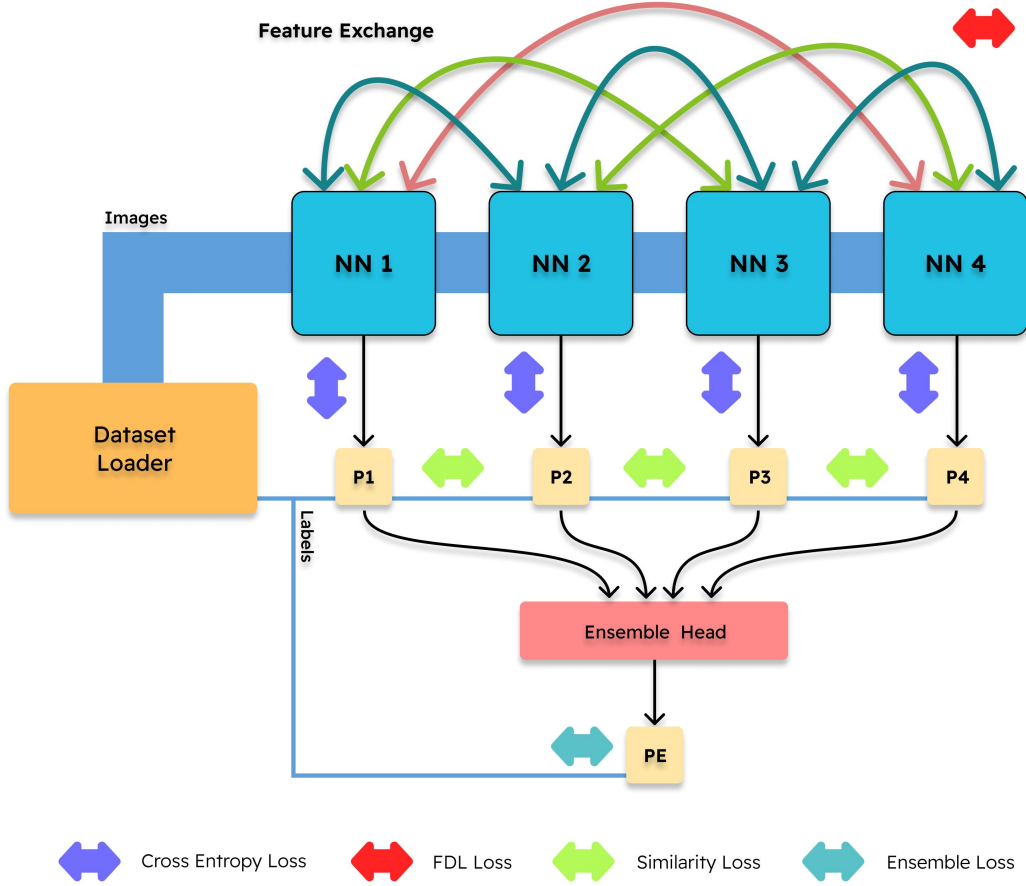


Figure 4: The architecture of four identical neural networks trained with FDL. All networks share a common minibatch. Features are exchanged and the networks weights are updated so as to maximize feature differences between them. The double sided arrows represent the different loss functions and their position indicates the location where they are invoked. Prediction vectors (P1, P2, P3, P4) are combined using an ensemble head to create PE.

If $\vec{v}_1$ and $\vec{v}_2$ are the prediction vectors for an input mini-batch I , then for networks N_1 and N_2 respectively, we have:

$$\vec{v}_1 = (v_{11}, \dots, v_{1p}) = \mathcal{F}_1 * I \quad (39)$$

$$\vec{v}_2 = (v_{21}, \dots, v_{2p}) = \mathcal{F}_2 * I \quad (40)$$

where $\mathcal{F}_1$ and $\mathcal{F}_2$ represents the parameters of N_1 and N_2 respectively, and $*$ is the convolution operator.

The output of the ensemble head can then be represented as,

$$\vec{v}_e = (v_1, \dots, v_p) = K * (v_{11}, \dots, v_{1p}, v_{21}, \dots, v_{2p}) \quad (41)$$

where K is a set of 1×1 convolution kernels.

During the training of the ensemble head, no gradient is passed back to the base networks. The training routine of the base networks and the ensemble head is performed in five phases, which we describe in section 4.3.2.

4.3.1 FEATURE DIFFERENCE LOSS FUNCTIONS

Convolutional layers operate as 3D filters to generate output images from input images. All convolutional layer features can most closely be interpreted as pixels in an image, and the $\mathcal{L}_2$ loss function is the most effective method for comparing pixels. Given a convolutional base neural network N , we define W_i as the i^{th} convolutional layer of the network. A feature vector F_{i-1} is fed into W_i , and the output F_i is generated.

$$F_i = W_i * F_{i-1} \quad (42)$$

We denote the shape of the tensor F_i as (B, C, H, W) where B is the number of images in a batch, C is the number of channels of each image and $H \times W$ is the resolution of each image. In Fig. 4, we show the architecture and the computation graph of four neural networks trained using our proposed method. First we portray the two network case, following which we extend our method for ensembles of more than two networks in section 4.3.2. Given, two identical networks N_1 and N_2 , we compute the feature difference loss for the i^{th} layer as:

$$L_i^{N_1, N_2} = \frac{1}{BCHW} \sum_{b=0}^{B-1} \sum_{c=0}^{C-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} (F_i^{N_1}(b, c, h, w) - F_i^{N_2}(b, c, h, w))^2 \quad (43)$$

The value of the feature loss is then summed up and averaged over the depth (D) of the network as:

$$L_{FDL}^{N_1, N_2} = \frac{1}{D} \sum_{d=0}^{D-1} L_d^{N_1, N_2} \quad (44)$$

Given the cross-entropy loss of network N_1 and N_2 as L_1 and L_2 respectively, we define the similarity loss function as:

$$S^{N_1, N_2} = (L_1 - L_2)^2 \quad (45)$$

Ideally, we want to perform maximization of the feature difference loss, Eq. 44. However, as is commonly followed in the training routines of generative adversarial networks (GANs), the discriminator's loss functions are not maximized, rather the negative of it is minimized. Here too, we minimize the negative of the FDL loss function, Eq. 44.

We pose the optimization criterion as:

$$N_E^*(W) = \underset{W}{\operatorname{argmin}} (-L_{FDL}^{N_1, N_2} + k S^{N_1, N_2} + k_1 L_X(\hat{y}_{N_1}, y) + k_2 L_X(\hat{y}_{N_2}, y)) \quad (46)$$

L_X is the cross-entropy loss function that acts on the the prediction vector $\hat{y}$ and the ground truth y . The constants k , k_1 , k_2 controls the weights ('importance') of the individual losses. However, directly optimizing Eq. 46 requires careful tuning of the constants. To avoid this, we train the entire system in several different phases

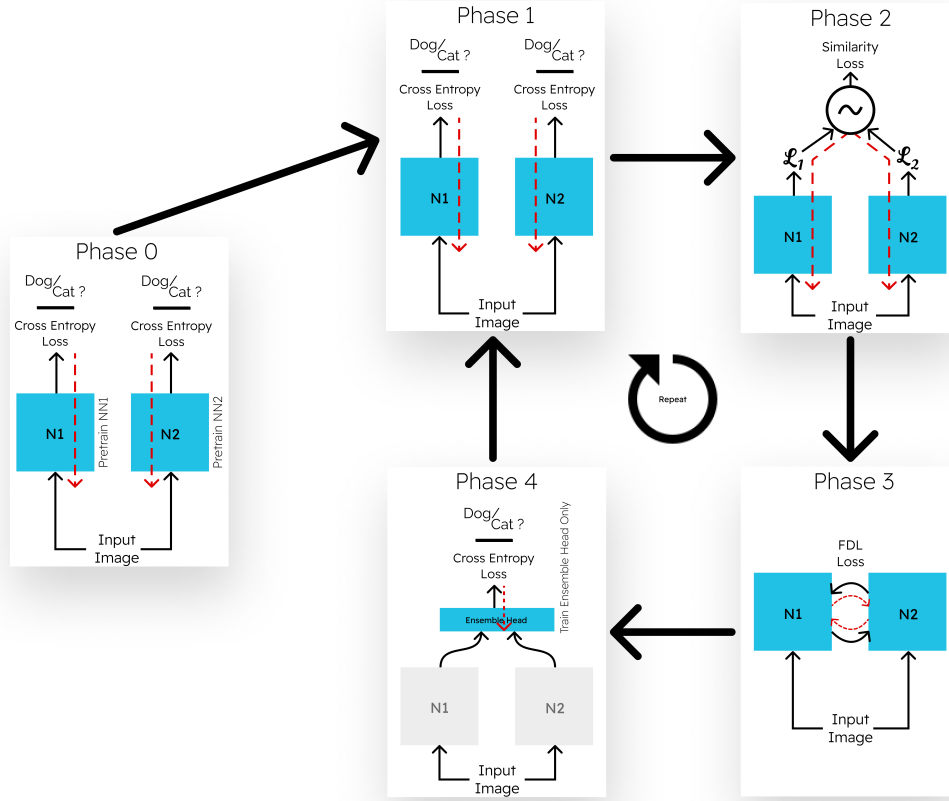


Figure 5: The training pipeline of two networks N1, N2 along with the ensemble head network trained with FDL loss. The red dotted line indicates the flow of gradients during backpropagation.

with each phase targeting a single loss function and it entirely eliminates the task of tuning the constants of Eq. 46.

4.3.2 TRAINING PHASES

4.3.3 PHASE 0

To train the ensemble network with the different losses of Eq. 46, we develop a training algorithm that is akin to the way generative adversarial networks are trained, i.e. in phases. We have five different phases of training as portrayed in Fig. 5. Without loss of generality we discuss the $N = 2$ case, i.e. two base models and a single ensemble head as the ensemble network. During training, we train N_1 and N_2 independently from a different seed for a few epochs. We denote

this pretraining phase as the zero'th phase as it occurs only once in the training cycle. After the pretraining phase is completed, phase 1 through 4 repeats in a cycle occurring once every iteration until convergence. If we invoke the FDL loss function from the very first iteration (i.e. without invoking phase zero), the training progresses haphazardly, leading to instability. So we pretrain N_1 and N_2 for a few epochs depending on the dataset. Smaller datasets such as the CIFAR datasets, the number of pretraining epochs are often just one epoch. Larger datasets require more pretraining. We mention all training details and hyperparameters for complete reproducibility in the supplementary section of this article. It is important to note that we stop training much before convergence. At this state, the networks have not reached their convergence point yet. If we invoke the FDL loss from this point onwards, the two neural networks diverge from their paths and arrive at different and diverse optimal points. The FDL loss maximizes the gap between the two optimal points, by making the networks compete against each other. The farther the final optimal points of each base network, the better it is for the ensemble as a whole.

4.3.4 PHASES 1 AND 2

In the first phase, we train the base models with a single minibatch from the training dataset and immediately move on to the next phase. In the second phase of training, we invoke the similarity loss. In a two network setting, we take the loss values of the two networks and compute the mean squared error between them. Since the two loss 'tensors' are a part of the computation graph, the effect of the mean squared error computation backpropagates through the entire network. The aim of the similarity loss function is to stabilize training. With the similarity loss, the cross-entropy loss values are kept within a reasonable range of one another. The reason why we choose to compare the loss values instead of the prediction vectors is the following. The loss value (a scalar) represents the height of a point on the

loss landscape whose value is determined by the parameters of the network and the input vector (a minibatch of training images). Several different points on this loss landscape can have the same loss value. By comparing the loss values of the base models, we force the network to arrive at a ‘similar’ quality optima at convergence. Due to the fact that stability issues are typical when training neural networks with negative loss landscapes (FDL or GAN), the presence of the similarity loss function is essential. If in case we train the base models to only minimize the negative of the FDL loss without having the similarity loss in the training pipeline, we observe that in some cases the weights of one of the base networks goes to zero, while the other network arrives at a perfectly good optimal convergence point. This is intuitively plausible, as it is optimal to ‘sacrifice’ the accuracy of one of the networks to have the FDL loss at a maximum value since the feature vector differences will then be at their greatest. The presence of the similarity loss prohibits this phenomena. Also, in our formulation of the similarity loss we do not compare the prediction vectors and instead choose to just compare the loss values. This is because it is possible for the networks to have the same loss value for different prediction vectors. Comparing the prediction vectors would instead lead the networks to arrive at the exact same optimal point, which will lead to a loss in ensemble diversity. A group of networks that output the same prediction vectors for any input images does not provide any additional information to the ensemble. In case of N networks, we find that invoking the similarity loss between every pair of ${}^N C_2$ networks is computationally intensive and so we randomly choose any two networks per iteration and invoke the similarity loss objective only on those two networks. A uniform random sampler ensures that all pairs of networks are chosen with equal likelihood.

4.3.5 PHASE 3

In the third phase, we invoke the FDL loss. In this step, we collect all output activations from only the convolutional layers. We broadcast the activation tensors

Algorithm 1: Training routine for two identical base networks in an FDL ensemble.

Given: Identical networks N_1 and N_2 , Ensemble Head network N_E
Data: $\mathbf{I}$ =Image, $\mathbf{y}$ =Label.

```

Pretrain  $N_1$  for one epoch.                                /*Phase 0*/
Pretrain  $N_2$  for one epoch.

while  $iter \leq iter_{max}$  do
     $\hat{\mathbf{y}}_1 \leftarrow N_1(\mathbf{I})$                                 /*Phase 1*/
     $L_1 \leftarrow L_X(\hat{\mathbf{y}}_1, \mathbf{y})$ 
    BackpropagateAndUpdate $_{N_1}(L_1)$ 
     $\hat{\mathbf{y}}_2 \leftarrow N_2(\mathbf{I})$ 
     $L_2 \leftarrow L_X(\hat{\mathbf{y}}_2, \mathbf{y})$ 
    BackpropagateAndUpdate $_{N_2}(L_2)$ 

     $L_1 \leftarrow L_X(N_1(\mathbf{I}), \mathbf{y})$                                 /*Phase 2*/
     $L_2 \leftarrow L_X(N_2(\mathbf{I}), \mathbf{y})$ 
     $S^{N_1, N_2} = (L_1 - L_2)^2$ 
    BackpropagateAndUpdate $_{N_1, N_2}(S)$ 

     $\hat{\mathbf{y}}_1 \leftarrow N_1(\mathbf{I})$                                 /*Phase 3*/
     $\hat{\mathbf{y}}_2 \leftarrow N_2(\mathbf{I})$ 
    Collect all feature tensors into buckets  $F_{N_1}$  and  $F_{N_2}$ .
    Compute feature difference loss,  $L_{FDL}$  from Eq. 44.
    BackpropagateAndUpdate $_{N_1, N_2}(-L_{FDL})$ 

     $\hat{\mathbf{y}}_c \leftarrow [N_1(\mathbf{I})^T, N_2(\mathbf{I})^T]^T$                                 /*Phase 4*/
     $\hat{\mathbf{y}}_E \leftarrow N_E(\hat{\mathbf{y}}_c)$ 
     $L_E \leftarrow L_X(\hat{\mathbf{y}}_E, \mathbf{y})$ 
    BackpropagateAndUpdate $_{N_E}(L_E)$ 

     $iter++$ 
end

```

to all other networks in the pool, and compare the mean squared difference between them in a pairwise fashion as per Eq. 43. This means that for N networks, the total number of transfers is ${}^N C_2$. This is crucially important as it ensures information exchange between all networks in the pool, which is the crux of our algorithm. The final FDL loss for N networks is the average over all ${}^N C_2$ transfers. Instead of maximizing the FDL loss, we minimize the negative of it.

$$L_{FDL}^* = \frac{1}{{}^N C_2} \sum_{i=1}^{N-1} \sum_{j=i+1}^N L_{FDL}^{i,j} \quad (47)$$

4.3.6 PHASE 4

Finally in the fourth phase of training, we train the ensemble head network N_E . The input to N_E is a concatenated array of all logits accumulated from all base networks. While training the head network, we do not pass the gradients back to the base networks. This is an important step required to measure the efficacy of the FDL loss. If the gradient that is computed at the ensemble end is used to update the weights of the base networks, then there is a possibility that the base networks improve the ensemble performance through this gradient update and not through FDL. Hence we choose to keep the weights of the base networks completely disjoint from the weights of the N_E network.

In none of the training phases, we optimize the ensemble’s performance by the loss computed at the ensemble head. Instead, we use feature exchange and FDL to allow the network to arrive at better optimas, by nudging it towards different solutions where the learnt feature representations are diverse. Each iteration comprises phase 1 through 4 exactly once. The process continues until convergence.

4.4 EXPERIMENTS

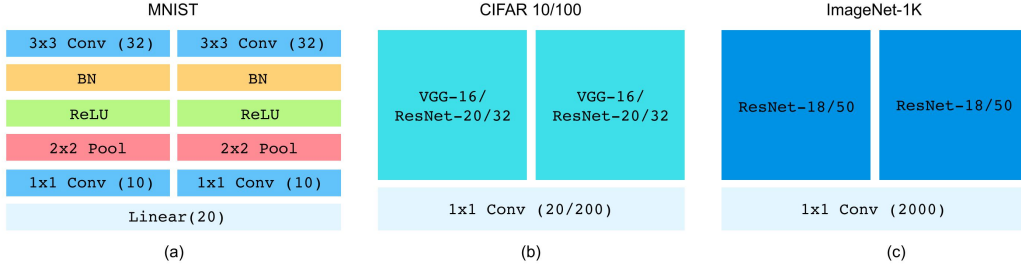


Figure 6: We perform experiments on the following models. **(a):** A simple two layer network on MNIST. **(b):** Deeper networks, VGG-16 and ResNet-20/32 on CIFAR-10/100. **(c):** Larger models: ResNet variants on ImageNet-1K.

Our first experiment consists of a simple network (Fig. 6(a)) that has a single convolution layer with M units of 3×3 filters. Training on the MNIST dataset, we vary M from 1 to 256, and record the performance of the single full-width

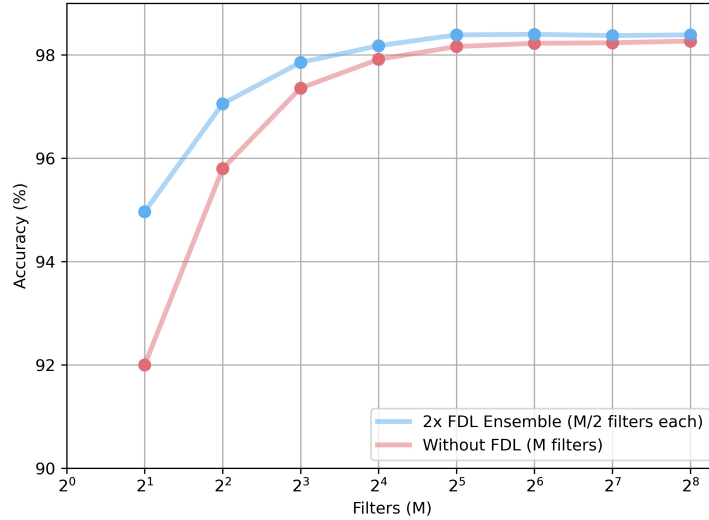


Figure 7: Number of Filters (M) vs Accuracy plot of a one layer ConvNet. Red line indicates single network of M filters. Blue line indicates FDL ensemble of 2x networks ($M/2$ filters each).

network (of M filters) and the ensemble performance of two half-width networks ($M/2$ filters each) (Fig. 7(a)). We observe that in every case, the two network FDL ensemble learns better representations and outperform the single full-width network.

We experiment with deeper networks, such as VGG-16 [Simonyan & Zisserman (2014)], ResNet-20 and ResNet-32 [He et al. (2016)] and with the CIFAR-10 and CIFAR-100 [Krizhevsky et al. (2009)] datasets. We use a single

Table 1: Results of experiments on a single layer ConvNet with M filters trained on MNIST. The middle column indicates accuracy results obtained with a single network of M filters. The right column indicates the accuracy results obtained with two networks with $M/2$ filters each, trained using feature difference loss across the two networks. Results indicate that two networks trained with FDL learns better representations, achieves higher accuracy and thus makes better use of the model capacity.

Filters	1x Network (%)	2x Half-Networks (%)
(M)	(M filters)	(M/2 filters each) + FDL
2	92.00 ± 1.41	94.97 ± 0.08
4	95.80 ± 0.59	97.06 ± 0.06
8	97.36 ± 0.22	97.86 ± 0.08
16	97.92 ± 0.06	98.18 ± 0.03
32	98.16 ± 0.05	98.39 ± 0.03
64	98.23 ± 0.07	98.40 ± 0.05
128	98.24 ± 0.05	98.38 ± 0.14
256	98.27 ± 0.05	98.39 ± 0.11

Table 2: Comparisons of ensemble methods in image classification task, performed on the CIFAR-10 and CIFAR-100 datasets.

Method	Accuracy(%)	
	CIFAR-10	CIFAR-100
VGG-16 (1x) baseline	93.66	74.61
VGG-16 RIE (2x)	93.7	76.95
VGG-16 SSE [Huang et al. (2017)]	94.05	75.31
VGG-16 FGE [Garipov et al. (2018)]	94.34	76.46
VGG-16 AE Full [Yang & Wang (2020)]	93.93	72.16
VGG-16 FDL (2x) [ours]	94.93	77.02

layer of 1×1 convolution units as the ensemble head, Fig. 6(b). From Table 2, we observe that ensembles trained with FDL outperform other methods. In CIFAR-10 experiments, FDL outperforms AE Full by 1.0%, SSE by 0.88%, FGE by 0.59% and improves over the single network baseline by 1.27%. In CIFAR-100 experiments, it outperforms AE Full by 4.86%, SSE by 1.71%, FGE by 0.56% and improves over the single network baseline by 2.41%. We additionally perform multi-network FDL ensembles with two, three and four networks with each of VGG-16, ResNet-20 and ResNet-32 on both CIFAR-10 and CIFAR-100 datasets. We portray the results in Fig. 7(b) and also in the supplementary section. For CIFAR-10, 4x FDL ensembles outperform the baseline by 1.56% for VGG-16, 2.24% for ResNet-20 and 1.67%. In case of CIFAR-100, 4x FDL ensembles outperform the baseline by 3.73% for VGG-16, 5.89% for ResNet-20 and 6.88% for ResNet-32.

On the ImageNet-1K [Russakovsky et al. (2015)] (Fig. 6(c)), we see a significant improvement of 1.58% in ResNet-18, and an improvement of 0.67%

Table 3: Comparisons of ensemble methods in image classification task, performed on the ImageNet-1K (right).

Method	Accuracy (%)
	ImageNet-1K
ResNet-50 (1x) baseline	76.38
ResNet-50 RIE (2x)	76.96
ResNet-50 SSE [Huang et al. (2017)]	76.67
ResNet-50 FGE [Garipov et al. (2018)]	76.69
ResNet-50 FDL (2x) [ours]	77.06

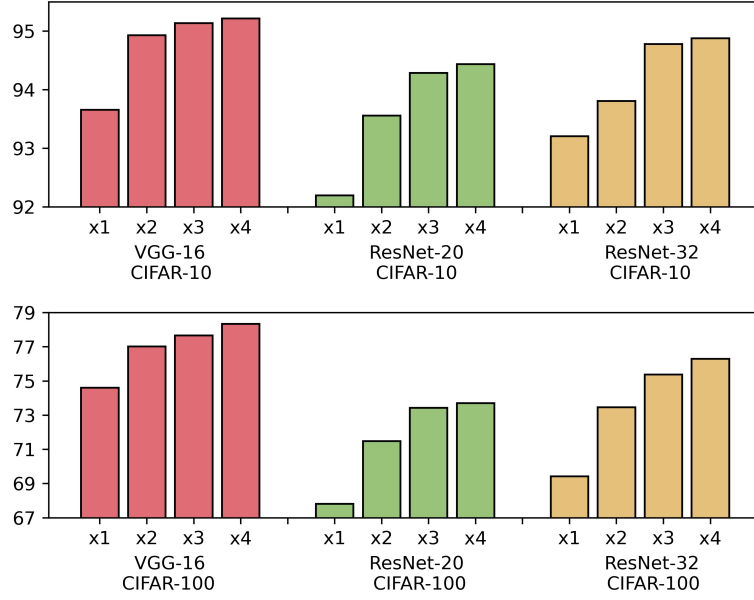


Figure 8: Multi-network FDL ensemble accuracies for various models (VGG-16, ResNet-20, ResNet-32) on (CIFAR-10 and CIFAR-100).

in ResNet-50 (Table 7). 2x ResNet-50 models trained with FDL outperforms SSE by 0.39% and FGE by 0.37% (Table 3).

Table 4: FDL Ensemble of multiple neural networks for the CIFAR-10 dataset. We report accuracy metrics of each base model along with the ensemble. Best individual and ensemble accuracies are marked in bold.

Model	CIFAR-10				FDL
	Network 1	Network 2	Network 3	Network 4	Ensemble
VGG-16 (x1)	93.66				93.66
VGG-16 (x2)	93.57	93.96			94.93
VGG-16 (x3)	93.53	93.93	93.60		95.14
VGG-16 (x4)	93.82	93.79	93.97	93.58	95.22
ResNet-20 (x1)	92.20				92.20
ResNet-20 (x2)	91.79	92.02			93.56
ResNet-20 (x3)	92.36	92.51	92.28		94.29
ResNet-20 (x4)	92.18	91.98	92.10	92.70	94.44
ResNet-32 (x1)	93.21				93.21
ResNet-32 (x2)	92.37	92.10			93.81
ResNet-32 (x3)	93.33	93.48	93.11		94.78
ResNet-32 (x4)	92.69	92.91	93.03	93.02	94.88

Table 5: FDL Ensemble of multiple neural networks for the CIFAR-100 dataset. We report accuracy metrics of each base model along with the ensemble. Best individual and ensemble accuracies are marked in bold.

Model	CIFAR-100				FDL
	Network 1	Network 2	Network 3	Network 4	Ensemble
VGG-16 (x1)	74.61				74.61
VGG-16 (x2)	74.27	74.44			77.02
VGG-16 (x3)	74.03	74.42	73.72		77.66
VGG-16 (x4)	74.82	73.88	73.73	74.26	78.34
ResNet-20 (x1)	67.82				67.82
ResNet-20 (x2)	67.63	68.26			71.48
ResNet-20 (x3)	67.57	67.57	68.50		73.43
ResNet-20 (x4)	67.11	67.76	67.48	67.71	73.71
ResNet-32 (x1)	69.42				69.42
ResNet-32 (x2)	69.25	68.80			73.46
ResNet-32 (x3)	69.40	69.70	68.77		75.38
ResNet-32 (x4)	69.20	69.60	69.45	69.10	76.30

Table 6: Hyperparameters for all experiments carried out. N = Number of networks in the ensemble. B = Batch Size. lr = Learning Rate. m = Momentum. Other hyperparameters that are common across all experiments are the following: Weight Decay = 5×10^{-4} . Nesterov momentum is used in all training sessions.

Model	Dataset	N	Network 1 Optimizer		Network 2 Optimizer		Network 3 Optimizer		Network 4 Optimizer		Similarity Optimizer		FDL Optimizer		Ensemble Head Optimizer	
			B	lr	m	lr	m	lr	m	lr	m	lr	m	lr	m	
VGG-16	CIFAR-10	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	-	-	-	1e-2	0.9	1e-5	0.85	1e-1	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-2	0.9	1e-1	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-4	0.85	1e-2
	CIFAR-100	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-5	0.85	1e-1	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-2	0.9	1e-1	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-4	0.85	1e-2
ResNet-20	CIFAR-10	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-4	0.85	1e-2	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-4	0.85	1e-2	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-4	0.85	1e-2
	CIFAR-100	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-5	0.85	1e-2	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-5	0.85	1e-2	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-5	0.85	1e-2
ResNet-32	CIFAR-10	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-4	0.85	1e-2	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-4	0.85	1e-2	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-4	0.85	1e-2
	CIFAR-100	x1	128	1e-2	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-5	0.85	1e-2	0.9
		x3	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	-	1e-2	0.9	1e-5	0.85	1e-2	0.9
		x4	128	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-2	0.9	1e-5	0.85	1e-2
ResNet-18	ImageNet	x1	256	1e-1	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	256	1e-1	0.9	1e-1	0.9	-	-	-	1e-5	0.9	2e-5	0.9	1e-4	0.9
ResNet-50	ImageNet	x1	256	1e-1	0.9	-	-	-	-	-	-	-	-	-	-	-
		x2	256	1e-1	0.9	1e-1	0.9	-	-	-	1e-5	0.9	1e-1	0.9	1e-4	0.9

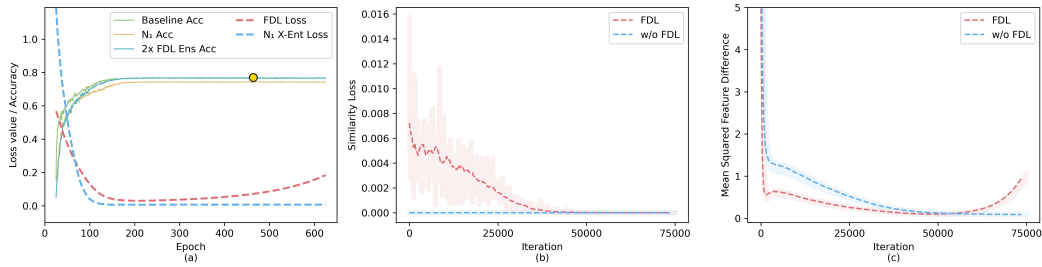


Figure 9: 2x VGG-16 ensemble training on CIFAR-100. (a) Loss and accuracy plots. (b) Similarity loss plots. (c) Plot of Mean Squared Feature Differences during the training.

4.5 ABLATION STUDIES

We discuss the efficacy of FDL loss by performing the same training routine of Fig. 5, with the same hyperparameters, but as two separate experiments: ‘with’ FDL and ‘without’ FDL. In Table 8, column [C] denotes the accuracies of ensembles of two identical base networks trained ‘with’ and ‘without’ FDL. Column [A] denotes the performance of each base network for which the ensemble in [C] is obtained. Column [B] denotes the individual best accuracy obtained during the entire training process. We observe that with FDL, not only the ensemble performance is better than without it, but also the base models trained with FDL outperform base models trained without FDL. We observe that FDL enables ensembles of two base ResNet models to achieve higher accuracies over the standard non-FDL methods. This indicates that FDL is a diversity encouraging loss function.

We also provide a visualization of feature difference maps between two VGG-16 models in Fig. 10 trained with and without FDL. We observe that FDL

Table 7: Test accuracies on the test set of ImageNet-1K. We observe that FDL ensemble performs better than the non-FDL ensemble.

Model	ImageNet Accuracy (%)		
	1x Network	2x Ensemble	2x FDL Ensemble
ResNet-18	70.012	71.014	71.594
ResNet-50	76.386	76.964	77.06

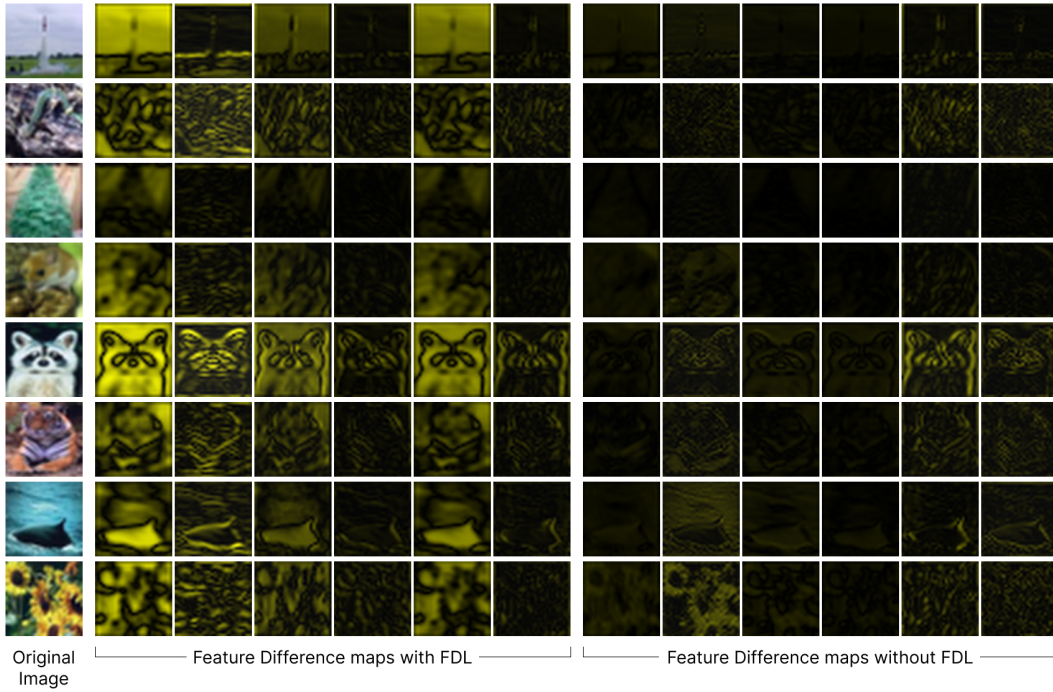


Figure 10: Differences between feature maps of two VGG-16 base models trained on the CIFAR-100 with and without FDL. The intensity of yellow on each feature map indicates the strength of the mean squared difference between two feature maps from the same layer and the same channel.

forces higher feature differences among base models, which we hypothesize as the primary contributor to increase in diversity, leading to the observed higher accuracies in FDL ensembles. In Fig. 9(a) we plot different loss plots and test accuracy plots for 2x VGG-16 ensemble on CIFAR-100. The FDL loss decreases initially (red line). After 200 epochs it starts to increase, even though the cross-entropy loss of N_1 keeps on decreasing (blue line). The yellow circle marks the epoch where maximum ensemble accuracy is achieved. In Fig. 9(b, c) we perform additional ablation studies, where we train the networks with FDL

Table 8: Results of ablation study performed on the ResNet-18 network on ImageNet. We observe that FDL ensemble performs better than the non-FDL ensemble.

Model	Network 1		Network 2		Ensemble
	[A]	[B]	[A]	[B]	[C]
ResNet-18	69.902	70.012	69.632	69.748	71.014
+ FDL	69.892	69.998	69.666	69.73	71.594
ResNet-50	76.042	76.386	75.97	76.306	76.964
+ FDL	76.186	76.246	76.052	76.108	77.06

and without FDL. In Fig. 9(b), we plot the similarity loss during training. We observe that in both cases the networks final attain close to zero similarity loss. This indicates that both networks achieve similar optimal points at convergence. However, in case of FDL, the initial few iterations the similarity loss is quite high and fluctuates rapidly. This indicates that with FDL the networks' states initially moves away from each other jumping across different local optimas as it explores the entire loss landscape. Whereas in the without-FDL scenario, the similarity loss is very close to zero right from the beginning of training. In Fig. 9(c) we plot the Mean Squared Feature Differences during training for both 'with' and 'without' FDL cases. We observe that FDL loss steadily decreases to zero if the networks are not trained with it (blue line), but increases later down during training (red line).

We also performed an ablation study with and without similarity loss. We found that without similarity loss, the overall training objective does not incentivize that all networks in the ensemble perform *similarly*. If we invoke FDL in this scenario, we see that in a two network scenario, the network that dominates initially in training, goes on to converge properly, while the other network goes on to attain weight values that are very close to zero. This is intuitive because FDL maximizes the feature differences, and one solution to maximizing feature differences is to make sure that one of the base model does not learn anything while the other network learns something. Thus the introduction of similarity loss is essential in maintaining training stability and balancing out the instability that is caused by a negative loss function (FDL).

4.6 DISCUSSIONS

In conventional ensemble methods, multiple neural networks are trained independently. The outcomes from these networks are then merged, either by voting or linear combination. However, this standard approach has a limitation: it does not

promote information sharing among the base networks within the ensemble. For neural network ensembles to reach optimal performance, there's a need to diversify their representations. We emphasize that simply training multiple networks and merging their outcomes isn't sufficient. There's a need for these networks to have diverse representations to maximize the ensemble's effectiveness. We introduce a novel technique called Feature Difference Loss (FDL). This technique aims to enhance the learned representations within an ensemble. It does so by promoting feature sharing during the training phase. In essence, FDL encourages networks within the ensemble to acquire distinct features. By ensuring that different networks focus on different features, FDL directly enhances model diversity. Initial tests conducted with ensembles of two to four networks show significant improvements over other ensemble methods. This boost in performance is especially evident in datasets with fewer samples per category, such as MNIST, CIFAR-10, and CIFAR-100. Even on extensive datasets like ImageNet, there were noticeable positive outcomes when using FDL.

In the realm of neural network ensembles, the traditional approach of independently training multiple networks and subsequently merging their outcomes has shown its limitations, especially in terms of promoting information sharing among ensemble members. Recognizing the critical role that diversified representations play in boosting ensemble performance, this section has introduced a promising technique, the Feature Difference Loss (FDL). FDL is not just another ensemble method but a paradigm shift in how we approach ensemble training. By actively encouraging networks within the ensemble to focus on distinct features, FDL directly tackles the challenge of enhancing model diversity. This not only optimizes the ensemble's overall performance but also ensures that each member network contributes uniquely to the ensemble's decisions. Tests have showcased the efficacy of FDL, especially in datasets with fewer samples per category. Its positive impact on extensive datasets like ImageNet further underscores its robustness and

potential for broader applications. As the field of neural networks continues to evolve, techniques like FDL underline the importance of continuous innovation. By promoting diversified knowledge and collaborative learning within ensembles, we move closer to achieving neural networks that are both diverse in knowledge and collaborative in approach.

5 POWERGRAD TRANSFORM

In standard neural network training, the gradients in the backward pass are determined by the forward pass. As a result, the two stages are coupled. This is how most neural networks are trained currently. However, gradient modification in the backward pass has seldom been studied in the literature. In this work we explore decoupled training, where we alter the gradients in the backward pass. We propose a simple yet powerful method called PowerGrad Transform, that alters the gradients before the weight update in the backward pass and significantly enhances the predictive performance of the neural network. PowerGrad Transform trains the network to arrive at a better optima at convergence. It is computationally extremely efficient, virtually adding no additional cost to either memory or compute, but results in improved final accuracies on both the training and test sets. PowerGrad Transform is easy to integrate into existing training routines, requiring just a few lines of code. PowerGrad Transform accelerates training and makes it possible for the network to better fit the training data. With decoupled training, PowerGrad Transform improves baseline accuracies for ResNet-50 by 0.73%, for SE-ResNet-50 by 0.66% and by more than 1.0% for the non-normalized ResNet-18 network on the ImageNet classification task.

5.1 MOTIVATION

We introduced the notion of learning capacity in section 1.1 and in Chapter 4, we explored methods to improve inter-network learning capacity. In this chapter,

we address the critical problem of neural network training, i.e. we find why some networks don't learn as good as other networks (non-BatchNormalized networks vs. BatchNormalized networks), we explore why even BatchNormalized networks under-utilize their inherent learning capacities, and we propose an alternative training methods to modify gradients during the learning process of a neural network and by doing so, allow it to explore parts of the loss landscape that would otherwise remain unexplored with traditional way of training.

5.2 INTRODUCTION

Backpropagation is traditionally used to train deep neural networks [Lillicrap et al. (2020)]. Gradients are computed using basic calculus principles to adjust the weights during backpropagation [LeCun et al. (1988)]. Alternatives to traditional gradients has rarely been studied in the literature hitherto. In normal training procedures, gradients are computed immediately in the backward pass utilizing the values obtained in the forward pass. This makes the backward pass coupled with the forward pass. However, decoupling the backward pass from the forward pass by modifying the gradients to improve training efficiency and final convergent accuracy has hardly been explored. In this work, we explore the landscape of decoupling the forward pass from the backward pass by altering the gradients and subsequently updating the network's parameters with the modified gradients. There are several ways to achieve gradient modification in the backward pass. We discuss a few techniques in Fig. 11. One could modify the gradients at either **(I)** multiple points in the backward computation graph or **(II)** at a point very early in the backward graph and allow this changed gradient to also affect the rest of the network's gradients as it backpropagates through the backward computation graph.

Type 0: No modification: In this method, we calculate the gradients using the standard calculus rules and use the chain rule to calculate the gradients of the rest

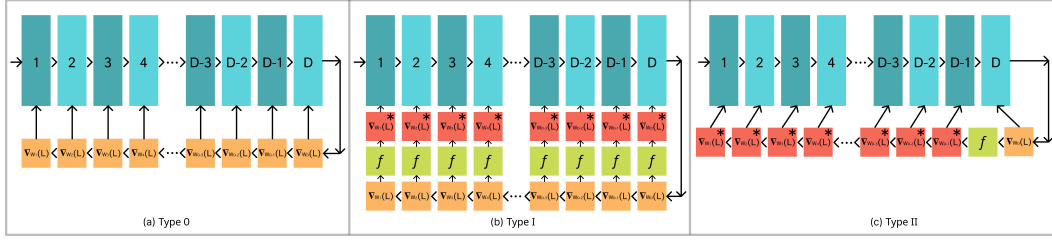


Figure 11: Different ways of altering gradients in the backward pass. Blue blocks denote a CNN’s different layers (convolutional, linear, batchnorm, softmax, etc.). Orange blocks indicate the backward graph with unmodified gradients. Green blocks represent transformation functions, while red blocks indicate transformed gradients. (a) **Type 0**: Backpropagation without any gradient modification. (b) **Type I**: Altering gradients after they are all computed. (c) **Type II**: Altering gradients at an earlier stage in the backward computation graph, subsequently leading to alteration of all other gradients.

of the network’s parameters, also known as backpropagation as portrayed in Fig. 11(a). The network is then updated with the gradient descent equation:

$$W_i^{t+1} = W_i^t - \lambda \nabla_{W_i}(L) \quad i = D, D-1, \dots, 1 \quad (48)$$

Type I: Gradient modification at multiple points independently: In this method, the gradients are first computed using standard procedure and then individually altered as shown in Fig. 11(b). Gradient clipping [Pascanu et al. (2013)] and Adaptive gradient clipping [Brock et al. (2021)] are examples of such modifications. In both of these methods, the gradients are first computed using standard rules and then they are modified using some function. It can be described as:

$$W_i^{t+1} = W_i^t - \lambda f(\nabla_{W_i}(L)) \quad i = D, D-1, \dots, 1 \quad (49)$$

where the gradients $\nabla_{W_i}(L)$ are transformed using the transformation function ‘ f ’ before the weight update.

Type II: Gradient modification at a point very early in the backward graph: In this type of modification, the gradient is altered at a very early stage in the backward computation graph and then all subsequent gradients are generated using the values obtained with the modified gradients. We illustrate this type of modification in Fig.

11(c). Because of the chain rule, network parameters whose gradients are connected to the point of alteration in the computation graph also gets subsequently altered. It can be described as:

$$W_D^{t+1} = W_D^t - \lambda f(\nabla_{W_D}(L)) \quad (50)$$

$$W_i^{t+1} = W_i^t - \lambda \nabla_{W_i}(L)^* \quad i = D - 1, \dots, 1 \quad (51)$$

where first the gradient $\nabla_{W_D}(L)$ is transformed using the transformation function ‘ f ’ and then this transformed gradient is propagated through the rest of the backward graph. All other gradient vectors $\nabla_{W_i}(L)^*$ are computed as is, but because of the early injection of the transformed gradient $\nabla_{W_D}(L)$, all other gradient vectors that are connected to the transformed gradient through the chain rule ($\nabla_{W_i}(L)^*$, $i=D-1, \dots, 1$), gets subsequently altered.

Type I modification is computationally more expensive than Type II modification as it requires altering the gradients of each and every parameter individually. Type II modification recomputes gradients at each and every location through the natural flow of backpropagation. We propose PowerGrad Transform (PGT) which is a type II gradient modification method. With virtually no additional compute or memory requirement, PowerGrad Transform enhances classification performance of a convolutional neural network, by leading the network to arrive at a higher quality optima at convergence, at which both training and test accuracies improve.

PowerGrad Transform modifies the gradients at the softmax layer, which is the earliest part of most convolutional neural network’s backward computation graph. By changing the gradient at this stage, all other gradients for all other parameters in the network such as the linear layer, batchnorm statistics and convolutional layer’s filter parameters are affected. We analyze and explore the behaviour of such

alteration both mathematically and through empirical experiments conducted on networks without batch-normalization. Experimentally, we see that PGT improved training and test accuracies for both normalized and non-normalized networks. PGT improves a network’s learning capacity and results in a better fit of the training data.

The following are the major contributions in this regard:

1. We introduce PowerGrad Transform, which decouples the backward and the forward passes of neural network training and enables a considerably better fit to the dataset, as assessed by both training and test accuracy measures. PGT is a performance enhancement method that alters the gradients in the backward pass before the update step leading to accelerated training and a significant boost in the network’s predictive performance.
2. We perform theoretical analysis of the properties of the PowerGrad transformation (section 5.4.1) and explore its effect on weight parameters and gradients (section 5.4.3), logits (section 5.4.2), (section 5.5.3), loss values (section 5.5.3) and class separation (section 5.4.4).
3. We study experimentally the degenerate behaviour of non-BN networks that is often seen during the normal training procedure, particularly at larger batch sizes. With PGT, we see improved gradient behaviour and a decreased likelihood of the weights attaining degenerate states.
4. We provide complete results from a variety of models (non-BN and BN ResNets, SE-ResNets) using the ImageNet dataset. It helps the network to improve its learning capabilities by locating a more optimum convergence point. Additionally, we conduct an ablation study and compare its impacts to those of regularization approaches such as label smoothing.

We examine techniques that are related to gradient modification and emphasize the distinction between them and our proposed method.

As described in Chapter 3, Gradient Clipping (GC) and Adaptive Gradient Clipping (AGC) are the two primary methods that has been used in the literature hitherto to perform gradient modifications in the backward pass with limited success. We show that our method, PowerGrad Transform (PGT) can surpass when used both independently and in conjunction with AGC. We demonstrate in our ablation studies, GC is not as effective in improving the performance of non-normalized networks. AGC performs better than GC in non-normalized networks. However, we show that PGT outperforms both in such networks. Our experiments demonstrate that PGT outperforms AGC, when used independently. When used together, it can further improve a network’s performance (section 5.5.2).

With regards to label smoothing, PowerGrad Transform works in a much different way. PGT does not smooth the ground truth labels, rather it modifies the gradient by smoothing the predicted probability vector. As we show in section 5.4, our proposed transformation has the opposite effect of label smoothing. From our studies, we show that PGT in fact leads to larger generalization gap between the training and test dataset, although it does lead to an increase in both training and test accuracies at convergence. Coupled with PGT, label smoothing can narrow the generalization gap, while retaining the benefits of PGT.

The key difference between PGT and distillation methods is that in the latter, the transformation is applied in the forward pass, while PGT is a backward pass modification only. Also, in distillation settings the temperature parameter is a part of the network’s computation graph. In the case of PowerGrad Transform, we directly tamper the gradients without introducing any change in the forward pass. It is a more generalized way of arriving at altered gradients in the backward pass. Moreover we show that modifying the gradients in the backward pass can improve performance in many cases. The rise in training accuracy indicates that PGT enables the network to learn more from the training data.

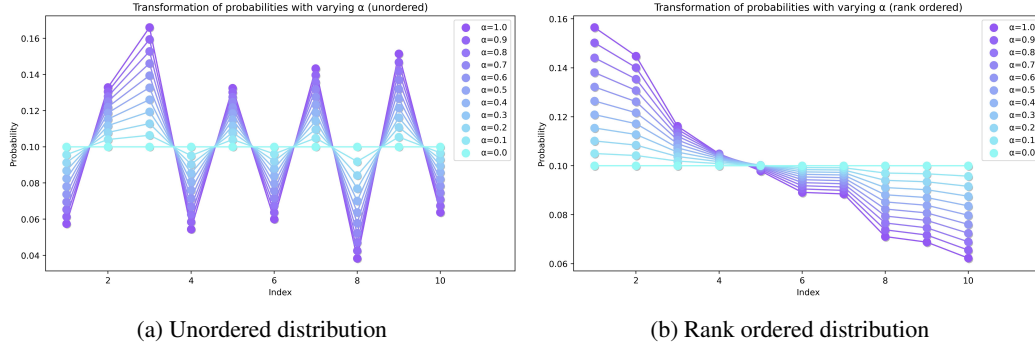


Figure 12: Illustration of the impact of PowerGrad Transform on probability values of an arbitrary distribution (number of classes, $C = 10$). $\alpha = 1$ indicates the unmodified distribution. As α reduces from 1 to 0, the probability distributions (left: unordered or right: ordered) becomes flatter and flatter, approaching the uniform distribution of $1/C$ for every index.

5.3 METHOD

The PowerGrad Transform technique is described in this section. It is a general technique and can be applied to any neural network. A neural network with parameters W generates C logits denoted by z for every input vector x . z is given as $z = Wx$. Then a set of probability values p_i are generated from the logits using a softmax function which is defined as $p_i = \frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}}$. p_i and z_i represent the predicted probability values and the logits for the i^{th} class respectively. Following this step, the loss function is invoked and the loss between the predicted probability values the ground truth labels (which is also a probability distribution) is calculated. If the loss function is cross-entropy loss, then the value of the loss is given as $L = -\sum_{i=1}^C q_i \log(p_i)$ where q_i is the ground truth label of the i^{th} class for a particular training example. By standard gradient update rule, we can calculate the gradient of the loss with respect to the logits which takes the form $\frac{\partial L}{\partial z_i} = p_i - q_i$.

The PowerGrad Transform technique is now described. We introduce a hyperparameter α , which takes a value between $[0, 1]$ and regulates the degree of gradient modification. Fig. 12 shows the effects of the transform under different values of α for different distributions. The PowerGrad Transform method modifies the predicted probability values in the backward pass as follows:

$$p'_i = \frac{p_i^\alpha}{\sum_{j=1}^C p_j^\alpha} \quad i = 1, \dots, C \quad 0 \leq \alpha \leq 1 \quad (52)$$

The above transformation changes the gradient of the loss with respect to the logits as follows:

$$\widehat{\frac{\partial L}{\partial z_i}} = p'_i - q_i \quad (53)$$

The rest of the backward pass proceeds as usual. We denote the original probability distribution as P (with values p_i at the i^{th} index) and the transformed distribution as P' (with values p'_i at the i^{th} index). An algorithm portraying the forward and backward process is shown in Algorithm 2 and 3 respectively.

Algorithm 2: PGT: Forward pass

Given: Neural network forward function F_W with weights W

Data: I = Image.

$p_i \leftarrow F_W(I), \quad i = 1, \dots, C \quad /*\text{Compute probabilities}*/$

Algorithm 3: PGT: Backward pass

Given: Neural network forward function F_W with weights W

Data: I = Image, q = Labels vector, α = PGT Hyperparameter, λ = Learning Rate.

while $iter \leq iter_{max}$ **do**

$p_i \leftarrow F_W(I), \quad i = 1, \dots, C \quad /*\text{Compute probabilities}*/$

$p'_i \leftarrow \frac{p_i^\alpha}{\sum_{j=1}^C p_j^\alpha}, \quad i = 1, \dots, C \quad /*\text{Transform probabilities}*/$

$\widehat{\frac{\partial L}{\partial z_i}} \leftarrow p'_i - q_i, \quad i = 1, \dots, C \quad /*\text{Compute gradients}*/$

$\widehat{\frac{\partial L}{\partial W}} \leftarrow \frac{\partial z}{\partial W} \widehat{\frac{\partial L}{\partial z}} \quad /*\text{Backprop}*/$

$W \leftarrow W - \lambda \widehat{\frac{\partial L}{\partial W}} \quad /*\text{Update weights}*/$

$iter++$

end

5.4 ANALYSIS

In this section, we first describe the properties of the PowerGrad Transform (PGT) and then we highlight how these suitable properties of PGT helps the neural network to improve the performance. We use the same setup as described in section 5.3. To explore the properties of PGT, we start by investigating the effect of the transform on the softmax probabilities.

5.4.1 PROPERTIES OF PGT

Lemma 1. For any arbitrary probability distribution P with probability values given by p_i for $i = 1, \dots, C$, the corresponding transformed probability values p'_i given by [Eq. 52] has a threshold $\left(\sum_{j=1}^C p_j^\alpha\right)^{\frac{1}{\alpha-1}}$ and

$$\begin{aligned} p'_i &\geq p_i, \text{ if } p_i \leq \left(\sum_{j=1}^C p_j^\alpha\right)^{\frac{1}{\alpha-1}} \\ p'_i &< p_i, \text{ if } p_i > \left(\sum_{j=1}^C p_j^\alpha\right)^{\frac{1}{\alpha-1}} \end{aligned} \tag{54}$$

Proof. The proof follows directly from the definition of p'_i . We notice that if $p'_i < p_i$, we get:

$$\frac{p_i^\alpha}{\sum_{j=1}^C p_j^\alpha} < p_i \tag{55}$$

$$\Rightarrow p_i^{\alpha-1} < \sum_{j=1}^C p_j^\alpha \tag{56}$$

$$\Rightarrow p_i > \left(\sum_{j=1}^C p_j^\alpha\right)^{\frac{1}{\alpha-1}} \tag{57}$$

□

We call this threshold, the *stationary threshold*. The stationary threshold is that value of p_i that does not change after the transformation. Therefore, when p_i is greater than the *stationary threshold*, $p'_i < p_i$.

Proposition 1. At $\alpha = 0$, the stationary threshold equals $1/C$ and all values of the transformed distribution p'_i reduces to the uniform distribution for $i = 1, \dots, C$.

Proof. From Eq. (54), we see that the stationary threshold at $\alpha = 0$ is $1/C$. Also, following from the definition of the transformed probabilities (Eq. 52) we conclude that if $\alpha = 0$, then all values of p'_i are $1/C$. Therefore the transformed distribution at $\alpha = 0$ is a uniform distribution.

Since we have established that values of p_i which are greater than the stationary threshold decreases and move down towards the stationary threshold, and values in p_i lower than the stationary threshold moves up towards the stationary threshold, this transformation makes the distribution more uniform (i.e. it smooths out the actual distribution) as α is decreased from 1 and down towards 0. This final observation we prove in the following theorem.

Theorem 1. For any arbitrary probability distribution P with probability values p_i for $i = 1, \dots, C$, the stationary threshold of the transformed distribution P' with probability values $p'_i = \frac{p_i^\alpha}{\sum_{j=1}^C p_j^\alpha}$, $0 \leq \alpha \leq 1$ is a monotonically non-decreasing function with respect to α .

Proof. To prove monotonicity, we first compute the gradient of the stationary threshold with respect to the variable in concern, α .

$$\frac{\partial}{\partial \alpha} \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} = \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} \left(\frac{\sum_{j=1}^c p_j^\alpha \log(p_j)}{(\alpha-1) \sum_{j=1}^c p_j^\alpha} - \frac{\log \left(\sum_{j=1}^c p_j^\alpha \right)}{(\alpha-1)^2} \right) \quad (58)$$

$$= \frac{1}{\alpha(\alpha-1)^2} \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} \left(\frac{(\alpha-1) \sum_{j=1}^C p_j^\alpha \log(p_j)}{\sum_{j=1}^c p_j^\alpha} - \alpha \log \left(\sum_{j=1}^c p_j^\alpha \right) \right) \quad (59)$$

If $a_1, \dots, a_n$ and $b_1, \dots, b_n$ are non-negative numbers, then using the log sum inequality,

we get $\sum_{j=1}^n a_j \log \left(\frac{a_j}{b_j} \right) \geq \left(\sum_{j=1}^n a_j \right) \log \left(\frac{\sum_{j=1}^n a_j}{\sum_{j=1}^n b_j} \right)$. Setting $a_j = p_j^\alpha$ and $b_j = 1$, we get the following lower bound

$$\sum_{j=1}^C p_j^\alpha \log(p_j^\alpha) \geq \left(\sum_{j=1}^C p_j^\alpha \right) \log \left(\frac{1}{C} \sum_{j=1}^C p_j^\alpha \right) \quad (60)$$

Substituting (60) in (59), we get:

$$\frac{\partial}{\partial \alpha} \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} \geq \frac{1}{\alpha(\alpha-1)^2} \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} \left((1-\alpha) \log(C) - \log \left(\sum_{j=1}^c p_j^\alpha \right) \right) \quad (61)$$

p^α is concave, and so by Jensen's inequality we get the following upper bound for the second term:

$$\left(\frac{1}{C} \sum_{j=1}^C p_j \right)^\alpha \geq \frac{1}{C} \sum_{j=1}^C p_j^\alpha \quad (62)$$

$$\Rightarrow \log \left(\sum_{j=1}^C p_j^\alpha \right) \leq (1-\alpha) \log(C) \quad (63)$$

Substituting (63) in (61),

$$\frac{\partial}{\partial \alpha} \left(\sum_{j=1}^c p_j^\alpha \right)^{\frac{1}{\alpha-1}} \geq 0 \quad (64)$$

□

We conclude from the analysis that the stationary threshold is a monotonic non-decreasing function with respect to α . Also the derivative of PGT function with respect to the true probabilities is non-negative which in turn means that the transformation is an order-preserving map. All values greater than the threshold move towards the threshold after transformation and all values below the threshold also move towards the threshold, and the threshold itself moves monotonically towards $1/C$ as α is decreased from 1 to 0. This concludes that the transformation smooths the original distribution.

5.4.2 EFFECT ON LOGITS

We know that neural networks uses the softmax function to generate prediction probabilities from logits in multi-class classification tasks. If we use cross-entropy loss to train the network then the partial derivative of the loss w.r.t. i^{th} logit is dependent on the value of the predicted probability and the value of the class label (either 0 or 1) of the i^{th} logit.

$$\frac{\partial L}{\partial z_i} = p_i - q_i \quad (65)$$

In general, the range of the $|\frac{\partial L}{\partial z_i}|$ is $[\delta, \epsilon]$ where, $\delta \rightarrow 0$ and $\epsilon \rightarrow 1$ in traditional training (Eq. 65) procedures of neural networks. Here, we propose a new method of neural network training which alters the gradients at the time of backward pass by modifying the predicted probability vector using PGT (Eq. 53) such that the directional derivative for each logit does not become too small or too large,

With inclusion of PGT in the last layer, the range of the $\widehat{|\frac{\partial L}{\partial z_i}|}$ becomes $[\delta^+, \epsilon^-]$ where, $\delta^+ > \delta$ and $\epsilon^- < \epsilon$

Now, we demonstrate how PGT restricts the magnitude of the directional derivative for each logit. For a training example, a neural network can assign the highest probability to either (i) a wrong class or to the (ii) correct class.

(i) Wrong Class Predicted: If the network assigns the highest probability for any class except the true class, then wrong class assignment happens. We analyze the effect on gradient due to PGT for this case. There are three possibilities

(a) Let, i^{th} class be the class for which the network assigns low probability, but it is the true class, then

$$p_i < p_{th} \Rightarrow p'_i > p_i \Rightarrow \widehat{\frac{\partial L}{\partial z_i}} = p'_i - 1 > p_i - 1 ; \text{ where } p_i \rightarrow 0 \text{ \& } q_i = 1$$

$$\text{So, } \widehat{|\frac{\partial L}{\partial z_i}|} = |p'_i - 1| < |p_i - 1| \Rightarrow \widehat{|\frac{\partial L}{\partial z_i}|} < \epsilon \text{ where; } \epsilon \rightarrow 1$$

(b) Let, j^{th} class be the class (not true class) for which the network assigns the highest probability, then

$$p_j > p_{th} \Rightarrow p'_j < p_j \Rightarrow \widehat{\frac{\partial L}{\partial z_j}} = p'_j < p_j ; \text{ where } p_j \rightarrow 1 \text{ \& } q_j = 0$$

$$\text{So, } \widehat{|\frac{\partial L}{\partial z_j}|} < \epsilon \text{ where; } \epsilon \rightarrow 1$$

(c) Let k denote the index for all those classes for which the predicted probability is low and also the ground truth class label is 0, then

$$p_k < p_{th} \Rightarrow p'_k > p_k \Rightarrow \widehat{\frac{\partial L}{\partial z_k}} = p'_k > p_k ; \text{ where } p_k \rightarrow 0 \text{ \& } q_k = 0$$

$$\text{So, } \widehat{|\frac{\partial L}{\partial z_k}|} > \delta \text{ where; } \delta \rightarrow 0$$

(ii) Correct Class Predicted: For a given training example, if the network assigns the highest probability to the true class then it assigns correct class to the training example. Now we observe the effect on gradient due to PGT for this case. There are two possibilities,

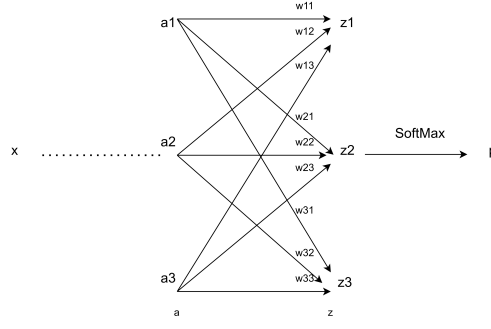


Figure 13: Fully connected layer of a convolutional neural network

(a) Let, i^{th} class be the class for which the network assigns the highest probability and it is also the true class, then

$$p_i > p_{th} \Rightarrow p'_i < p_i \Rightarrow \widehat{\frac{\partial L}{\partial z_i}} = p'_i - 1 < p_i - 1 ; \text{ where } p_i \rightarrow 1 \ \& \ q_j = 1$$

$$\text{So, } \left| \widehat{\frac{\partial L}{\partial z_i}} \right| = |p'_i - 1| > |p_i - 1| \Rightarrow \left| \widehat{\frac{\partial L}{\partial z_i}} \right| > \delta \text{ where; } \delta \rightarrow 0$$

(b) Let, k^{th} index is used for all those classes for which the predicted probability is low and also the actual label value 0, then

$$p_k < p_{th} \Rightarrow p'_k > p_k \Rightarrow \widehat{\frac{\partial L}{\partial z_k}} = p'_k > p_k ; \text{ where } p_k \rightarrow 0 \ \& \ q_k = 0$$

$$\text{So, } \left| \widehat{\frac{\partial L}{\partial z_k}} \right| > \delta \text{ where; } \delta \rightarrow 0$$

After analyzing all these cases ((i)a,b,c & (ii)a,b), we are able to deduce that PGT helps to restrict the directional derivative to be within limit such that for every logit/direction the directional derivative is not very small or very large.

5.4.3 EFFECT ON WEIGHT GRADIENTS

In this section, we discuss the impact of PGT on weight gradients. Let the input and output of the last convolutional layer of a convolutional neural network be $\mathbf{x}$ and $\mathbf{a}$ respectively (Fig. 13). The last layer which is a fully connected layer, produces logits $\mathbf{z}$ by combining $\mathbf{a}$ with weights and thereafter the network generates the predicted probability vector $\mathbf{p}$ from the logits using softmax activation function. We note that $a_i \geq 0 \ \forall i$ as, $\mathbf{a}$ is the output of ReLU activation functions.

Here, $z_i = \sum_{j=1}^C w_{ij}a_j$, so $\frac{\partial L}{\partial w_{ij}} = a_j \frac{\partial L}{\partial z_i}$. However, if we apply PGT while training the neural network, then $\widehat{\frac{\partial L}{\partial w_{ij}}} = a_j \widehat{\frac{\partial L}{\partial z_i}}$

If the activation of the j^{th} neuron is zero, i.e. $a_j = 0$ then there is zero gradient for all weights which acts on a_j both for normal training and training with PGT, as a_j has no contribution in the computation of logits. However, the interesting thing is to observe the effect of PGT when $a_j > 0$. As, the activation values (a_j) are positive, so $\widehat{\frac{\partial L}{\partial z_i}} > \frac{\partial L}{\partial z_i} \Rightarrow \widehat{\frac{\partial L}{\partial w_{ij}}} > \frac{\partial L}{\partial w_{ij}} \forall j \in (1, 2, \dots, C)$ and similar relationship for $a_j < 0$. With PGT, the weight updating rule for $(t + 1)^{th}$ iteration then becomes,

$$w_{ij}^{t+1} = w_{ij}^t - \eta \frac{\widehat{\frac{\partial L}{\partial w_{ij}}}}{\frac{\partial L}{\partial w_{ij}^t}} \quad (66)$$

The weight update equation using PGT, can also be viewed in terms of the following equation,

$$w_{ij}^{t+1} = w_{ij}^t - \frac{\eta}{C_{PGT}} \frac{\partial L}{\partial w_{ij}^t} \quad (67)$$

For neural networks, the exponential nature of the softmax function produces sharp distribution for predicted probability vector. For example, if $\mathbf{z}$ is $[20, 30, 10]$ then $\mathbf{p}$ is equal to $[4 \times 10^{-5}, 0.99995, 2 \times 10^{-9}]$. When the network is at the initial phase of training, most of the training examples are misclassified. Under this scenario, PGT helps to update the weights in a better manner than traditional gradients. Suppose the label vector $\mathbf{q}$ is $[1, 0, 0]$ then it indicates that z_1 is the logit corresponding to ground truth class and z_2 is the logit where the network has assigned the highest probability. Similarly, z_3 is the logit where the network has assigned a low probability as well as the label vector is also 0 for this logit. Here $\frac{\partial L}{\partial w_{1j}} = \epsilon a_j > 0 \Rightarrow \widehat{\frac{\partial L}{\partial w_{1j}}} < \frac{\partial L}{\partial w_{1j}^t} \Rightarrow C_{PGT} > 1 \forall j \in (1, 2, \dots, C)$. Advanced gradient updation algorithms like AdaGrad [Duchi et al. (2011)] reduces the gradient in the direction where the directional derivative for a weight is high. It reduces the learning rate for the weight so that the gradient in that direction is not too large in order to make the update process less volatile. The formulation

of PGT also implicitly reduces the partial derivative in the same manner for the high derivative directions, the only difference with the AdaGrad is that C_{PGT} depends only on the current iteration's gradient and not on previous iterations like AdaGrad. Similarly we can observe that for this example, $\frac{\partial L}{\partial w_{3j}^t} = \delta a_j \approx 0 \Rightarrow \widehat{\frac{\partial L}{\partial w_{3j}^t}} > \frac{\partial L}{\partial w_{3j}^t} \Rightarrow C_{PGT} < 1 \forall j \in (1, 2, \dots, C)$. As directional derivative for the weights associated with third logit is extremely low, PGT helps to increase the derivative in these directions so that the training process does not become too slow.

5.4.4 EFFECT ON CLASS SEPARATION

During final iterations of training, we analyze the iteration-wise gradient behaviour where the network assigns the highest probability to the correct class for a given training example; we deduce the effects of PGT in the next immediate training iteration. For example, if $\mathbf{z}$ is $[30, 20, 10]$ then $\mathbf{p}$ is $[0.99995, 4 \times 10^{-5}, 2 \times 10^{-9}]$ and $\mathbf{q}$ is $[1, 0, 0]$. There is no change in the weight update due to PGT when $a_j = 0$. However, when $i \neq 1$ and $a_j > 0$, $|\widehat{\frac{\partial L}{\partial w_{1j}^t}}| > |\frac{\partial L}{\partial w_{1j}^t}|$ and $\widehat{\frac{\partial L}{\partial w_{ij}^t}} > \frac{\partial L}{\partial w_{ij}^t}$. Here in this example $\frac{\partial L}{\partial z_1^t} < 0 \Rightarrow \frac{\partial L}{\partial w_{1j}^t} < 0 \Rightarrow \widehat{w_{1j}^{t+1}} > w_{1j}^{t+1} > w_{1j}^t \forall j$. We also know that $z_i^{t+1} = \sum_{j=1}^C w_{ij}^{t+1} a_j$ and $\widehat{z_i^{t+1}} = \sum_{j=1}^C \widehat{w_{ij}^{t+1}} a_j$. It indicates that $\widehat{z_1^{t+1}} > z_1^{t+1} > z_1^t$ as all the weights associated with first logit is larger while using PGT in $(t+1)^{th}$ iteration. Similarly, for the non-true classes $\widehat{w_{ij}^{t+1}} < w_{ij}^{t+1} < w_{ij}^t$ when $i \neq 1 \forall j$ as $\frac{\partial L}{\partial z_i^t} > 0$. So in $(t+1)^{th}$ iteration, $\widehat{z_i^{t+1}} < z_i^{t+1} < z_i^t$ when $i \neq 1$. We observe that $dist(\widehat{z_1^{t+1}}, \widehat{z_i^{t+1}}) > dist(z_1^{t+1}, z_i^{t+1})$ where the first logit corresponds to the true class and all other logits indexed by i are non-true classes. Therefore the distance between correct and incorrect class logits increases due to PGT and this leads to better class separation.

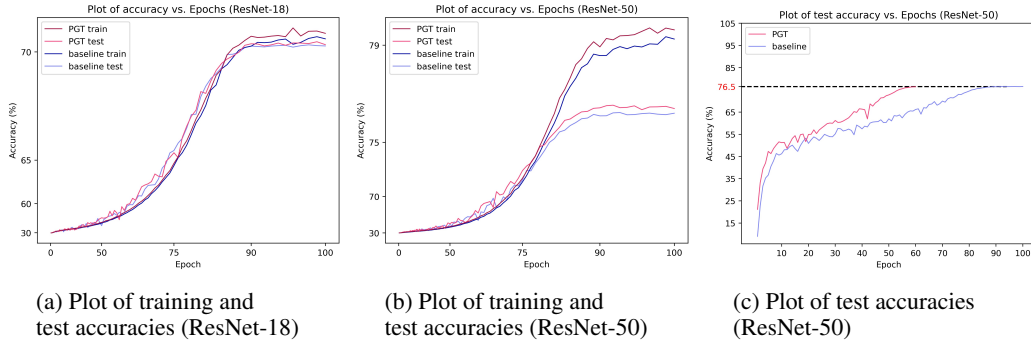


Figure 14: Log-linear plots of training and test accuracies and comparison with baseline of batch-normalized variants: **(a)** ResNet-18 ($\alpha = 0.25$), **(b)** ResNet-50 ($\alpha = 0.3$). For better visualization of accuracies obtained in the final iterations, we plot both axes in log scale. PowerGrad Transform improves upon the baseline in all cases. The corresponding accuracies are mentioned in Table 9. **(c)** Training speed comparison between PGT (60 epochs) and baseline (100 epochs). They both converge to the same test accuracy (76.5%) on ImageNet-1K. PGT’s accelerated training saves 40% of the epoch budget.

5.5 EXPERIMENTS

5.5.1 BATCHNORMALIZED NETWORKS

We perform experiments on different variants ResNets using the ImageNet-1K dataset [Deng et al. (2009)]. All models are trained on four V100 GPUs with a batch size of 1024. We utilize a common set of hyperparameters for all experiments, which are as follows: 100 epoch budget, 5 epochs linear warmup phase beginning with a learning rate of 4×10^{-4} and ending with a peak learning rate of 0.4, a momentum of 0.9 and weight decay of 5×10^{-4} , the SGD Nesterov optimizer and mixed precision. In our studies, we employ either a step scheduler (dividing the learning rate by 10 at the 30th, 60th, and 90th epochs) or a cosine decay scheduler [Loshchilov & Hutter (2016)]. We find $\alpha = 0.25$ and $\alpha = 0.05$ to be good choices for ResNet-18 and ResNet-50, though larger values such as $\alpha = 0.3$ also have good performance as well. The experimental results are shown in Table 9, and we mention the value of the PGT hyperparameter (α) in each experiment. We explore different values of α in section 5.6 and provide detailed grid plots in the supplementary section.

In our experiments with Squeeze-and-Excitation variant of ResNet-50 i.e. SE-ResNet-50 [Hu et al. (2018)], we observe significant improvements; a 0.97% boost in training accuracy and a 0.734% increase in test accuracy. In our experiments with ResNet-50, we find an increase of 0.5% and 0.656% performance enhancement over the cosine scheduler baseline for training and testing respectively. The corresponding improvement over the step scheduler baseline for ResNet-50 is 0.57% (training) and 0.524% (testing). ResNet-101 sees a higher improvement in training fit, 0.81% to be exact, while the improvement over the test set is 0.362%. Smaller networks such as SE-ResNet-18 and ResNet-18 sees accuracy boosts which are smaller but nevertheless positive. Therefore we conclude that with PGT, the consistent improvements in training accuracies across all cases is because the networks train better and arrive at better optimas during convergence. Per epoch training and test accuracy plots of ResNet-18 and ResNet-50 (both with and without PGT) are shown in Fig. 14(a, b). For better visualization of the top end of the plots, we scale both axes in log scale. Because both training and test accuracies improve, it indicates that PowerGrad Transform allows the network to learn better representations.

These improvements are obtained with the number of epochs being kept the same as that of the baseline model. Alternatively, practitioners can apply PGT and have the networks converge to accuracy levels comparable to the baseline with a significantly reduced epoch budget. In Fig. 14(c), we illustrate one such instance in which we train a ResNet-50 to its cosine scheduler baseline in just 60 epochs with PGT, and match its baseline performance that is acquired in 100 epochs. This hints to the observation that PGT not only makes the network perform better but also speeds up training. Because of the acceleration in training that is made possible by PGT, the total amount of training time can be decreased by 40%.

Table 9: Results and comparison table for CNN networks trained on ImageNet-1K (sorted in descending order w.r.t. the last column denoting test accuracy improvements over baseline). Best training and test accuracies are highlighted in red and blue respectively. Accuracy differences are highlighted in yellow. In all experiments, we observe that PGT positively improves both training and test accuracies. We also observe that deeper networks such as SE-ResNet-50 and ResNet-50 accrue higher gains over the baselines as compared to shallow networks like ResNet-18.

Model	Scheduler	Method	PowerGrad Transform (α)	Train Accuracy (%)	Train Diff (%)	Test Accuracy (%)	Test Diff (%)
SE-ResNet-50	Cosine	Baseline	-	81.5		77.218	
		PGT	0.3	82.47	+0.97	77.952	+0.734
ResNet-50	Cosine	Baseline	-	79.18		76.56	
		PGT	0.05	79.68	+0.5	77.216	+0.656
ResNet-50	Step	Baseline	-	78.99		75.97	
		PGT	0.05	79.56	+0.57	76.494	+0.524
ResNet-101	Cosine	Baseline	-	82.29		77.896	
		PGT	0.3	83.1	+0.81	78.258	+0.362
SE-ResNet-18	Cosine	Baseline	-	71.42		71.09	
		PGT	0.25	71.6	+0.18	71.436	+0.346
ResNet-18	Step	Baseline	-	69.95		69.704	
		PGT	0.25	70.3	+0.35	69.844	+0.14
ResNet-18	Cosine	Baseline	-	70.38		70.208	
		PGT	0.25	70.53	+0.15	70.298	+0.09

5.5.2 NON-BATCHNORMALIZED NETWORKS

This section empirically examines some of the issues that occur when networks are trained without normalization layers and we provide insight into the effects of PowerGrad transform on such networks. We use ResNet-18 [He et al. (2016)] as the foundation model for all empirical trials owing to its popularity among deep learning practitioners and the relative simplicity with which it can be trained on a big dataset such as ImageNet-1K [Russakovsky et al. (2015)]. We found deeper networks (such as ResNet-34 and ResNet-50 models) are impossible to train without BatchNormalization because of increased depth. So in order to conduct our empirical experiments, we solely focus on the ResNet-18 architecture. For our experiments, we designate different layers with their corresponding layer indices. We provide a detailed layer-wise index list of ResNet-18 below.

For the layer indices used in figures Fig. 16, 17, 18, we provide a block diagram depicting the different convolutional layers of ResNet-18 with corresponding layer

indices in Fig. 15. Each layer represents a convolutional layer, with the layer index and the number of filters denoted alongside it. Downsampling convolutional layers in skip connections are drawn and their layer indices are shown on the skip connection paths.

ResNet-18 is composed of 20 convolutional layers and one fully connected (FC) layer. At the input, it has a single convolutional layer with filters of size 7×7 . Three convolutional layers are placed in the downsampling skip connections. The remaining convolutional layers have several 3×3 filters that are grouped together to create a block. Each basic block is connected by a skip connection. At the end of all convolutional layers, a global average pooling layer downsamples the features to a fixed-size feature vector and transfers it to the FC layer, which fits the features using regression and outputs logits. The logits z_i produced by the FC layer are fed into a softmax layer, which transforms them to predicted probabilities p_i using the equation $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

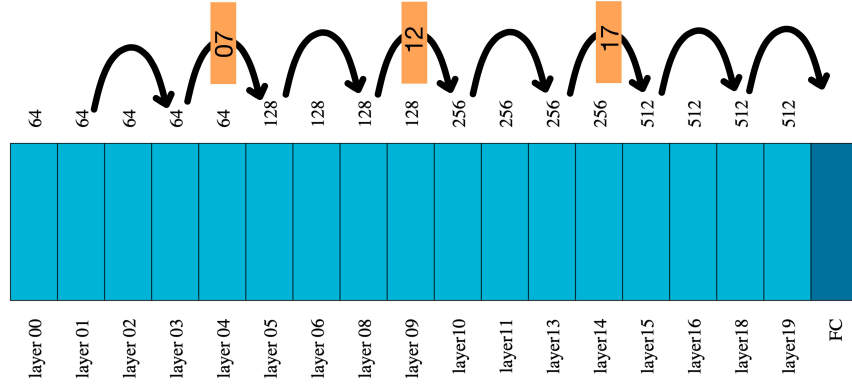


Figure 15: ResNet-18 architecture with layer indices of convolutional layers and the number of filters in each layer.

We concentrate on the following metric: the per-filter L2-norm of each layer’s weight tensor. Throughout the training process, we monitor variations in the per-filter weight norm. Because a full plot of each layer’s data would use an inordinate amount of space, we focus on a few critical layers’ plots. The supplementary section contains full plots for all layer for various settings. The

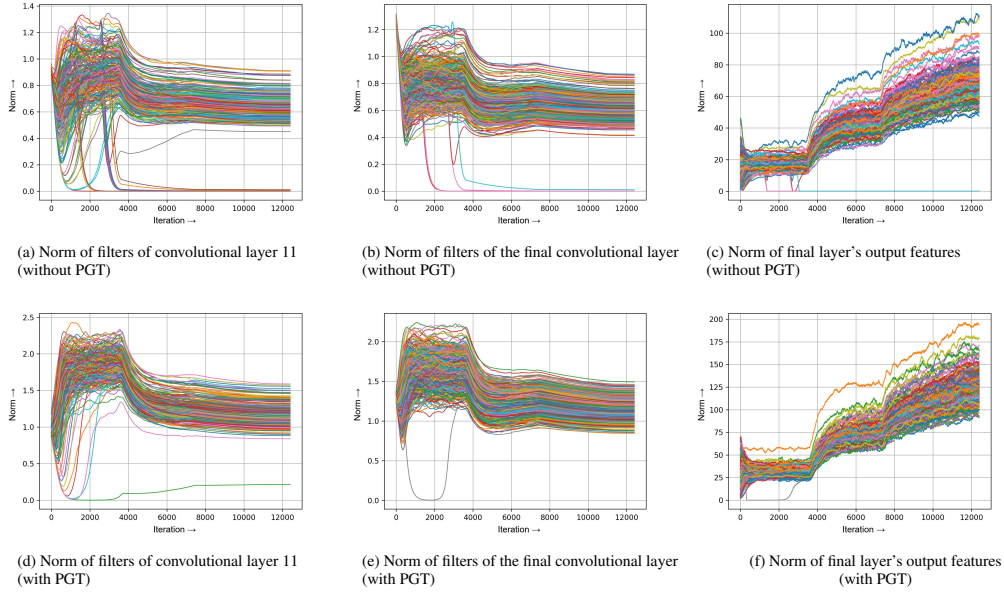


Figure 16: Experiment demonstrating the Zeroing Out phenomena, as observed in different filters and feature vectors. PGT greatly reduces the zeroing out of filters and promote better learning.

norm of the weights of the 11th convolutional layer is shown in Fig. 16(a). Layer 11 includes 256 filters. Each colour on the figure represents the iteration-wise evolution of a particular filter throughout the course of training. The critical point to note is that some filters achieve a norm of zero during training. We refer to this event as ‘Zeroing Out’, and occurs when a channel of a weight tensor gets fully filled with zeros. If all the coefficients of the filter becomes zero, only then the L2 norm of that filter becomes zero. If we observe zero filter norm for a filter, then that filter does not contribute at all to determine the input-output relationship of a dataset, as the feature tensor it produces is also an all-zero tensor. According to our findings, when a filter tensor becomes zeroed out, it does not recover with further training. We observe both effects **1) zeroed out filter** **2) zeroed out feature**.

Similar to the plot 16(a), we see the plot of the final convolutional layer in Fig. 16(b), where we observe a number of filters / units completely dropping out towards the end of training leading to underutilization of the network’s parameters. In Fig. 16(c), we see the norm of the feature vector at the output side of the global average pooling layer of ResNet-18. The feature vector here directly interfaces

with the fully connected layer immediately after. Any zeroed out regions in this tensor directly leads to permanent information loss, as it does not contribute to the learning of decision boundaries in the upcoming fully connected layer, and indeed there exists a few zeroed out features in the baseline model, which we discuss next. The impact of having a zeroed out filter or a feature might have a variety of consequences for training and optimization. If a zeroed out filter is present in a convolutional layer, there are two possible outcomes. If the layer is immediately followed by a fully connected regression layer, there is an abrupt loss of information. For the reason that a zeroed out filter always generates zeroed out features regardless of the input image, when this tensor is fed to a fully connected layer, it does not communicate any information that is useful for classification. In the event that the zeroed out filter has a skip connection linked across it, this may be circumvented as long as there are no overlapping zeroed out channels in the two feature maps. On the other hand, if a feature tensor has a zeroed out channel at the input of a layer, the layer creates a zeroed out feature channel at the output, regardless of the weights. However, if there is a zeroed out channel in the feature map just before the fully connected layer, then this results in a permanent loss of information as it does not contribute to the learning of decision boundaries. ResNet-18 suffers from the same kind of information loss, as the final feature tensor (which is generated after the global average pooling layer) has multiple channels that are zeroed out [Fig. 16(c)]. This information collapse phenomenon occurs when a channel that has been zeroed out stays zeroed out for all images in the dataset.

The impact of a feature tensor or weight tensor being zeroed out is seen to be particularly common in training procedures involving larger batch sizes in networks without normalization layers. As a result, the network's parameters are underutilized since some of the filters are not involved in extracting any useful information that is important for performing classification. When working with

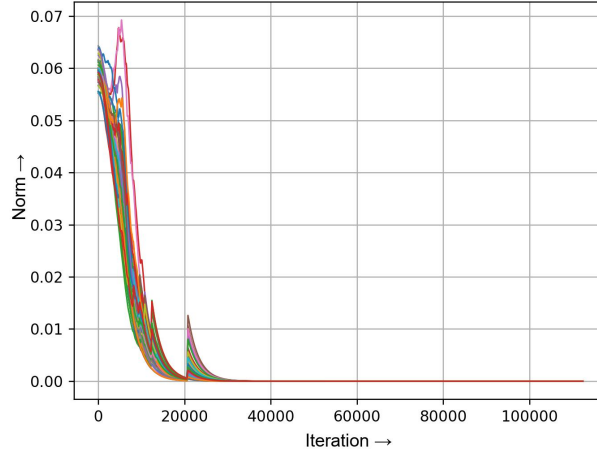


Figure 17: Per filter norm vs. iteration plot of layer 5 of an unnormalized ResNet-18, trained with a very high batch size of 1024. The weight matrix of the entire layer zeroes out during training.

large batch sizes, it is possible that an entire layer zeroes out [Fig. 17]. In a network without skip connections, this can cause a full collapse in training. For example, if a weight tensor completely zeroes out, and it always produces a zero feature tensor, then the input feature tensor to the next convolutional layer is a zero tensor and thus the output of the next layer is always a zero tensor irrespective of its weights, and this continues until the end of the neural network pipeline leading to zero logits. Also, zeroed out weights tensors lead to zeroed out gradients hence stopping training for all subsequent iterations leading to a collapse in training. Although residual networks bypass this difficulty via the use of the skip connection, it does not solve the layer’s non-participation in the overall objective of classification. It is possible to solve this issue by using methods such as gradient clipping (GC), adaptive gradient clipping (AGC), and PowerGrad Transform (PGT), all of which regulate the amount of gradient that flows at various junctions of the network. GC and AGC modify the gradients at each and every filter of every layer. PGT performs this operation at only one junction, which is at the softmax layer.

In a similar fashion to the study in section 5.5.2, we examine the norm profiles of features and weights in the PGT and AGC. The supplementary section includes

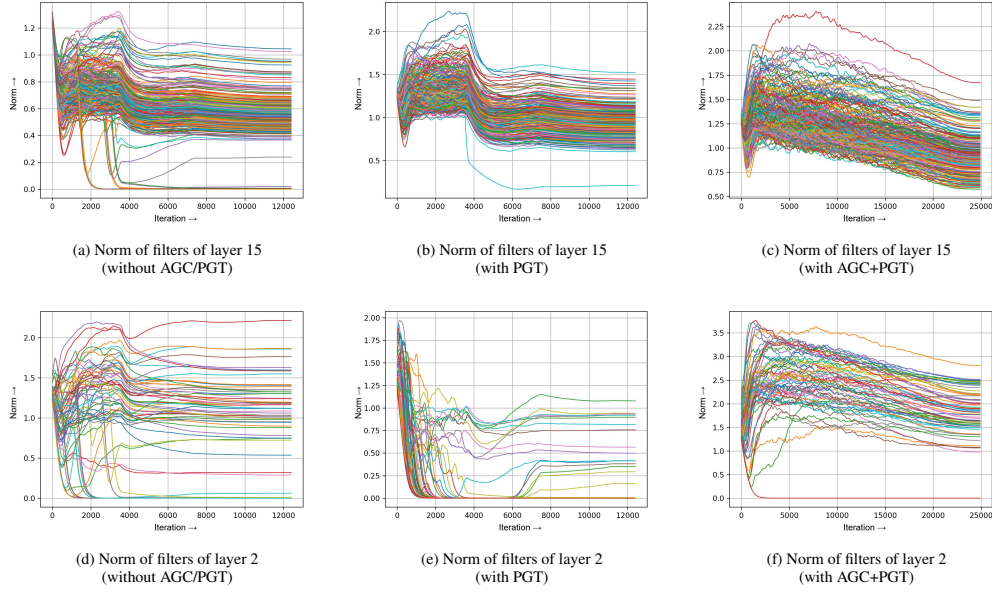


Figure 18: Experiment demonstrating the efficacy of PGT and AGC+PGT, as observed in different layers.

all the detailed graphs of all layer. In Fig. 16(d), we show the per-filter norm plot of the 11th convolutional layer as it progresses through a PGT enabled training session. We observe that the number of zeroed out filters has considerably reduced. We observe similar benefits in plot of Fig. 16(e), which is the plot of the per-filter norm of the final convolutional layer. Coming to benefits in feature norms, we observe in plot of Fig. 16(f), the final feature tensor obtained with PGT enabled training do not contain any zeroed out regions, leading us to assert that information loss is heavily mitigated as the features pass on from the feature extracting convolutional layers to the fully connected stage for regression.

AGC is excellent at eliminating the issue of filter and feature zeroing out. However, AGC recovers a lower performance benefit than PGT. In the instance of PGT, we see that it also mitigates the zeroing out phenomenon in most layers. In Fig. 18(a), we see a plot of the 15th layer of the non-BN ResNet-18 trained without PGT. As we can see that there has been quite a few filter dropouts in the baseline. PGT recovers most of the filters [Fig. 18(b)]. In some layers however, the performance of PGT is slightly weaker than the baseline. The 2nd convolutional

Table 10: Results for non-normalized ResNets on ImageNet-1K. Each experiment is conducted with a 100 epoch budget using the cosine decay scheduler. Best training and test accuracies are highlighted in red and blue respectively. Top differences in training and test accuracies are marked in yellow. The baseline in each segment is taken as the reference point for all ‘Diff’ accuracies in that segment. The first three segments highlight comparison for experiments where the methods are applied individually, while the last segment denotes performance metrics for combinations of different methods (GC + PGT, AGC + PGT). We see that at any batch size, with PGT, the performance of non-normalized version of ResNet-18 improves significantly over the baseline owing to the improvements in training as observed in Fig. 18.

Model	Batch Size	Method	PowerGrad Transform (α)	Train Accuracy (%)	Train Diff (%)	Test Accuracy (%)	Test Diff (%)
ResNet-18 (non-BN)	1024	Baseline	-	66.27	-	64.816	-
	1024	PGT	0.92	66.62	+0.35	65.498	+0.682
	512	Baseline	-	68.02	-	66.552	-
	512	PGT	0.25	69.5	+1.48	67.236	+0.684
	256	Baseline	-	68.86	-	66.796	-
	256	GC	-	69.04	+0.18	67.064	+0.268
	256	AGC	-	69.06	+0.2	67.298	+0.502
	256	PGT	0.25	69.97	+1.11	67.814	+1.018
	256	Baseline	-	68.86	-	66.796	-
	256	GC+PGT	0.25	68.67	-0.19	67.088	+0.292
	256	AGC+PGT	0.25	70.92	+2.06	68.856	+2.06

layer has more zeroed out filters in PGT [Fig. 18(e)] than in the baseline [Fig. 18(d)]. Despite that, PGT achieves a higher overall accuracy (Table 10). This necessitates the simultaneous use of AGC and PGT, for the non-BN ResNet-18.

We provide performance metrics on the test set of ImageNet-1K for different batch sizes for the non-BN ResNet-18 and also experiment with different methods (GC, AGC, PGT along with their combinations). Table 10 shows the findings. To begin, we use a batch size of 1024. The baseline performance for this batch size is drastically inferior to baselines for other batch sizes. PGT helps regain some of the lost performance by 0.682% (65.498% vs. 64.816%). This illustrates that in unnormalized networks, accuracy suffers significantly when batch sizes are large. Additionally, the training process is often unstable at this batch size, resulting in filters and sometimes entire layers frequently zeroing out (as seen in section 5.5.2). At a batch size of 512, invoking PGT improves the training accuracy baseline by 1.48% and the test accuracy baseline by 0.684%, while at a batch size of 256, the

improvement in training and test accuracies are 1.11% and 1.018% respectively. In comparison, the test accuracy improvements obtained by GC and AGC at batch size of 256 is much less, at 0.27% and 0.5%, respectively.

On the training accuracy front, since we get a significant boost (1.48% at batch size of 512 and 1.11% at batch size of 256), it leads us to infer that when PowerGrad Transform is used, the network fits the training dataset more tightly and the convergence optima is significantly superior. We also notice that the training versus testing gap increases with PGT as compared to the gap at baseline even though both training and test accuracies improve. Therefore it allows us to conclude that the improvements in training and test accuracies are obtained not through regularization like it does in methods such as label smoothing; rather the improvements are obtained as PGT enables the network to utilize its learning capacity better and arrive at a higher performing optima at convergence, something which is inaccessible through traditional gradient based training.

We also experiment with combinations of GC/AGC and PGT. The advantage of combining GC with PGT is not very significant. When AGC and PGT are combined, we see a tremendous increase in test accuracy of over 2.06% over the baseline. As seen in Fig. 18(c) and 18(f), activating AGC and PGT concurrently results in no filter dropouts in all layers while also improving the accuracy by a significant margin. Additionally, we provide comprehensive graphs of the norm profiles of the AGC + PGT method for all layers in the supplementary section.

5.5.3 GRID SEARCH AND EFFECT ON LOSS

As seen in Fig. 19(a), the logit norm increases as α falls from 1 to 0. In the initial training phase, PGT causes the gradient in both the valid and wrong classes to rise as the network misclassifies majority of the examples. Due to the exponential nature of the softmax function, the predicted probability distribution may frequently

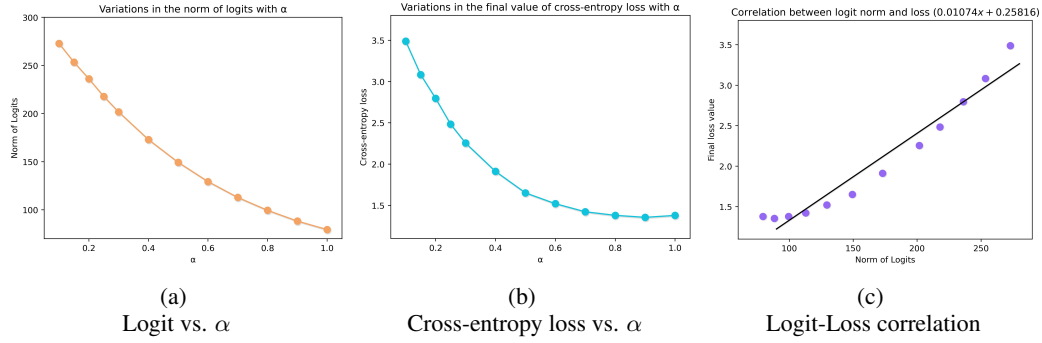


Figure 19: Plots of various statistical measures: **(a)** Variations in the logit norm vs. α . The logit norm is calculated per image over the test set of ImageNet-1K (at the linear layer of the ResNet-18 architecture) and then averaged. **(b)** Variations in the final loss values obtained for different α settings. **(c)** Correlation between loss and logits. Regression line equation : $(0.01074x + 0.25816)$.

be close to a delta distribution, even in instances of misclassifications. PGT reshapes the predicted probability distribution. It redistributes probability from the confident classes to the less confident ones. Gradient rises in all classes. We also see that the final value of the loss rises as α drops, and that the logit norm and the final value of the loss are linearly correlated Fig. 19(c). What is surprising is that: *it is possible to achieve higher accuracies even though the loss values are larger during convergence*. In our studies with ResNet-50, for instance, if PGT is not invoked, the corresponding test accuracy is 76.56% (Table 9) with a cross-entropy loss value of 0.852. However, if PGT is activated with $\alpha = 0.05$, the corresponding test accuracy is 77.216% (Table 9) with a loss value of 2.360. This is markedly different from the coupled gradient descent based training procedures. PGT enables the network to arrive at such optima where the loss values are high, but both training and test performance is better. From these plots of logits and losses we find that with the inclusion of PGT in the training process, the model can have access to such regions of the loss landscape that is otherwise inaccessible to traditional gradient based training procedures. There is no effect of PGT on small datasets such as CIFAR-10. In these scenarios, networks almost always achieve 100% training accuracy, leaving no room for PGT to assist.

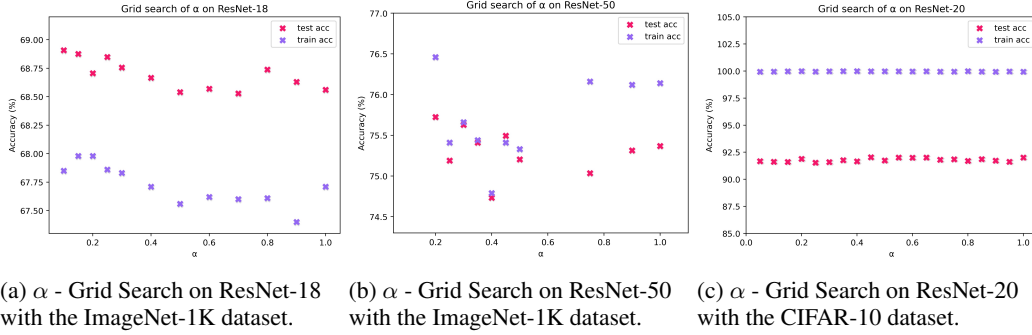


Figure 20: Grid search for various values of α for different architectures with the ImageNet-1K/CIFAR-10 dataset. Each datapoint is collected over a 50 epoch budget for ImageNet experiments and 360 epochs for CIFAR-10 experiment, with other hyperparameters kept the same as mentioned in section 5.5.1.

5.6 ABLATION STUDIES

We conduct an ablation study to investigate the effects of PowerGrad Transform for different values of the hyperparameter (α). Grid plots are made available in the supplementary section. We have previously demonstrated from the experiments depicted in section 5.5, that PowerGrad Transform causes the network to arrive at a better optima and fit the training dataset better than the baseline. In this ablation study, we use the ResNet-50 architecture and we combine our proposed method with other regularization techniques such as label smoothing and report our findings in Table 11. We also investigate the two optimal peaks of the hyperparameter (α) obtained from ResNet-50’s grid search ($\alpha = 0.05$ and $\alpha = 0.3$). First, we examine the effect of PGT on the step scheduler baseline in order to later compare it to the cosine scheduler baseline. **Row-1)** We begin with the step scheduler baseline (75.97%). **Row-2)** PGT improves upon the step scheduler baseline (test set) by a substantial margin with 0.524% (76.494% as opposed to 75.97%). **Row-3)** Introducing the cosine scheduler yields a 0.59% improvement (76.56% vs. 75.97%) over the step scheduler. **Row-4)** After introducing label smoothing, the test accuracy relative to the cosine scheduler baseline increases by only 0.138% (from 76.56% to 76.698%). **Row-5)** However, introducing PGT with $\alpha = 0.3$ alone (without label smoothing) improves the cosine scheduler baseline

Table 11: Ablation study for ResNet-50 on ImageNet-1K. LS column denotes whether label smoothing has been applied. A smoothing hyperparameter value of 0.1 has been used. The hyperparameter column that controls the amount of PGT gradient modification injected at softmax, Eq. 52.

#Row	Model	Scheduler	Label Smoothing	PowerGrad Transform (α)	Train Accuracy (%)	Test Accuracy (%)	Gap (%)
1.	ResNet-50	Step	$\times$	$\times$	78.99	75.97	3.02
2.	ResNet-50	Step	$\times$	0.3	79.56	76.494	3.066
3.	ResNet-50	Cosine	$\times$	$\times$	79.18	76.56	2.62
4.	ResNet-50	Cosine	0.1	$\times$	78.81	76.698	2.112
5.	ResNet-50	Cosine	$\times$	0.3	79.43	76.886	2.544
6.	ResNet-50	Cosine	0.1	0.3	78.47	76.968	1.502
7.	ResNet-50	Cosine	$\times$	0.05	79.68	77.216	2.464
8.	ResNet-50	Cosine	0.1	0.05	77.69	76.39	1.3

by 0.326% (76.886% vs. 76.56%). **Row-6)** Combining PGT ($\alpha = 0.3$) with label smoothing improves the performance on the test set further by 0.408% (from 76.56% to 76.968%) and reduces the generalization gap (from 2.54% to 1.5%). However, the impact of combining PGT with label smoothing can vary depending on the value of the hyperparameter (α). **Row-7)** With a PGT hyperparameter value of $\alpha = 0.05$, we notice the greatest performance improvement, 1.246% over the step scheduler test baseline and 0.656% over the cosine scheduler test baseline. **Row-8)** Adding label smoothing to PGT ($\alpha = 0.05$) hurts performance even though it reduces the generalization gap.

5.7 IMPLEMENTATION

A code snippet of the algorithm implemented using PyTorch [Paszke et al. (2019)] is shown below. In PyTorch, the available method to modify gradients in the backward pass is using the `register_hook` function which is called in the `forward` function. We explore the effect of this change from a theoretical standpoint in section 5.4. In section 5.5, we offer empirical data and experiments to support the proposed method. In PyTorch, the available method to modify gradients

in the backward pass is using the `register_hook` function which is called in the `forward` function.

```
# Modification of gradients take place
# inside the forward function
def forward(self, x, labels):
    # x: Input tensor with shape [N, C, H, W]
    # labels: One hot ground truth with shape [N, C]
    logits = self.layers(x)

    def powerGradTransform(grad):
        alpha = 0.3
        # Recover the predicted probabilities
        # generated in the forward pass
        grad_temp = grad / grad.shape[0]
        pred = grad_temp + labels
        # Transform the predicted probabilities
        pred = pred**alpha
        pred /= pred.sum(-1, keepdim=True)
        # Modify the gradient
        modified_grad = pred - labels
        modified_grad = modified_grad * grad.shape[0]
        return modified_grad

    if self.training:
        logits.register_hook(powerGradTransform)
    return logits
```

Figure 21: Python code of PowerGrad Transform based on PyTorch.

5.8 DISCUSSIONS

In the realm of standard neural network training, the gradients generated during the backward pass are inherently determined by the forward pass. Consequently, these two phases are intricately linked and operate in tandem. This is the foundational approach upon which the majority of existing neural networks are built and trained. However, a niche within this field that remains largely unexplored is the concept of modifying gradients during the backward pass. We present in this chapter, a novel concept of decoupled training. This technique deviates from the conventional approach by introducing modifications to the gradients during the backward pass, effectively creating a level of independence between the two phases of neural network training.

The primary method introduced in this research is "PowerGrad Transform." At its core, PowerGrad Transform is designed to modify the gradients right before the weight update occurs in the backward pass. This modification is not just a minor alteration, but a transformative one that substantially augments the predictive capabilities of the neural network. The overarching objective of this method is to guide the network towards a more optimal convergence point, ensuring that the final model is both precise and reliable. A significant advantage of the PowerGrad Transform is its computational efficiency. Unlike other methods that may introduce considerable overheads in terms of memory or computational power, the PowerGrad Transform seamlessly integrates into the neural network training process without incurring any additional costs. This efficiency is particularly notable, given the substantial improvements it brings to the model's performance.

Moreover, with PowerGrad Transform is versatile in its application. It has been designed to be easily integrated into existing neural network training pipelines. This ease of integration ensures that researchers and practitioners can effortlessly adopt this method without the need for substantial modifications to their current workflows. One of the salient outcomes of employing the PowerGrad Transform is the marked improvement in accuracies observed across datasets. Both training and test set performances exhibited significant enhancements, underscoring the efficacy of this method. This improvement in accuracy is not just a marginal increment but is substantial enough to be deemed noteworthy, especially when considering the minimal computational overheads associated with its implementation. We further delve into the technical intricacies and the underlying mechanisms of the PowerGrad Transform. We elucidate the mathematical foundations that govern this transformative method and provides insights into the rationale behind its design. Various experiments and evaluations are presented, showcasing the tangible benefits of this approach across different neural network architectures and datasets. These empirical evaluations underscore the universal applicability of the PowerGrad

Transform, highlighting its potential to redefine the way neural networks are trained in the future.

In conclusion, the PowerGrad Transform is a innovative approach in the domain of neural network training. By decoupling the forward and backward passes and introducing strategic gradient modifications, it paves the way for more efficient, accurate, and robust neural networks. Its simplicity, coupled with its undeniable efficacy, positions it as a significant shift in the landscape of deep learning research and applications.

6 CONCLUSION

In this thesis we attack several problems related to the learning capacity of a neural network from both an inter-network and an intra-network perspective. A convolutional neural network’s learning capacity may be intuitively defined as the complementary of its learning deficit at convergence. If a neural network converges to a suboptimal minimum, then the network’s learning ability is not exploited to its utmost potential. If all the networks in a pool have identical weights, then the pool’s total knowledge is equal to the knowledge obtained by any one network in the pool. Both of these scenarios indicate an underutilized learning ability. To combat these, we investigate a variety of strategies for encouraging networks, both within a single network and across networks, to make the most of their respective learning potential. To improve intra-network learning, we propose a backward pass gradient modification technique we call PowerGrad Transform, in which we tweak the gradients just before an optimization update step to make the network traverse parts of the loss landscape that would otherwise be avoided by conventional training techniques. We discover that networks trained with the PowerGrad Transform method provide considerably improved training and test accuracy metrics, resulting in an increased utilization of learning capacity. The capacity to learn is equally relevant when discussing a collection of neural networks. Feature Difference Loss functions are used in the inter-network scenario to convert the shared feature representations of all of the networks in the pool into a single real number. Afterwards, we use adversarial training to maximize the inverse

of the feature difference losses, which forces a cohort of networks to generate a wide variety of features.

6.1 KNOWLEDGE DIVERSIFICATION

In traditional neural network ensemble methodologies, multiple networks are trained individually, and the resulting outcomes are combined, typically through voting mechanisms or linear combinations. A significant limitation of this approach is the absence of information sharing among the ensemble’s foundational networks. We argue that for an enhanced performance of neural network ensembles, there is a pressing need to diversify their representations. With this in mind, we introduce a novel technique geared towards enhancing the learned representations within an ensemble. This is achieved by encouraging feature sharing during the training process. Our proposed method, termed “Feature Difference Loss” (FDL), is designed to stimulate the networks within the ensemble to capture distinct features. This direct promotion of model diversity stands in contrast to traditional methods. In our rigorous testing phase, we utilized ensembles comprising two to four networks. The results yielded notable improvements in comparison to other ensemble techniques. Intriguingly, these improvements were particularly evident for datasets characterized by fewer samples per category. This was observed in widely-used datasets such as MNIST, CIFAR-10, and CIFAR-100. Moreover, even when applied to extensive datasets like ImageNet, we observed positive outcomes, further underscoring the efficacy of our approach.

When diving deeper into the rationale for diversification, we found that ensembles combine the outcomes of multiple base networks to boost generalization. A critical factor that determines the efficacy of an ensemble is the diversity among its base networks. Essentially, for an ensemble to outperform its constituent networks, these networks must not only be accurate but also diverse in their errors.

When errors are uncorrelated, the ensemble’s average error decreases. However, a common pitfall in conventional ensemble methods is that they independently train each network, leading to a lack of incentive for the networks to learn diverse features. This is where our proposed FDL comes into play. By actively promoting diversity during training, FDL ensures that each network in the ensemble learns unique features, leading to more diverse and uncorrelated errors.

We introduced the Feature Difference Loss (FDL) as a loss function added to the conventional loss function used during training. FDL measures the difference in features learned by different networks in the ensemble. By penalizing similarities and rewarding differences, FDL encourages each network to focus on different aspects of the input data. Mathematically, FDL is computed as the negative pairwise mean squared similarity between the features extracted by the networks in the ensemble. This computation ensures that if two networks learn similar features, FDL will push the overall training towards discouraging the networks from learning similar features. For our experiments, we added the FDL term to the conventional loss functions such as cross-entropy. The combined loss function then guides the training of the networks in the ensemble. We also made the training regime free from hyperparameters, other than the ones that are typically found in traditional neural network training.

Our experiments involved training ensembles of varying sizes, from two to four networks, on several benchmark datasets. The results consistently demonstrated the superiority of our approach in comparison to conventional ensemble methods. On datasets like MNIST, CIFAR-10, and CIFAR-100, which have fewer samples per category, the benefits of our approach were even more pronounced. Notably, on the ImageNet dataset, despite its vast size and complexity, our method demonstrated a marked improvement in performance. When contrasted with other ensemble techniques, our approach consistently outperformed them across a range of metrics.

Furthermore, in terms of model diversity, our method ensured that the networks in the ensemble learned a broader set of features, leading to more diverse and uncorrelated errors. This, in turn, boosted the overall performance of the ensemble, making it more robust and reliable.

In conclusion, through our innovative approach, we have showcased the importance of actively promoting diversity during the training of neural network ensembles. Our Feature Difference Loss (FDL) technique encourages networks within an ensemble to capture distinct features, leading to improved performance, especially in datasets with fewer samples per category. This represents a significant advancement in the field of neural network ensembles and sets the stage for further exploration and refinement of these ideas.

6.2 POWERGRAD TRANSFORM

In conventional neural network training, the gradients produced during the backward pass are dictated by the forward pass. This invariably ties the two stages together, and this coupling forms the bedrock of how most neural networks are currently trained. Interestingly, while this approach has stood the test of time, the avenue of modifying gradients in the backward pass remains relatively uncharted in academic literature. Through our research, we have ventured into this less-explored domain, focusing on the concept of decoupled training. This approach hinges on the idea of altering gradients during the backward pass. Our contribution to this realm is the introduction of the PowerGrad Transform—a method that, despite its simplicity, holds immense potential. The essence of the PowerGrad Transform lies in adjusting the gradients prior to the weight update in the backward pass. Through our extensive experiments, we found that this seemingly minor tweak can substantially elevate the predictive prowess of a neural network. More than just

enhancing performance, the PowerGrad Transform steers the network towards a more optimal convergence point.

One of the standout attributes of the PowerGrad Transform is its computational efficiency. It seamlessly integrates into the training process without burdening the system with added computational or memory costs. Yet, the benefits it pays in terms of improved accuracies on both training and testing datasets are substantial. Its design ensures that it can be effortlessly incorporated into existing training frameworks, making it a versatile tool for practitioners and researchers alike. Delving deeper into the underpinnings of the PowerGrad Transform, it's crucial to understand the traditional gradient-based training process. In standard training, the forward pass computes activations, and the backward pass computes gradients based on these activations, which are then used to update the model's weights. This cyclic process continues until convergence. While effective, this method doesn't always guarantee that the network will converge to the best possible optima. Our PowerGrad Transform disrupts this cycle by introducing a decoupling mechanism. Instead of relying solely on the forward pass's outcomes to determine the backward pass gradients, we introduce a transform that modifies these gradients. This pushes the network to explore a broader solution space, eventually converging to a better optima, as depicted by the experimental results.

At its core, the PowerGrad Transform is a mathematical operation applied to the gradients during the backward pass. Without divulging the intricate mathematical details, we can state that this transform scales and shifts the gradients in a manner that enhances their influence on the weight update process. This controlled manipulation ensures that the weight updates are more informed, pushing the network towards regions in the solution space that offer better performance. An essential aspect of the PowerGrad Transform is its adaptability. The transform parameters can be adjusted based on the dataset, the architecture of the neural

network, and the specific training objectives. This flexibility ensures that the PowerGrad Transform remains relevant across a myriad of applications. Our claims surrounding the efficacy of the PowerGrad Transform aren't merely theoretical. We subjected it to a battery of experiments across diverse datasets and network architectures.

In our experiments, we observed substantial improvements in convergence rates. Networks trained with the PowerGrad Transform reached optimal performance levels faster than their traditionally trained counterparts. Furthermore, the final accuracies achieved on both training and testing datasets were invariably higher when the PowerGrad Transform was employed. The implications of our work extend beyond just enhancing the training of individual neural networks. The PowerGrad Transform, with its ability to improve convergence and overall performance, has the potential to improve transfer learning, and other advanced training techniques. Given its computational efficiency, it can be significant in scenarios where resources are constrained, such as edge computing and mobile devices. In summary, with the PowerGrad Transform, we have ventured into the relatively unexplored domain of decoupled training. By introducing a mechanism to modify gradients during the backward pass, we have showcased how neural networks can be trained to achieve superior performance without incurring additional computational costs. Our method, which is both simple and powerful, represents a significant stride forward in neural network training methodologies.

7 LIST OF PUBLICATIONS

1. Bishshoy Das and Sumantra Dutta Roy. Knowledge Diversification in Ensembles of Identical Neural Networks. In 33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022. BMVA Press, 2022. URL <https://bmvc2022.mpi-inf.mpg.de/0798.pdf>. [Das & Roy (2022)]
2. Bishshoy Das*, Milton Mondal*, Brejesh Lall, Shiv Dutt Joshi, and Sumantra Dutta Roy. Altering Backward Pass Gradients improves Convergence, 2023. In Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering. Larkspur Landing South San Francisco, USA and KSIR Virtual Conference Center, USA. URL <https://ksiresearch.org/seke/seke23paper/paper177.pdf>. [Das et al. (2023)]

Not Related to the Thesis :—

3. Bishshoy Das and Sumantra Dutta Roy. Edge-aware Image Super-resolution using a Generative Adversarial Network. SN Computer Science, 2022. [Das & Roy (2022)]
4. Bishshoy Das, H Dutta, and Manas Kamal Bhuyan. Graph-based Depth Estimation in a Monocular Image using Constrained 3D Wireframe Models. In International Conference on Computer Vision and Image Processing, pp. 419-433. Springer, 2020. [Das et al. (2020)]

* Equal Contribution

5. Milton Mondal*, Bishshoy Das*, Sumantra Dutta Roy, Pushpendra Singh, Brejesh Lall, and Shiv Dutt Joshi. Adaptive CNN Filter Pruning using Global Importance Metric. *Computer Vision and Image Understanding*, 222:103511, 2022. [Mondal et al. (2022)]
6. Mondal M*, Das B*, Lall B, Singh P, Roy SD, Joshi SD. Feature independent Filter Pruning by Successive Layers analysis. *Computer Vision and Image Understanding*. 2023 Sep 3:103828. [Mondal et al. (2023)]
7. Mondal M, Das B, Lall B, Singh P, Roy SD, Joshi SD. Multi-band Convolutional Neural Network for Efficient Utilization of Model Parameters. [Mondal et al. (2023)]

* Equal Contribution

REFERENCES

- [1] Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7350–7357, 2020.
- [2] Maxwell M Aladago and Lorenzo Torresani. Slot machines: Discovering winning combinations of random weights in neural networks. In *International Conference on Machine Learning*, pp. 163–174. PMLR, 2021.
- [3] Monther Alhamdoosh and Dianhui Wang. Fast decorrelated neural network ensembles with random weights. *Information Sciences*, 264:104–117, 2014.
- [4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2654–2662, 2014.
- [5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [6] Tanmay Batra and Devi Parikh. Cooperative learning with visual attributes. *arXiv preprint arXiv:1705.05512*, 2017.
- [7] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.
- [8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

- [9] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 92–100, 1998.
- [10] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [11] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [12] Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.
- [13] Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.
- [14] Hugh Chen, Scott Lundberg, and Su-In Lee. Checkpoint ensembles: Ensemble methods from a single training process. *arXiv preprint arXiv:1710.03282*, 2017.
- [15] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4794–4802, 2019.
- [16] Jan Chorowski and Navdeep Jaitly. Towards better decoding and language model integration in sequence to sequence models. *arXiv preprint arXiv:1612.02695*, 2016.
- [17] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. In *International conference on machine learning*, pp. 874–883. PMLR, 2017.
- [18] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In

- Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 702–703, 2020.
- [19] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [20] Bishshoy Das and Sumantra Dutta Roy. Knowledge diversification in ensembles of identical neural networks. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. URL <https://bmvc2022.mpi-inf.mpg.de/0798.pdf>.
- [21] Bishshoy Das and Sumantra Dutta Roy. Edge-aware image super-resolution using a generative adversarial network. *SN Computer Science*, 2022.
- [22] Bishshoy Das, H Dutta, and Manas Kamal Bhuyan. Graph-based depth estimation in a monocular image using constrained 3d wireframe models. In *International Conference on Computer Vision and Image Processing*, pp. 419–433. Springer, 2020.
- [23] Bishshoy Das, Milton Mondal, Brejesh Lall, Shiv Dutt Joshi, and Sumantra Dutta Roy. Altering backward pass gradients improves convergence. In *Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering*, 2023. URL <https://ksiresearch.org/seke/seke23paper/paper177.pdf>.
- [24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

- [25] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.
- [26] Abhimanyu Dubey, Otkrist Gupta, Pei Guo, Ramesh Raskar, Ryan Farrell, and Nikhil Naik. Pairwise confusion for fine-grained visual classification. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 70–86, 2018.
- [27] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.
- [28] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1): 1997–2017, 2019.
- [29] Jiemin Fang, Yukang Chen, Xinbang Zhang, Qian Zhang, Chang Huang, Gaofeng Meng, Wenyu Liu, and Xinggang Wang. Eat-nas: elastic architecture transfer for accelerating large-scale neural architecture search. *arXiv preprint arXiv:1901.05884*, 2019.
- [30] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pp. 1607–1616. PMLR, 2018.
- [31] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8803–8812, 2018.

- [32] Shiming Ge, Shengwei Zhao, Chenyu Li, and Jia Li. Low-resolution face recognition in the wild via selective knowledge distillation. *IEEE Transactions on Image Processing*, 28(4):2051–2062, 2018.
- [33] Shiming Ge, Zhao Luo, Chunhui Zhang, Yingying Hua, and Dacheng Tao. Distilling channels for efficient deep tracking. *IEEE Transactions on Image Processing*, 29:2610–2621, 2019.
- [34] Yixiao Ge, Dapeng Chen, and Hongsheng Li. Mutual mean-teaching: Pseudo label refinery for unsupervised domain adaptation on person re-identification. *ICLR*, 2020.
- [35] Giorgio Giacinto and Fabio Roli. Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9-10):699–707, 2001.
- [36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144, 2020.
- [38] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.
- [39] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

- [40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [41] Yu-Lin He, Xiao-Liang Zhang, Wei Ao, and Joshua Zhexue Huang. Determining the optimal temperature parameter for softmax function in reinforcement learning. *Applied Soft Computing*, 70:80–85, 2018.
- [42] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.
- [43] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [44] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pp. 278–282. IEEE, 1995.
- [45] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.
- [46] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.
- [47] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.
- [48] Hiroshi Inoue. Adaptive ensemble prediction for deep neural networks based on confidence level. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1284–1293. PMLR, 2019.

- [49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.
- [50] Siddhartha Jain, Ge Liu, Jonas Mueller, and David Gifford. Maximizing overall diversity for improved uncertainty estimates in deep ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4264–4271, 2020.
- [51] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, 2019.
- [52] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pp. 462–466, 1952.
- [53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [54] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [56] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7, 1994.

- [57] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.
- [58] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.
- [59] Duong H Le, Trung-Nhan Vo, and Nam Thoai. Paying more attention to snapshots of iterative pruning: Improving model compression via ensemble distillation. *arXiv preprint arXiv:2006.11487*, 2020.
- [60] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pp. 21–28, 1988.
- [61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [62] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.
- [63] Hakmin Lee, Hong Joo Lee, Seong Tae Kim, and Yong Man Ro. Robust ensemble model training via random layer sampling against adversarial attack. *arXiv preprint arXiv:2005.10757*, 2020.
- [64] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

- [65] Zhengqi Li, Qianqian Wang, Noah Snavely, and Angjoo Kanazawa. Infinitenature-zero: Learning perpetual view generation of natural scenes from single images. In *ECCV*, 2022.
- [66] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.
- [67] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.
- [68] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.
- [69] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 212–220, 2017.
- [70] Xuan Liu, Xiaoguang Wang, and Stan Matwin. Improving the interpretability of deep neural networks with knowledge distillation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 905–912. IEEE, 2018.
- [71] Yong Liu and Xin Yao. Ensemble learning via negative correlation. *Neural networks*, 12(10):1399–1404, 1999.
- [72] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

- [73] Shuchang Lyu, Qi Zhao, Yujing Ma, and Lijiang Chen. Make baseline model stronger: Embedded knowledge distillation in weight-sharing based ensemble network. 2021.
- [74] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Atlanta, Georgia, USA, 2013.
- [75] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.
- [76] Milton Mondal, Bishshoy Das, Sumantra Dutta Roy, Pushpendra Singh, Brejesh Lall, and Shiv Dutt Joshi. Adaptive cnn filter pruning using global importance metric. *Computer Vision and Image Understanding*, 222:103511, 2022.
- [77] Milton Mondal, Bishshoy Das, Brejesh Lall, Pushpendra Singh, Sumantra Dutta Roy, and Shiv Dutt Joshi. Feature independent filter pruning by successive layers analysis. *Computer Vision and Image Understanding*, pp. 103828, 2023.
- [78] Milton Mondal, Bishshoy Das, Brejesh Lall, Pushpendra Singh, Sumantra Dutta Roy, and Shiv Dutt Joshi. Multi-band convolutional neural network for efficient utilization of model parameters. 2023.
- [79] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 4696–4705, 2019.

- [80] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.
- [81] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
- [82] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. In *Doklady an ussr*, volume 269, pp. 543–547, 1983.
- [83] Derek Partridge and Wojtek Krzanowski. Software diversity: practical statistics for its measurement and exploitation. *Information and software technology*, 39(10):707–717, 1997.
- [84] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2(417):1, 2012.
- [85] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318. PMLR, 2013.
- [86] Nikolaos Passalis and Anastasios Tefas. Unsupervised knowledge transfer using similarity embeddings. *IEEE transactions on neural networks and learning systems*, 30(3):946–950, 2018.
- [87] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett

- (eds.), *Advances in Neural Information Processing Systems* 32, pp. 8024–8035. Curran Associates, Inc., 2019. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-...-deep-learning-library.pdf>.
- [88] Tim Pearce, Felix Leibfried, and Alexandra Brintrup. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, pp. 234–244. PMLR, 2020.
- [89] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural networks*, 11(4):761–767, 1998.
- [90] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. Deep co-training for semi-supervised image recognition. In *Proceedings of the european conference on computer vision (eccv)*, pp. 135–152, 2018.
- [91] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.
- [92] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.
- [93] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [94] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.
- [95] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

- [96] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [97] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.
- [98] Robert E Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pp. 1401–1406. Citeseer, 1999.
- [99] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pp. 37–52. Springer, 2013.
- [100] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [101] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472. IEEE, 2017.
- [102] Samuel Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, pp. 9058–9067. PMLR, 2020.
- [103] Guocong Song and Wei Chai. Collaborative learning for deep neural networks. *ICLR*, 2018.
- [104] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

- [105] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.
- [106] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.
- [107] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5265–5274, 2018.
- [108] Jiyue Wang, Pei Zhang, and Yanxiong Li. Memory-replay knowledge distillation. *Sensors*, 21(8):2792, 2021.
- [109] Keze Wang, Liang Lin, Chenhan Jiang, Chen Qian, and Pengxu Wei. 3d human pose machines with self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2019.
- [110] Ning Wang, Wengang Zhou, Yibing Song, Chao Ma, and Houqiang Li. Real-time correlation tracking via joint model compression and transfer. *IEEE Transactions on Image Processing*, 29:6123–6135, 2020.
- [111] Andrew M Webb, Charles Reynolds, Dan-Andrei Iliescu, Henry Reeve, Mikel Luján, and Gavin Brown. Joint training of neural network ensembles. *stat*, 1050:12, 2019.
- [112] Charles Weill, Javier Gonzalvo, Vitaly Kuznetsov, Scott Yang, Scott Yak, Hanna Mazzawi, Eugen Hotaj, Ghassen Jerfel, Vladimir Macko, Ben Adlam, et al. Adanet: A scalable and flexible framework for automatically learning ensembles. *arXiv preprint arXiv:1905.00080*, 2019.

- [113] Long Wen, Liang Gao, and Xinyu Li. A new snapshot ensemble convolutional neural network for fault diagnosis. *Ieee Access*, 7: 32037–32047, 2019.
- [114] Yanzhao Wu, Ling Liu, Zhongwei Xie, Juhyun Bae, Ka-Ho Chow, and Wenqi Wei. Promoting high diversity ensemble learning with ensemblebench. *arXiv preprint arXiv:2010.10623*, 2020.
- [115] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.
- [116] Zhirong Wu, Alexei A Efros, and Stella X Yu. Improving generalization via scalable neighborhood component analysis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 685–701, 2018.
- [117] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4753–4762, 2016.
- [118] Ting-Bing Xu and Cheng-Lin Liu. Data-distortion guided self-distillation for deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5565–5572, 2019.
- [119] Jun Yang and Fei Wang. Auto-ensemble: An adaptive learning rate scheduling based deep learning model ensembling. *IEEE Access*, 8: 217499–217509, 2020.
- [120] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. Mutualnet: Adaptive convnet via mutual learning from network width and resolution. In *European Conference on Computer Vision*, pp. 299–315. Springer, 2020.

- [121] Jiangchao Yao, Jiajie Wang, Ivor W Tsang, Ya Zhang, Jun Sun, Chengqi Zhang, and Rui Zhang. Deep learning from noisy image labels with quality embedding. *IEEE Transactions on Image Processing*, 28(4):1909–1922, 2018.
- [122] Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3903–3911, 2020.
- [123] Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13876–13885, 2020.
- [124] Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris C Holmes, Frank Hutter, and Yee Teh. Neural ensemble search for uncertainty estimation and dataset shift. *Advances in Neural Information Processing Systems*, 34:7898–7911, 2021.
- [125] Andrew Zhai and Hao-Yu Wu. Classification is a strong baseline for deep metric learning. *arXiv preprint arXiv:1811.12649*, 2018.
- [126] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.
- [127] Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713–3722, 2019.

- [128] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4320–4328, 2018.
- [129] Tianyi Zhou, Shengjie Wang, and Jeff A Bilmes. Diverse ensemble evolution: Curriculum data-model marriage. *Advances in Neural Information Processing Systems*, 31, 2018.
- [130] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.
- [131] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.