Scaling Local Learning for Supervised and Self-supervised Learning

Adeetya Patel

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

February 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Adeetya Patel**

Entitled: **Scaling Local Learning for Supervised and Self-supervised Learning**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Essam Mansour

_____ Examiner
Dr. Essam Mansour

_____ Examiner
Dr. Ching Suen

_____ Supervisor
Dr. Eugene Belilovsky

Approved by _____
Dr. Lata Narayanan, Chair
Department of Computer Science and Software Engineering

February 07, 2023

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Scaling Local Learning for Supervised and Self-supervised Learning

Adeetya Patel

Traditional neural network training methods optimize a monolithic objective function jointly for all the components. This can lead to various inefficiencies in terms of potential parallelization. Local learning is an approach to model-parallelism that removes the standard end-to-end learning setup and utilizes local objective functions to permit parallel learning amongst model components in a deep network. Recent works have demonstrated that variants of local learning can lead to efficient training of modern deep networks. However, in terms of how much computation can be distributed, these approaches are typically limited by the number of layers in a network. Hence, the first study explores how local-learning can be applied at the level of splitting layers or modules into sub-components, adding a notion of width-wise modularity to the existing depth-wise modularity associated with local learning. We investigate local-learning penalties that permit such models to be trained efficiently. Our experiments on various datasets demonstrate that introducing width-level modularity can lead to computational advantages over existing methods and opens new opportunities for improved model-parallel distributed training. The second study focuses on adapting existing local-learning frameworks to self-supervised learning tasks, specifically using the SimCLR method. However, existing local-learning frameworks lack in performance due to task-relevant information collapse in early layers. To address the issue, we propose modifying the local objective functions layerwise to gradually increase the problem difficulty with depth. We found that our method was able to maintain the similar performance as the end-to-end trained model while also increasing parallelization.

Acknowledgments

I would like to express my deepest gratitude to my research supervisor, Prof. Eugene Belilovsky, for his unwavering support, guidance, and mentorship throughout my graduate studies. His expertise, invaluable feedback, and constant encouragement have been instrumental in the completion of this thesis. His kind-heartedness and genuine interest in my success as a researcher have left a lasting impact on me. I cannot thank him enough for his patience and understanding for the entire process. I greatly value that he was consistently accessible to talk about the challenges I encountered over the past two years. I am truly grateful to have had the opportunity to work with such an esteemed, dedicated, and compassionate mentor.

I would also like to express my appreciation to Dr. Michael Eickenberg for his contributions to my research. His ideas and suggestions have been very helpful and valuable throughout the process. I am grateful for his support and collaboration.

I am also incredibly grateful to my family and friends for their unwavering support, encouragement, and understanding throughout this journey. Their love and support have been a source of strength for me, and I could not have done this without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In traditional neural network training methods, such as backpropagation, the parameters of each layer of the network must wait for the signal to propagate to the top and back down before updating, which can be time-consuming and may not fully utilize available resources (Jaderberg et al., 2017a). The local learning framework addresses this issue by proposing the simultaneous optimization of the parameters at each layer. Instead of training each layer of the network sequentially, this framework suggests that the parameters of each layer should be learned and updated in parallel (Belilovsky et al., 2020; Laskin et al., 2021; Nøkland & Eidnes, 2019; Wang et al., 2021). This is achieved by applying a separate loss to each layer. It allows for more efficient optimization of the parameters in terms of training time. In this thesis, we present two closely related studies that examine the potential benefits and limitations of local learning frameworks. Additionally, we propose new methods as extensions to existing approaches in order to address any identified limitations, which are described briefly in the subsequent sections.

1.1 Neuron-Group Level Parallelization

Existing frameworks for local learning Belilovsky et al. (2020) enables the efficient training of the deep neural network by allowing each layer of the network to be learned in isolation using local objective functions, in contrast to the end-to-end training. However, the maximum parallelization that existing local learning approaches can achieve is limited by the number of layers in the architecture

used. We address this limitation and introduce a new local learning method which enables the parallelization of the deep network at individual neuron-group level going beyond the layer level parallelization of existing approaches.

Through a series of experiments on various datasets, we aim to investigate the performance and behavior of neural networks trained using a purely local objective function at neuron-group level, in comparison to those trained using the layer-level objective functions and traditional joint adaptation approach. We will analyze the functional properties of the resulting networks, examining the role of feedback between layers and the impact on the overall performance of the model. Additionally, we will explore the conditions under which local loss function training may outperform the traditional approach, and provide insights on the potential practical implications for training deep neural networks. Our findings have the potential to inform the design and optimization of future neural network architectures, as well as provide a deeper understanding of the functional properties of deep networks. It is to be noted that this study only focuses on supervised learning approaches, specifically large-scale image classification task. The methodology and findings of the study are discussed in depth in Chapter 3.

1.2 Local Learning for Self-supervised Learning

Self-supervised learning, in which a model learns to perform a task using only unlabeled data, has the potential to greatly improve the efficiency and effectiveness of machine learning systems. However, traditional approaches to self-supervised learning often involve training an end-to-end model, which can be computationally intensive and may not scale well to larger datasets. To address this challenge, we propose the use of local learning methods for self-supervised learning tasks as a means of increasing parallelization and improving efficiency. Specifically, we apply the Decouple Greedy Learning (DGL) framework, as described in Belilovsky et al. (2020), to the popular SimCLR method T. Chen, Kornblith, Norouzi, and Hinton (2020).

This approach has the potential to parallelize the training of the models in self-supervised learning tasks by learning each layer of the network in parallel. However, we observe the issue of task-relevant information collapse in the initial layers of locally trained models, as described in Wang et al. (2021),

which poses a significant challenge in achieving optimal performance on locally trained models. This phenomenon results in degradation of performance as the initial layers fail to propagate maximum task-relevant information to the subsequent layers. To address this issue, Wang et al. (2021) proposes the use of a decoder model to reconstruct the input at every layer. However, this approach is not scalable and incurs a significant computational burden.

In order to overcome this challenge, we propose a new method that incorporates additional techniques to ensure maximum task-relevant information propagation to subsequent layers. Our hypothesis is that systematically increasing the problem difficulty layer by layer can facilitate maximum task-relevant information propagation. To test this hypothesis, our method proposes layerwise modification of the local loss function such that the problem difficulty increases with depth. By conducting experimental evaluations, we demonstrate the effectiveness of our method in improving the performance of self-supervised learning models. The results indicate that our method achieves performance that is comparable to that of traditional end-to-end training methods. Overall, this study contributes to the field by providing a new approach for improving the efficiency and scalability of self-supervised learning methods through the use of local learning frameworks. The methodology and findings of this study are thoroughly discussed in the Chapter 4.

1.3 Related Work

Sequential local learning, where a network is built up through greedily adding individual layers and solving local layerwise optimization problems, has been studied in a number of classical works Fahlman and Lebiere (1989); Ivakhnenko and Lapa (1965). These kinds of approaches were commonly used on simple datasets particularly in the case of unsupervised models (Bengio, Lamblin, Popovici, and Larochelle (2007); Vincent et al. (2010). Their use was motivated primarily by difficulties associated with joint optimization of deep networks such as the vanishing gradient problem. However, such methods fell out of favor with the advent of modern techniques to tackle joint deep network training such as improved initialization, residual connections He, Zhang, Ren, and Sun (2016a) and normalizationIoffe and Szegedy (2015). These sequential local learning techniques have been revisited recently by Belilovsky, Eickenberg, and Oyallon (2019a) which showed that this

approach can yield high-performance models on large datasets and architectures. Belilovsky et al. (2020) further demonstrated that this can be performed in the parallel local learning setting, where layers are learned simultaneously, allowing for improved distributed training of deep networks. It can even be done in an asynchronous manner if each layer is allowed to maintain a memory. Nøkland and Eidnes (2019) demonstrated an alternative loss function which yields improved performance in the local learning setting. Wang et al. (2021) further extended this idea, illustrating a novel regularization term that combats the collapsing of the representation towards the target supervised task through the use of an autoencoder.

Gomez et al. (2022); Xiong, Ren, and Urtasun (2020) introduce the idea of overlapping local layers and blocks while performing parallel local learning. This can improve performance but re-introduces locking constraints that prevent a number of distributed applications of parallel local learning, for example it is not directly compatible with asynchronous training as in Belilovsky et al. (2020). Laskin et al. (2021) has performed a study comparing a number of local learning techniques including interlocking backpropagation on several common large scale tasks.

Pyeon, Moon, Hahn, and Kim (2021) has proposed techniques for learning the architecture in the setting of decoupled learning. Techniques proposed in this work are compatible with our study and can be naturally combined.

Related to our work Veness et al. (2019) studied a local objective where each neuron solves a binary classification problem. Results were illustrated in an online-learning setting but have not been extended to the standard offline learning settings or to complex datasets such as imagenet.

Belilovsky et al. (2020); Choromanska et al. (2019); Lee, Zhang, Fischer, and Bengio (2015) considers local objective functions with globally generated targets. These, however, require feedback communication between the various layers and model components. Our work on the other hand focuses on local learning objectives and their extensions.

Jaderberg et al. (2017b) proposed to also use auxiliary networks to locally approximate a gradient in order to allow for parallel learning of neural network layers and blocks. However, the auxillary models must predict a high-dimensional and time-varying target, which creates a lot of instability, making it impractical in practice Belilovsky et al. (2020); Huo, Gu, and Huang (2018a).

Pipelining Y. Huang et al. (2018) is a systems-level solution that can alleviate to a degree the

issues introduced by backpropagation that prevent parallel training at different modules. However, this techniques does not remove the fundamental limits (often described as locks in Jaderberg et al. (2017b)) and thus does not allow for a full parallelization.

G. E. Hinton and Salakhutdinov (2006) introduced greedy layerwise unsupervised learning for training Deep Belief Networks (DBN). Bengio et al. (2007) proposed an algorithm for training deep architectures using a contrastive divergence objective, which follows the greedy layerwise training approach. Their method demonstrated improved performance over traditional unsupervised learning methods. Another example is the work of Erhan, Courville, Bengio, and Vincent (2010) who introduced the concept of pre-training deep architectures using stacked autoencoders, which utilizes a greedy layerwise training approach. This method also showed improved performance in various applications.

1.4 Contributions

This thesis presents several contributions to the field of deep learning, specifically in the area of distributed local learning. The main contributions of this thesis are as follows:

- We propose the extension of the recent Decoupled Greedy Learning (DGL) framework to include neuron-level groups, referred to as Grouped Neuron DGL (GN-DGL). This new approach increases the degree of parallelization that was previously possible with existing methods.

- We introduce the use of a diversity-promoting penalty to facilitate the learning of diverse feature representations among different neuron groups. Our experiments show that use of this penalty improves the performance of the GN-DGL method.

- We present a thorough comparison of different variants of the GN-DGL method with existing local learning methods such as DGL (Belilovsky et al., 2020) and InfoPro Wang et al. (2021). Our results demonstrate that the GN-DGL method outperforms these existing methods in certain scenarios.

- We also study the limitations of adapting the recent Decoupled Greedy Learning (DGL) framework to the SimCLR method. Our findings provide insight into the challenges of adapting this framework for self-supervised learning tasks.

- To address these limitations, we introduce a new method called Decoupled Greedy Self-supervised Learning (DGSSL), which improves the performance of locally trained SimCLR models.

- To evaluate the effectiveness of our proposed method, we conduct ablation studies focusing on variants of the DGSSL method. These variants include one that increases problem difficulty with depth, and another that decreases problem difficulty with depth. Subsequent ablation study focuses on the impact of using both positive and negative pairs of samples for layerwise loss modification, as well as the impact of only using positive pairs.

1.5 Working Papers

The work titled "Local Learning with Neuron Groups," presented by Patel, Eickenberg, and Belilovsky (2022) at the ICLR 2022 “From Cells to Societies: Collective Learning Across Scales Workshop" serves as a foundation for this thesis. Specifically, a substantial portion of Chapter 3 is drawn from the aforementioned work by Patel et al. (2022). Additionally, an extended version of Chapter 3 is currently under review for the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2023. Furthermore, efforts are underway to prepare Chapter 4 for publication. The author of this thesis served as the lead author to all of the aforementioned works, and the co-authors acknowledge the incorporation of these works within the thesis.

Chapter 2

Background

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of deep learning algorithm that are particularly well-suited for image and video analysis due to their ability to extract and recognize complex features in the input data (LeCun, Bengio, et al., 1995). CNNs are composed of multiple layers of interconnected nodes, with each layer responsible for extracting different types of features from the input. These layers are organized in a hierarchical manner, with lower layers extracting simple features such as edges and colors, and higher layers combining these features to recognize more complex objects and patterns (LeCun, Bengio, & Hinton, 2015).

One of the key features of CNNs is their use of convolutional layers, which are designed to automatically learn spatial hierarchies of features. These layers use a small, learnable kernel or filter that is convolved with the input data to compute a set of output features (G. E. Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). By applying multiple convolutional layers, CNNs can learn increasingly complex features of the input data, such as edges, shapes, and objects.

In addition to convolutional layers, CNNs also incorporate pooling layers which are responsible for reducing the spatial dimensions of the input data while maintaining the most important features. This property, known as translation invariance, allows CNNs to recognize objects in any position within the input data (Scherer, Müller, & Behnke, 2010). Pooling layers can be implemented using various techniques, such as max pooling, average pooling etc. Pooling layers also help to reduce the

number of parameters in the model, making it more efficient and easier to train (He, Zhang, Ren, & Sun, 2016b).

One of the major advantages of CNNs is their ability to be trained efficiently, even on large datasets (Krizhevsky, Sutskever, & Hinton, 2012). This is in part due to their use of convolutional and pooling layers, which reduce the number of parameters and computational cost of the model. In comparison to other types of neural networks, such as fully connected networks, CNNs have significantly lower training times and are able to achieve good performance with fewer training examples (Simonyan & Zisserman, 2014). Additionally, CNNs can be trained using techniques such as model parallelism, which involves dividing the model into smaller sub-models that can be trained in parallel, further improving training speed (Goyal et al., 2017). This can be achieved using various techniques such as data parallelism, where the input data is divided among multiple GPUs, or model parallelism, where different layers of the model are assigned to different GPUs (C.-C. Chen, Yang, & Cheng, 2018). Another approach is the use of local learning methods (Belilovsky, Eickenberg, & Oyallon, 2019b; Bengio et al., 2007; Nøkland & Eidnes, 2019; Wang et al., 2021), which we further discuss in subsequent sections.

Due to their ability to automatically learn hierarchical representations of the input data, CNNs have achieved state-of-the-art performance on a wide range of tasks, including image and video classification (Karpathy et al., 2014), object detection (Ren, He, Girshick, & Sun, 2015), and segmentation (Xu et al., 2018). Their efficiency and effectiveness make them a powerful tool for a wide range of applications, and techniques such as model parallelism and weight sharing can further improve the speed and scalability of these algorithms. They are also widely used in applications such as medical imaging (), facial recognition, and autonomous driving. These applications require the ability to recognize complex patterns in visual data, and CNNs are particularly well-suited to this task. As such, CNNs continue to be a vital and influential aspect of the field of deep learning. Due to their ability to automatically learn hierarchical representations of the input data, CNNs have achieved state-of-the-art performance on a wide range of tasks, including image and video classification (Karpathy et al., 2014), object detection (Ren et al., 2015), and segmentation (Xu et al., 2018). Their efficiency and effectiveness make them a powerful tool for a wide range of applications, and techniques such as model parallelism and weight sharing can further improve the speed and

scalability of these algorithms. They are also widely used in applications such as medical imaging (Litjens et al., 2017), facial recognition (Parkhi, Vedaldi, & Zisserman, 2015), and autonomous driving (Bojarski et al., 2016). These applications require the ability to recognize complex patterns in visual data, and CNNs are particularly well-suited to this task (Krizhevsky et al., 2012). As such, CNNs continue to be a vital and influential aspect of the field of deep learning (LeCun et al., 2015).

2.2 Supervised Learning

Supervised learning is a type of machine learning where the model is trained on labeled data, meaning that the correct output is provided for each example in the training dataset (Bengio, Courville, & Vincent, 2013; Krizhevsky et al., 2012). The goal of supervised learning is to learn a function that can map the input data to the corresponding outputs, so that the model can predict the output for new, unseen examples (Goodfellow, Bengio, & Courville, 2016). This is achieved through the use of algorithms that learn from the labeled training data and make predictions on new data.

Supervised learning algorithms can be classified into several categories, such as regression and classification algorithms (Mahesh, 2020). Regression algorithms are used to predict continuous values, such as the price of a house based on its size and location (Hastie, Tibshirani, Friedman, & Friedman, 2009). On the other hand, classification algorithms are used to predict discrete values, such as the type of animal in an image. These algorithms are designed to learn complex patterns in the data and make accurate predictions based on those patterns.

There are many popular supervised learning algorithms, such as decision trees (Kingsford & Salzberg, 2008), support vector machines (Cortes & Vapnik, 1995), and neural networks (LeCun et al., 2015). These algorithms have been extensively studied and applied in various domains, including natural language processing (Brown et al., 2020), computer vision (Krizhevsky et al., 2012), and speech recognition (G. Hinton et al., 2012). They have proven to be effective at solving a wide range of problems in these domains, making them an important tool in the machine learning toolkit.

One of the main challenges in supervised learning is the need for large amounts of labeled data, which can be time-consuming and expensive to obtain. Another challenge is the potential for overfitting, where the model performs well on the training data but poorly on new, unseen data.

To address these challenges, researchers have developed various techniques, such as regularization (Kukačka, Golkov, & Cremers, 2017) and cross-validation (Kohavi et al., 1995), to improve the generalization ability of supervised learning models. These techniques help to prevent overfitting and improve the performance of the model on new data.

Overall, supervised learning has proven to be a powerful tool for solving a wide range of problems in different fields. Despite its challenges, it continues to be an active area of research and development, with many exciting developments and applications yet to come.

2.3 Self-supervised Learning

Self-supervised learning has gained significant attention in recent years as a means of leveraging vast amounts of unlabelled data to improve the performance of machine learning models (Brown et al., 2020; Schuhmann et al., 2022). The basic idea behind self-supervised learning is to use the inherent structure of the data to create pseudolabels, or labels that are not explicitly provided by human annotators (Doersch, Gupta, & Efros, 2015). These pseudolabels are then used to train a neural network on a pretext task, which is a task that is not the final task of interest but is designed to learn useful representations for the final task.

There are two main approaches to self-supervised learning: generative and contrastive (Liu et al., 2021). Generative approaches involve training on tasks such as image-inpainting and video frame prediction, where the model is trained to generate missing parts of an image or predict future frames in a video (Kim, Woo, Lee, & Kweon, 2019; Pathak, Krahenbuhl, Donahue, Darrell, & Efros, 2016). These tasks encourage the model to learn rich representations of the data by forcing it to generate plausible samples.

Contrastive approaches, on the other hand, focus on learning close projections of augmented versions of the same sample and pushing far the projections of different samples (T. Chen et al., 2020; Jaiswal, Babu, Zadeh, Banerjee, & Makedon, 2020). This is done by training the model to differentiate between different augmentations of the same sample and different samples. This approach has been shown to be particularly effective for learning representations of images and videos.

Overall, self-supervised learning is a promising approach for leveraging unlabelled data to improve the performance of machine learning models. It has been used to improve the performance of various tasks, but it is important to be aware of its limitations. Further research is needed to better understand the strengths and limitations of self-supervised learning and to develop more effective methods for creating pseudolabels.

2.4 Contrastive Self-supervised Learning

Contrastive learning is a method which learns useful representations by comparison. In supervised learning, labels for each sample is provided in order to discriminate it from other samples. Whereas, in contrastive self-supervised learning, we use pseudolabels which try to pull the augmented versions of samples in a batch closer to each other and at the same time it also tries to learn to push the different samples far from each other.

This notion of similarity can also be used in supervised learning setup by mapping the representations from same class closer and vice versa. Hence, the contrastive learning can be applied to both supervised (Khosla et al., 2020) and self-supervised settings (T. Chen et al., 2020). In this thesis, we would only focus on the use of contrastive learning in context of self-supervised learning.

In self-supervised contrastive learning, a similarity metric is used to measure distances between two feature representations. In computer vision tasks, these representations are learned by an encoder network and fed to a contrastive loss function. In other words, a sample is taken from a dataset and few augmentations are applied to obtain an augmented version of the sample. Now, these two views of a sample are considered positive samples to train the encoder network and remaining samples from the dataset are regarded as negative samples. Such setup helps a model to learn useful representations by differentiating positive samples from negative samples. Such a method of learning requires ample number of negative examples in a batch/dataset to be able to learn useful representations. Based on this idea, it is understood why contrastive learning algorithms tend to benefit by training with larger batch sizes (T. Chen et al., 2020).

2.4.1 SimCLR: A Simple Contrastive Learning Framework

A simple framework for contrastive learning (SimCLR) paper (T. Chen et al., 2020) proposed a simplified framework for contrastive self-supervised learning, where the model learns by maximizing the agreement between positive samples obtained by applying appropriate transformations on data as shown in Figure 2.3.

An important component of a SimCLR framework is its data augmentation module. As shown in Figure 2.3, a sample x is taken from a dataset and a set of appropriate augmentations are applied on x to obtain two views x_i and x_j of a sample to be considered a positive pair. T. Chen et al. (2020) have shown that applying specific augmentations significantly affects performance of the model. Specifically, when random crop and color distortion is applied in sequence, the model produces highest top-1 accuracy on ImageNet dataset. On the other hand, applying sobel filtering, rotation, gaussian noise, and gaussian blur are not as effective as applying random crop, cutout, and color distortion for performance on ImageNet dataset.

Feature extractor $f(\cdot)$: A convolutional neural network based encoder network is used to extract feature representations from two augmented views of a sample. T. Chen et al. (2020) have adapted a commonly used ResNet architectures (He et al., 2016a) to extract representations h_i and h_j, where i and j indicates two different views of sample.

Projection head $g(\cdot)$: As shown in Figure 2.3, a projection head is attached to an encoder network $f(\cdot)$, which takes in a feature representation h_i and h_j as input to produce an embedding vector z_i and z_j to apply a contrastive loss. T. Chen et al. (2020) have used a one hidden layer MLP to apply non-linear transformations on representations for their experiments. Although, several results suggests that using a single layer MLP as a projection head in SimCLR degrades the performance of the model compared to using one hidden layer MLP. In addition to that, some experiments from the paper have also concluded that applying contrastive loss on raw representations deteriorates the performance of the network, which establishes the importance of having a projection head in SimCLR.

The encoder network along with the projection head $g(\cdot)$ is trained using an end-to-end backpropagation algorithm and a single contrastive loss function is applied on vector embeddings produced

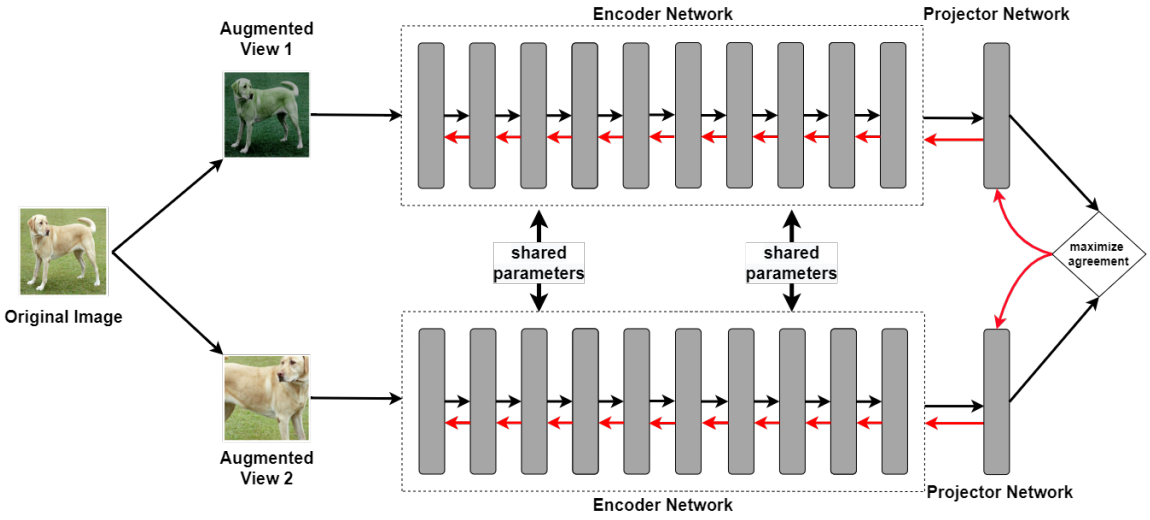


Figure 2.1: A simple framework for contrastive learning. Two augmented views x_i and x_j of a sample x are processed by a shared encoder $f(\cdot)$ to obtain representations h_i and h_j. A projector network $g(\cdot)$ takes these representations as input and obtain embeddings z_i and z_j. The contrastive loss is applied on z_i and z_j to maximize the agreement between the views.

by the projection head. In this work, we propose to alter this encoder network $f(\cdot)$ by dividing it into smaller encoder modules to train independently using separate local contrastive loss functions. We demonstrate that having different architectures of projection heads can significantly affect the performance of the locally trained model.

Evaluation method In SimCLR, during linear evaluation mode, the projection head $g(\cdot)$ is discarded and an MLP module is attached after the encoder network $f(\cdot)$. Based on architectural design, linear or non-linear transformation of representations h_i and h_j is performed by the MLP module to evaluate the standard categorical cross-entropy loss. This method is also known as linear probe evaluation, which is further detailed in Sec 2.7. Note that the parameters of encoder module $f(\cdot)$ are not updated during linear evaluation training mode. Though, there exist methods of evaluation which also fine tunes the parameters of the encoder module on a subset of dataset (Zhai, Oliver, Kolesnikov, & Beyer, 2019), which is commonly known as semi-supervised learning.

2.5 Backpropagation

Typically, a neural network is trained using an end-to-end backpropagation algorithm to jointly train all the layers of a network using a global loss function. Such joint training of all layers using a single loss function typically increases network capacity to learn a function for large-scale datasets efficiently. The Backpropagation algorithm is a well-known method used in the training of artificial neural networks (Rumelhart, Hinton, & Williams, 1985). It is an efficient algorithm that allows for the adjustment of weights in the network in order to improve the network's performance on a given task. The algorithm is based on the gradient descent optimization technique, which involves the calculation of gradients of the error function with respect to the network weights.

The Backpropagation algorithm operates in two phases: a forward phase and a backward phase. In the forward phase, the input data is fed through the network and the output is generated. In this phase, the input data is passed through each layer of the network in a sequential manner, with the weights of each layer being applied to the input data in order to produce the output of that layer. The output of each layer is then passed as the input to the next layer, until the output of the final layer is produced. This output is then compared to the desired target, and the error between the two is calculated. In the backward phase, the error between the output and the desired target is used to compute the gradients of the error function with respect to the weights. These gradients are then used to update the weights in the direction that minimizes the error. This process is repeated for each weight in the network, until the error is reduced to an acceptable level.

However, such sequential approach introduces several locking issues while using it to train large models as shown in Figure 2.2, which can be understood through the perspective of data dependencies (Jaderberg et al., 2017a). Forward locking occurs when the processing of a layer is dependent on the completion of processing in the previous layers. This means that the processing of each layer must be completed in a specific sequence before the next module can begin. Update locking expands upon this by requiring that all dependent layers must be processed in forward direction before any updates can be made to a particular layer. Backwards locking takes this one step further by requiring that all dependent layers be processed in both the forward and backward directions before a particular layer can be updated. This added constraint can significantly impact the efficiency of

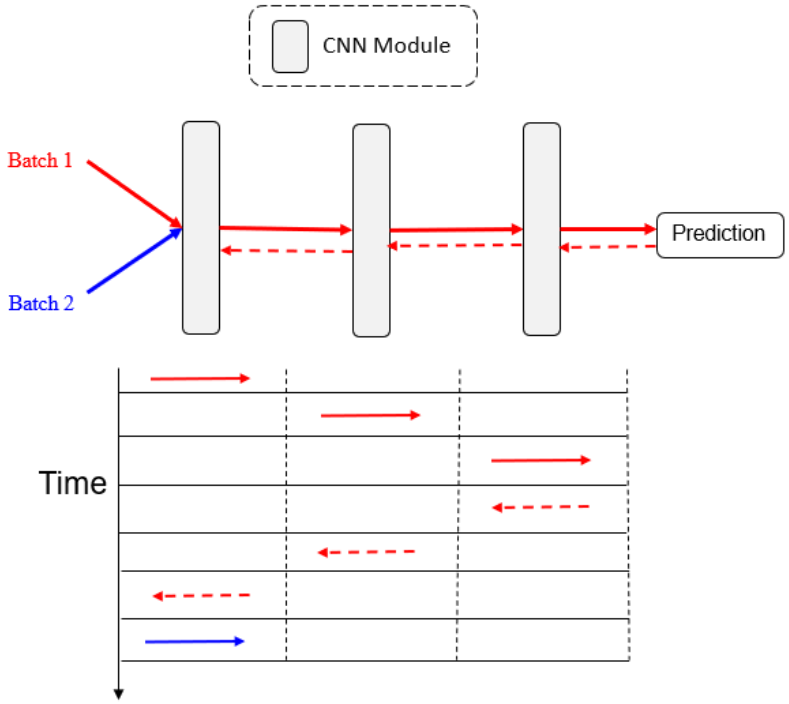


Figure 2.2: Data-flow diagram across time for standard backpropagation algorithm, highlighting the fundamental limitations of backpropagation algorithm: forward, update and backward locking.

the processing, as it requires additional computations to be completed before updates can be made. These locking problems can hinder the efficiency of neural network training and impede parallel processing capabilities.

To address this issue, researchers have attempted to introduce local loss functions for each layer or module, which can be trained using a sequential greedy learning procedure Bengio et al. (2007); Ivakhnenko and Lapa (1965). This approach allows each layer or module to be trained independently, which can help to reduce the computational and memory requirements of the overall training process. While this approach can be effective in some cases, it can also be somewhat limited in terms of the types of models that can be trained, and may not always achieve the same level of performance as traditional end-to-end training methods.

2.6 Distributed Training

Distributed training of neural networks is a common approach used in deep learning to improve the efficiency and effectiveness of training large models. This approach involves the use of multiple devices, such as multiple GPUs or multiple machines, to train a model in parallel. There are four main approaches to distributed training: data parallelism, model parallelism, pipeline parallelism and local parallelism as described in Laskin et al. (2020).

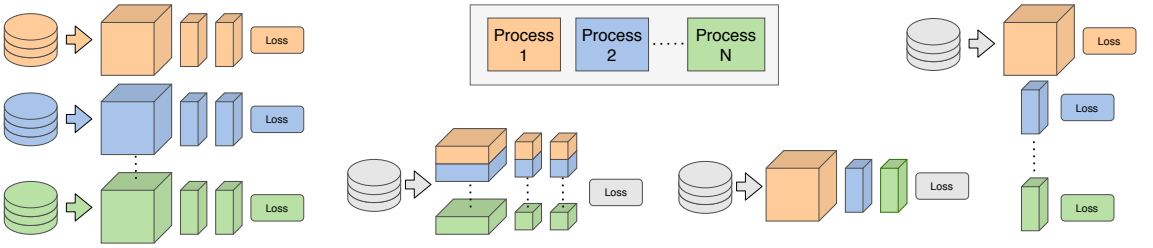


Figure 2.3: Approaches of distributed training in deep learning. Extracted from Laskin et al. (2020).

2.6.1 Data Parallelism

Data parallelism involves distributing the data across multiple devices and processing it in parallel. This approach allows for faster training times as multiple devices can process the data simultaneously. However, the model must be able to fit on a single device as it is replicated across all devices. This requires careful consideration of the data processing pipeline and how it can be effectively divided and distributed across devices, as well as the communication and synchronization between devices during training (Krizhevsky, 2014).

One major advantage of data parallelism is the ability to achieve faster training times. This is because multiple devices can be used in parallel to process the data, allowing for much faster training times (Goyal et al., 2017). Data parallelism can also be easier to implement, as it does not require significant changes to the model architecture. It is a common approach for training large models on multiple GPUs or across multiple machines in a distributed computing environment (Dean et al., 2012).

However, there are also some limitations to consider when using data parallelism. One limitation

is that the model must be able to fit on a single device, as it is replicated across all devices. This may limit the size and complexity of the model that can be trained using data parallelism (You, Gitman, & Ginsburg, 2017). There may also be communication overhead between devices, which can decrease the overall speed of training. Additionally, distributing the data across multiple devices may require significant changes to the data processing pipeline, which can be challenging and require additional resources.

2.6.2 Model Parallelism

Model parallelism involves dividing the model across multiple devices, allowing the model to be trained on larger datasets and leveraging the computational power of multiple devices for faster training times. This approach is useful for training very large models that would not fit on a single device, as it allows for the use of more computational resources to train the model. However, it can be more difficult to implement as the model must be specifically designed to be split across multiple devices. This requires careful consideration of the model architecture and how it can be effectively divided and distributed across devices, as well as the communication and synchronization between devices during training (Guan, Yin, Li, & Lu, 2019).

One major advantage of model parallelism is the ability to train very large models that would not fit on a single device. This is because the model is divided across multiple devices, allowing for the use of more computational resources to train the model. Model parallelism can also lead to faster training times, as multiple devices can be used in parallel to process the data or model. This can be particularly useful for training very large models or datasets that would take a long time to train on a single device (Gomez et al., 2020).

2.6.3 Limitations

Despite having many advantages, there are several limitations to consider when using distributed training in deep learning. One major disadvantage of data and model parallelism is the communication overhead that is required. In order to update the weights of the model, the gradients must be calculated and transmitted between the different parts of the model. This can lead to a significant amount of communication overhead, as the gradients need to be transmitted between different GPUs or devices

in order to update the model (C.-C. Chen et al., 2018). This communication overhead can slow down the training process and limit the scalability of the model, as it becomes more difficult to add additional GPUs or devices to the training process. This is because the more GPUs or devices that are added, the more gradients need to be transmitted and the greater the communication overhead becomes (Goyal et al., 2017). This can ultimately lead to a diminishing return on the performance gain from adding additional GPUs or devices, as the communication overhead becomes a significant bottleneck.

Another disadvantage of such approaches is the reliance on a central coordinator to collect and aggregate the gradients from different nodes in order to update the model. In a distributed training setup, the model is trained on multiple nodes or devices, and the gradients are calculated and transmitted to a central coordinator in order to update the model (Chilimbi, Suzue, Apacible, & Kalyanaraman, 2014). This central coordinator can become a bottleneck in the training process, as it must collect and aggregate the gradients from all of the different nodes in order to update the model (Konečnỳ et al., 2016). This can limit the scalability of the model and reduce the overall efficiency of the training process, as the central coordinator may become a bottleneck and limit the ability to add additional nodes or devices to the training process. This is especially problematic when training large and complex models, as the central coordinator may need to handle a large number of gradients and the communication overhead may become significant.

In addition to the communication overhead and reliance on a central coordinator, such approaches may also suffer from convergence issues. Due to the distributed nature of the training process, the gradients may be noisy or inconsistent, which can lead to slower convergence and reduced performance of the model. This can be especially problematic when training large and complex models, as the convergence issues may become more pronounced as the model size increases. Additionally, the convergence issues may be exacerbated by the communication overhead and reliance on a central coordinator, as these factors may introduce additional noise and inconsistency into the gradients and make it more difficult for the model to converge (Keuper & Preundt, 2016). Overall, these disadvantages of model parallelism and data parallelism can make it challenging to effectively train neural networks in these settings, and may require the use of alternative algorithms or optimization techniques in order to achieve good performance.

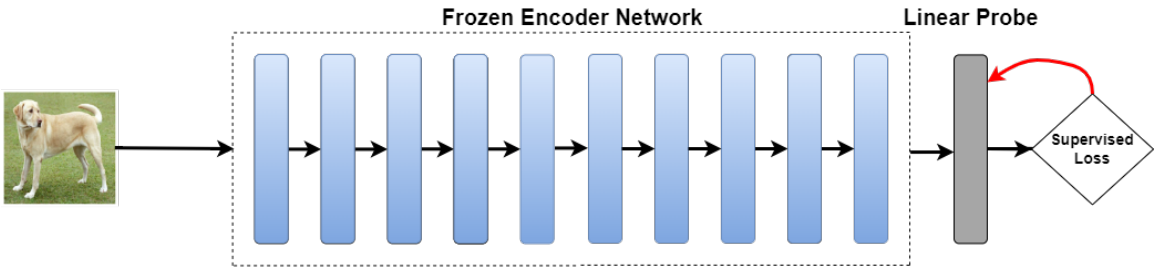


Figure 2.4: Linear Probe Evaluation: a linear classifier (probe) is trained while CNN encoder weights are frozen during this evaluation phase.

2.7 Linear Probe Evaluation

Linear probe evaluation is a method used to evaluate the representations learned by a convolutional neural network (CNN). The idea behind linear probe evaluation is to take the features learned by a CNN and use them as input to a linear classifier (probe) as shown in Figure 2.4. This classifier is trained to perform a task using the CNN features, and its performance is used as a measure of the quality of the CNN representations (Alain & Bengio, 2016). This approach has been used to evaluate the representations learned by CNNs in a variety of tasks, such as object recognition, face recognition, and natural language processing. Linear probe evaluation has been shown to be a useful tool for understanding the properties of CNN representations, and it has been used to identify the specific layers and units in a CNN that are most important for a given task.

2.8 Local Learning Methods

In this section, we describe the local learning framework of Nøkland and Eidnes (2019) and Decoupled Greedy Learning Belilovsky et al. (2020). Additionally, we also discuss some of the limitations of using purely local loss functions and introduce a previously explored solution to use an Information Propagation Loss as described in Wang et al. (2021).

Several inefficiencies of a fully end-to-end back-propagation algorithm have been reported in recent works. As described in Jaderberg et al. (2017a), mainly three inefficiency of the back-propagation algorithm prohibit parallelization are forward, backward, and update locking. Backward

locking requires the error signal to propagate through all subsequent modules before it can reach to a module. Update locking does not allow update of a module before a forward signal has propagated through the full network. And forward locking issue does not allow a module to propagate a signal forward until it has received signals from the previous layers.

Several methods have been proposed to address backward locking issues Choromanska et al. (2019); Huo, Gu, and Huang (2018b); Huo, Gu, Huang, et al. (2018); Lee et al. (2015); Nøkland (2016). Jaderberg et al. (2017a) describes update unlocking as far more severe issue and proposes Decoupled Neural Interfaces (DNI). DNI achieves update unlocking by directly predicting gradients for each module using an auxiliary network depending only on the input. The gradient of weights can be high-dimensional as the model size increases. Hence, this method does not scale well.

A scalable approach to update unlocking was recently proposed in Belilovsky et al. (2020). We describe this approach and its limitations briefly in the following section.

2.8.1 Decoupled Greedy Learning (DGL)

Local learning framework of Decoupled Greedy Learning (DGL) proposes to decouple each layer or module to train in isolation using an independent objective function as shown in Figure 2.5. Here, each module is trained in-parallel online along with the corresponding auxiliary module, which essentially addresses the fundamental limits of backpropagation algorithm: forward, update and backward locking. This approach significantly helps to reduce memory consumption and training time by parallelization. Although training all layers jointly using a global objective function ensures efficient adaptation to function for large datasets to achieve high performance, DGL has been shown to work well on large-scale image classification tasks under supervised learning settings.

Another recent work in this direction Wang et al. (2021) shows that training each layer or module using local loss functions similar to end-to-end loss function results in collapse of task-relevant information in early layers. To address this issue of information collapse, they propose an Information Propagation Loss (InfoPro), which adds a regularizer term to the standard local loss function. The regularizer term tries to reconstruct the input at each layer or a module using an auxiliary decoder network, effectively introducing two separate auxiliary heads for each local module. It has been shown to achieve high performance in large-scale image classification tasks and outperforming

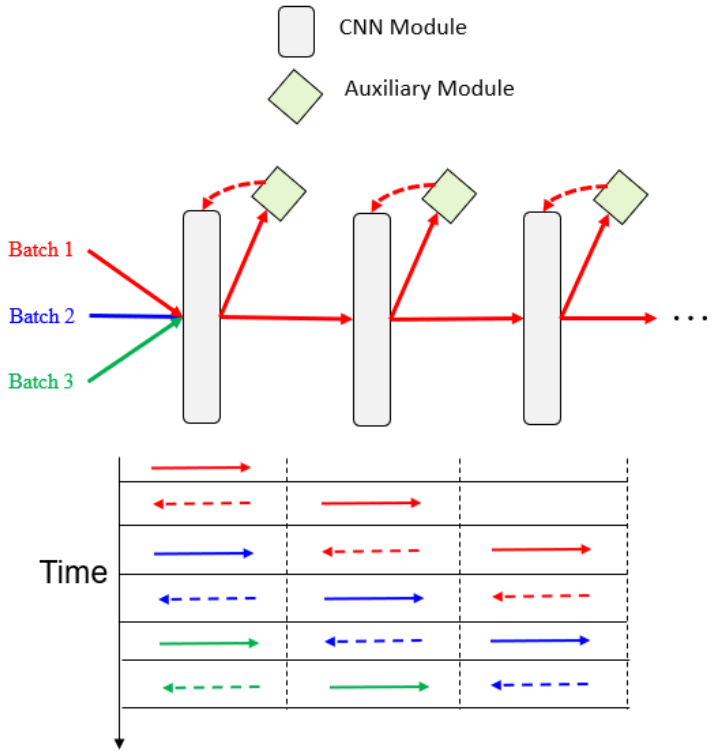


Figure 2.5: Local learning framework of Belilovsky et al. (2020); Nøkland and Eidnes (2019), where each module is trained in-parallel online using local auxiliary loss.

previously proposed Decoupled Greedy Learning (DGL).

Unfortunately, this approach to address the issue of information collapse is not computationally scalable, as the auxiliary decoder module consists of multiple convolutional layers which operates on the upsampled version of feature representations. Size of the decoder module is dependent on the size of the model to maintain the high performance on a given task.

In this thesis, we extend the local learning framework of Decoupled Greedy Learning by introducing another dimension of modularity to increase parallelization of the models. We also study the application of local learning frameworks in self-supervised learning settings especially with SimCLR to potentially achieve parallelization and reduce training times.

Chapter 3

Local learning with Neuron Groups

3.1 Introduction

Neural networks are typically trained by stochastic gradient descent in combination with the back-propagation algorithm to optimize a monolithic objective function jointly for all the components (G. Huang, Liu, Pleiss, Van Der Maaten, & Weinberger, 2019; Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015) . This learning algorithm allows joint adaptation of all layers and neuron connections in a network based on a global objective function. It is typically believed that this joint adaptation is essential to obtain a high performance for large-scale data sets as joint adaptation towards an overarching objective allows the individual layers and neurons to adapt their functionality efficiently. On the other hand, several recent works have studied the idea to use a purely local loss function, where there is no feedback between the independent layers. These approaches are based on the classic sequential greedy learning procedure Bengio et al. (2007); Ivakhnenko and Lapa (1965) but allow the different model components to simultaneously learn their model parameters. Surprisingly, relying purely on this forward communication, high-performance models can be constructed Belilovsky et al. (2020); Nøkland and Eidnes (2019). These observations have implications on both the functional properties of high-performance deep networks as well as practical implications for distributed training of neural networks. If layerwise learning can still bring high performance, this raises the question: to what degree do neural network components need to be trained jointly?

We can obtain some inspiration from biological neural systems. Biological neural systems rely on extremely localized synaptic updates, often modeled as Hebbian learning Hebb (1949) or adaptations thereof. The primary ways for a biological neural system to incorporate global information into learning is by way of feedback/recurrent neural connections, which can bring information that was computed at a later stage back to earlier neurons and hence be incorporated by local learning. Local learning in biological systems includes spatially local processing in retinotopic maps, which inspired locally connected architectures such as convolutional networks. Further, in biological systems, there often exist parallel pathways that perform different but somewhat overlapping computations (e.g. magno- and parvocellular visual pathways). We take inspiration from this idea and translate it to a channelwise splitting of computations such that the different components can be efficiently hosted on different processing units.

In this work we investigate the limits of using supervised local loss functions, moving them from the layer level to the “neuron group" level. Specifically, we consider the case where non-overlapping groups of neurons within a network layer each have their own local objective function and optimize their parameters in isolation. Our investigation reveals that networks can still demonstrate high performance despite this modification. We also investigate how to encourage these decoupled networks to learn diverse behavior (i.e. learn different representations from each other) based solely on forward communication. We illustrate the practical applications of this by comparing the proposed method to existing local learning approaches in the context of total training time, inference time, and model performance.

3.2 Methods

3.2.1 Background and Notation

In traditional approaches, the parameters of each layer of the network are learned sequentially, which can be time-consuming and may not fully utilize available resources. The local learning framework introduced by Belilovsky et al. (2020); Nøkland and Eidnes (2019) presents a promising solution to the challenge of training deep neural networks efficiently. These local learning frameworks address this issue by proposing the simultaneous optimization of the parameters at each layer j,

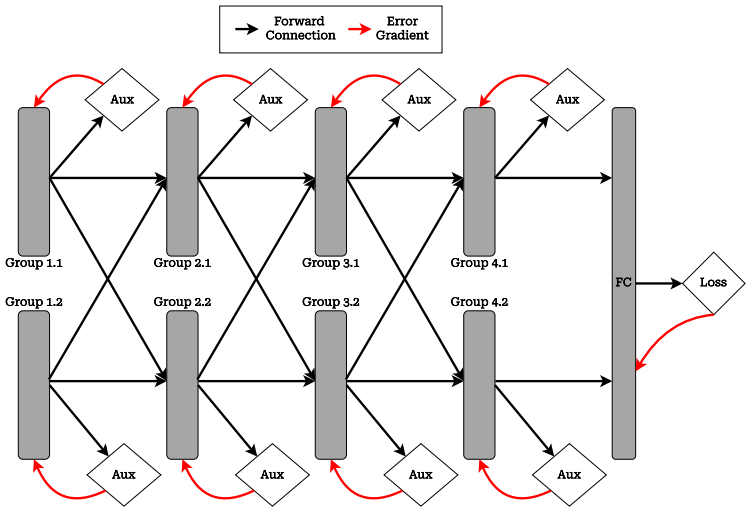


Figure 3.1: Grouped Neuron DGL, each layer and neuron group has a local objective. All local groups are learned in parallel, feeding their output directly to the next layer's groups.

denoted as θ_j. Instead of training each layer of the network sequentially, this framework suggests that the parameters of each layer should be learned jointly and in parallel. This approach allows for more efficient optimization of the parameters in terms of training time.

To formalize this concept, we must first introduce some notation. Let us consider an input to a neural network, denoted as x_0. This input is then passed through a series of layers, each performing a set of operations denoted as $f_{\theta_j}(x_{j-1})$, where θ_j refers to the parameters of the network at layer j. In addition to these operations, a local loss function is also applied to the representation x_j at each layer, denoted as $\mathcal{L}(y, x_j; \gamma_j, \theta_j)$. The parameters γ_j in this loss function correspond to those of an auxiliary network.

One important aspect of this framework is the introduction of the auxiliary network, which is used to compute the parameters θ_j along with the auxiliary network parameters γ_j as mentioned in the local loss function above. The auxiliary network is trained along with the corresponding layer from the main network, and its purpose is to provide additional information to the local loss function that can help to improve the optimization process. For example, the auxiliary network may be trained to predict the labels of the input data from the layer representations x_j, which can then be used to guide the optimization of the corresponding layer in the main network.

Algorithm 1: Grouped-Neuron DGL

Input: Mini-batches $\mathcal{S} \triangleq \{(x_0^t, y^t)\}_{t \leq T}$
Initialize Parameters $\{\theta_j^i, \gamma_j^i\}_{j \leq J, i \leq G}$.
for $(x_0, y) \in \mathcal{S}$ **do**
  for $j \in 1, ..., J$ **do**
    for $i \in 1, ..., G$ **do**
      $x_j^i \leftarrow f_{\theta_j^i}(x_{j-1})$.
      Compute $\nabla_{(\gamma_j^i, \theta_j^i)} \hat{\mathcal{L}}(y, x_j^i; \gamma_j^i, \theta_j^i)$.
      $(\theta_j^i, \gamma_j^i) \leftarrow$Update params (θ_j^i, γ_j^i).
    end
    $x_j \leftarrow \|_{i=1}^{G} x_j^i$.
  end
end

3.2.2 Grouped Neuron DGL (GN-DGL)

In this section, we introduce a new method for parallelizing deep neural network training called Grouped Neuron DGL (GN-DGL). The primary motivation for this method is to improve the efficiency of the training process by breaking down the objective function into smaller, parallelizable components.

In order to achieve this goal, we propose to divide each layer into smaller groups that have isolated losses. This approach builds upon the concept of layerwise local learning. We introduce a group-wise local loss term as $\mathcal{L}(y, x_j^i; \gamma_j^i, \theta_j^i)$. Here, x_j^i refers to a subset of the overall representation x_j, and the parameters γ_j^i and θ_j^i correspond to those of an auxiliary network and the neuron group i in the layer j, respectively. By dividing the objective function in this manner, we are able to parallelize the optimization process even further, potentially leading to greater efficiency.

One key aspect of our GN-DGL method is the way in which we learn each of these groups in parallel. Rather than sequentially optimizing each group as has been done in previous work Belilovsky et al. (2019a), we propose to learn each group online, as has been demonstrated in other approaches such as Belilovsky et al. (2020); Nøkland and Eidnes (2019). Our approach is illustrated in Algorithm 1, where we optimize a greedy objective for each neuron group. It is important to note that during a forward pass, each neuron group i at layer j receives as input the output representations x_{j-1}^k from all groups of the previous layer $j-1$ (i.e. all k in layer $j-1$). This is illustrated in Figure 3.1. We note that in our implementation, the last fully connected layer is decoupled from the

previous layer's neuron groups and is divided into its own groups, which are trained using the same approach as the other layers.

Algorithm 2: Stop-Gradient GN-DGL

Input: Mini-batches $\mathcal{S} \triangleq \{(x_0^t, y^t)\}_{t \leq T}$
Initialize Parameters $\{\theta_j^i, \gamma_j^i\}_{j \leq J, i \leq G}$.
for $(x_0, y) \in \mathcal{S}$ **do**
  for $j \in 1, ..., J$ **do**
    for $i \in 1, ..., G$ **do**
      $x_j^i \leftarrow f_{\theta_j^i}(x_{j-1})$.
      $z_j^i = sg\big[\big\|_{k \neq i} x_j^k\big]$
      Compute$\nabla_{(\gamma_j^i, \theta_j^i)} \hat{\mathcal{L}}(y, x_j^i, z_j^i; \gamma_j^i, \theta_j^i)$.
      $(\theta_j^i, \gamma_j^i) \leftarrow$Update params (θ_j^i, γ_j^i).
    end
    $x_j \leftarrow \big\|_{i=1}^{G} x_j^i$.
  end
end

3.2.3 Stop-Gradient Grouped Neuron DGL

The second method, an extension of GN-DGL method, loosens the communication restrictions between neuron-groups by allowing each neuron group to send its output representations to the auxiliary networks of the other neuron groups in the same layer. However, this is only a forward communication and no information is sent back to the previous layers during backward pass of the optimization process. This is achieved by introducing a stop-gradient operation before the output representations are passed to the auxiliary networks, as shown in Algorithm 2. This ensures that no gradients can flow back through the communication channels during the backward pass, while still allowing the auxiliary networks to make use of the information contained in the gradient-decoupled output representations of the previous layer neuron groups. We refer to this approach as the Stop-Gradient Grouped Neuron DGL (Stop-Gradient GN-DGL)

It is worth noting that the Stop-Gradient GN-DGL method does come with some additional communication overhead due to the need to transmit the output representations of neuron groups. However, we believe that the potential performance benefits outweigh this cost in many cases. To reduce the communication cost, we suggest spatially pooling the output representations before

sending them to the auxiliary networks. This reduces the size of the data being transmitted, while still preserving some of the important information contained in the output representations.

We believe that the Stop-Gradient GN-DGL method has the potential to improve the overall performance of deep neural networks by allowing each neuron group to make use of more context when making its own predictions. In our experiments, we will compare the performance of the Stop-Gradient GN-DGL method to the GN-DGL.

3.2.4 Grouped Neuron DGL with diversity-promoting penalty

An expected weakness of local group losses is that representations learned in different groups will be correlated and redundant to each other, not permitting optimal aggregation of predictive power. We attempt to address this by introducing a diversity-promoting penalty. Specifically, we allow auxiliary modules to send their softmax output distributions to each other. We add a diversity-promoting term in the loss function which encourages diversity locally, based solely on communication of softmax layer outputs between auxiliary modules.

To achieve this, we utilize a variant of the penalty studied in Dvornik, Schmid, and Mairal (2019). During training, the class probabilities for each group is encouraged to be close to the target vector with one non-zero entry for class label. Even though the highest entry of the class probabilities p_j^i is used to make the prediction about class, the other entries can also be used to extract useful information from the network. To represent it formally, we consider each neuron group of the network and their corresponding auxiliary modules to be parameterized by θ_j^i and γ_j^i respectively, where i and j indicates the group index and layer index respectively. Each group leads to the class probabilities $p_j^i = \text{softmax}(f_{(\theta_j^i, \gamma_j^i)}(x_{j-1}))$. Now consider $\hat{p}_j^i = \frac{p_j^i \odot m_y}{\|p_j^i \odot m_y\|_1}$ where m_y is a vector with zero at position y and 1 otherwise.

Now, for a group a in the layer j, we compute the diversity-promoting penalty with each remaining group b in the same layer as shown in Eq. 1,

$$\phi(\hat{p}_j^a, \hat{p}_j^b) = \texttt{sim}(\hat{p}_j^a, sg[\hat{p}_j^b]) \tag{1}$$

Here, sg denotes the stop-gradient operation and $\texttt{sim}$ is the cosine similarity. It is important to

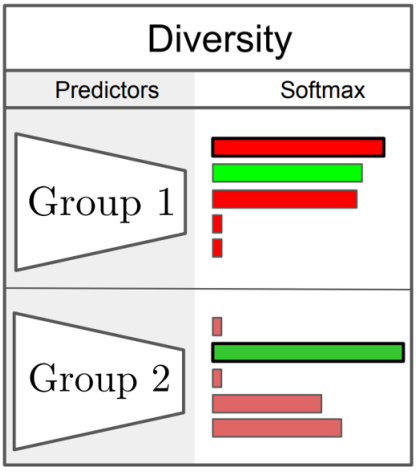


Figure 3.2: To encourage diverse feature learning among different neuron-groups, each neuron-group within a layer is encouraged to output different softmax activations for non-target labels.

note here that to estimate the diversity-promoting penalty for group a with all remaining groups b of the same layer, we use a gradient-decoupled version of $\hat{p}_j^b$. This means that no gradient signal needs to be sent back. This penalty encourages diversity by pushing the non-target-label softmax activations of the different groups away from each other as shown in Figure 3.2. This penalty only produces negligible overhead in communication between auxiliary modules.

Layerwise ensemble evaluation Typically, the prediction of a neural network is given by the softmax outputs of the last layer of the network. Whereas, the Layerwise Ensemble Evaluation method allows us to assess the performance of a neural network by aggregating the predictions made by different groups of neurons at different depths within the network. Following Belilovsky et al. (2019b) we investigate how each group performs on the task and evaluate how each group of neurons within the network contributes to the overall prediction made by the network.

To begin, we consider the output of the softmax layer, denoted by $\boldsymbol{l}_{ij}$, produced by each group of neurons. Here, i and j represent the indices of the neuron group over width and depth, respectively. We then take a weighted mean of all of these vectors to obtain the final output, l_e, as shown in Eq. 2.

$$l_e = \frac{1}{K \times G} \sum_{i=1}^{G} \sum_{j=1}^{K} \alpha_{ij} \boldsymbol{l}_{ij} \tag{2}$$

Here, G is the number of neuron groups and K is the number of layers. The weights α_{ij} are kept constant for the groups within a given layer, but are varied for different layers using the weighting scheme proposed in Belilovsky et al. (2019b).

This method allows us to understand the performance of each neuron group independently, as well as how they contribute to the overall prediction made by the network. By evaluating the performance of the network in this way, we can identify any potential issues or areas for improvement within each neuron-group. This can be particularly useful in the context of local learning, where the network is trained on separate local loss functions.

3.3 Experiments and Results

Our experimental results start with an initial set of ablations and then focus on evaluations of performance tradeoffs in model-parallel learning setting. First, we study a simple VGG6a model using layerwise and group-neuron level local loss functions. This allows us to establish basic performance characteristics of this approach compared to layerwise training. In the second set of experiments, we focus on an application to model-parallel training, comparing our method in the case of multi-layer modules to the recent methods such as Decoupled Greedy Learning (DGL) Belilovsky et al. (2020) and local learning with Information Propagation (InfoPro) Wang et al. (2021).

Datasets We perform our ablations using the popular CIFAR-10 dataset used in many prior works Belilovsky et al. (2020); Huo, Gu, and Huang (2018a) in order to obtain intuition on the performance of the various components. CIFAR-10 consists of 60,000 32x32 color training images and 10,000 test images, each belonging to one of 10 classes. In addition to CIFAR-10, we also use CIFAR-100 and a downsampled version of the Imagenet dataset (denoted as Imagenet32 Chrabaszcz, Loshchilov, and Hutter (2017)) in our experiments studying trade-offs in model-parallelism criteria. CIFAR-100 is similar to CIFAR-10, but has 100 classes instead of 10. Imagenet32 is a version of the Imagenet dataset that has been downsampled to 32x32 resolution, making it more computationally feasible for certain types of models. We believe that by utilizing a diverse set of datasets for our experiments, we can more thoroughly understand the implications of our findings and how they may apply in different

contexts.

Training hyper-parameters For experiments in Sec 3.3.1, we utilized the Adam optimizer Kingma and Ba (2014) for training our networks. We set the batch size to 128 and trained the networks for a total of 150 epochs. After tuning the hyperparameters, we found a specific learning rate decay schedule which improved the overall performance of the method. Specifically, we set the initial learning rate to 0.001 and decreased it to 0.0005 and 0.0001 at epochs 50 and 100, respectively. In addition, we utilized a dropout rate of 0.01 to prevent overfitting and improve generalization.

For the experiments in Sec 3.3.2, we adopted the hyper-parameters from a previous study Wang et al. (2021). These included the use of the SGD optimizer with a Nesterov momentum of 0.9 and an initial learning rate of 0.8. The models were trained for total 160 epochs. We employed cosine learning rate annealing to progressively decrease the learning rate as training progressed. The batch size was set to 1024 and we utilized an ℓ_2 weight decay ratio of 1e-4 to regularize the model.

3.3.1 Layerwise and Grouped Neuron DGL

The VGGNet used in the experiments, denoted VGG6a, consists of six layers (four convolutional and two fully connected) and is taken from Nøkland and Eidnes (2019). The convolutional layers have 128, 256, 512, 512 channels respectively, and 8192, 1024 features in the last two fully connected layers. The VGG6a architecture used in this experiment is shown in Figure 3.3. We note that the architecture presented in the figure is based on DGL framework of Belilovsky et al. (2020). To report DGL results, we train each layer with its own auxiliary network as described in Belilovsky et al. (2020). For GN-DGL experiments, we further split the layers widthwise into 2 to 10 neuron groups. The neuron groups within each layer and their auxiliary networks are trained locally by the auxiliary loss. The details about widthwise splitting are further elaborated below at the end of Sec 3.3.1. Each neuron group has its own auxiliary loss except the last fully connected layer, as shown in Figure 3.1. We use the same auxiliary network design and loss (termed *predsim*) as Nøkland and Eidnes (2019).

We define naive splitting where each neuron group is trained locally by their corresponding auxiliary networks and there is no encouragement of diversity among local modules from the same layer. Results of naive splitting experiments on VGG6a is presented in Table 3.1. Here the increasing

number of group size is indicated with G-Group, where G is the minimum number of groups in any layer. We also report the DGL Belilovsky et al. (2020) test accuracy for VGG6a, where the network is only divided depthwise and each layer is a local module in itself with their own auxiliary network and loss. We observe that as we increase G, the test accuracy slightly decreases (which is expected). We believe the decrease in accuracy is mainly due to correlated feature, arising from each neuron group being trained in isolation.

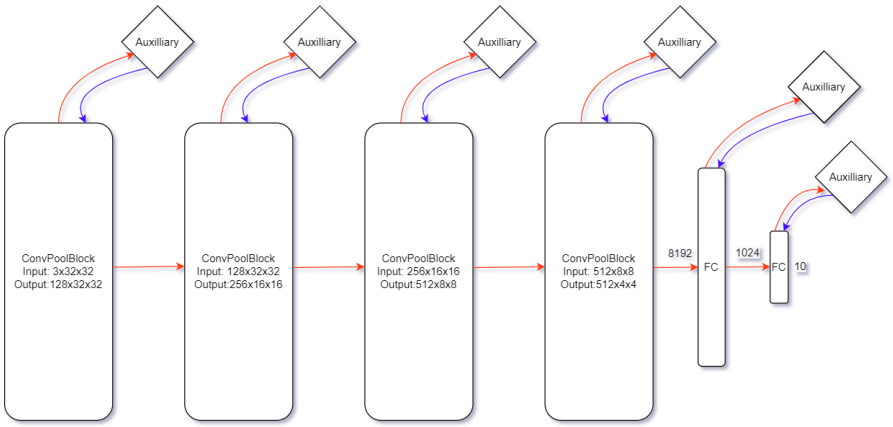


Figure 3.3: Architecture of VGG6a described with local learning framework of DGL Belilovsky et al. (2020).

Method	DGL	2-Group	4-Group	6-Group	8-Group	10-Group
Test Accuracy	92.25	91.63	90.55	89.8	89.07	88.91

Table 3.1: Grouping of neurons in each layer of VGG6a. We observe with increased number of groups, there is performance degradation, however a surprisingly high overall accuracy can be maintained despite a lack of communication across neuron groups. We note that the presented results do not utilize stop-gradient techniques, nor a local diversity-promoting penalty.

Method	DGL	6-Group GN-DGL	6-Group GN-DGL with diversity	6-Group GN-DGL Stop Gradient + diversity
Test Accuracy	92.2	89.8	90.6	91.9

Table 3.2: Ablating the effect of local diversity-promoting penalty and use of stop-gradient communication in GN-DGL.

	VGG6a	2xVGG6a	3xVGG6a
DGL	92.25	93.27	93.27
6-Group GN-DGL	89.8	92.6	93.28

Table 3.3: Ablating the effect of width in GN-DGL. We observe that as width increases, using GN-DGL begins to yield performance comparable to DGL, with increased parallelization.

Diversity of features and Stop-Gradient In order to improve performance of the overall model, we permit forward communication between within-layer modules. Specifically we allow softmax activations and gradient-decoupled pre-auxiliary module features to be sent across local modules. This corresponds to the stop-gradient and diversity-based approaches discussed in Sec 3.2.3 and 3.2.4. In Table 3.2, we show the ablations which demonstrate that these additions consistently improve the performance of the model allowing it to nearly recover the DGL performance.

Increasing Width We also investigate the effect of increasing width in Table 3.3, we observe that increasing the width of a 6-group GN-DGL can bring the performance closer to the DGL performance.

Widthwise splitting of VGG6a We describe the widthwise splitting on the VGG6a architecture. Specifically, we focus on the splitting of the first three convolutional layers. First, we introduce the 2-Group configuration, in which the first three convolutional layers are split in a (2,4,2) pattern. This means that the first convolutional layer is divided into two neuron groups, the second into four, and the third into two. This configuration allows for a balance between preserving the original architecture and introducing a degree of parallelization.

Next, we introduce the G-Group configurations for $G \in \{4, 6, 8, 10\}$. In these configurations, the splitting pattern of (2,4,2) is multiplied by $G/2$. For example, in the 6-Group configuration, the split is (6,12,6). These configurations allow for a greater degree of parallelization as G increases. Our goal in using these splitting configurations is to evaluate the effects on the performance and efficiency of the method as G increases.

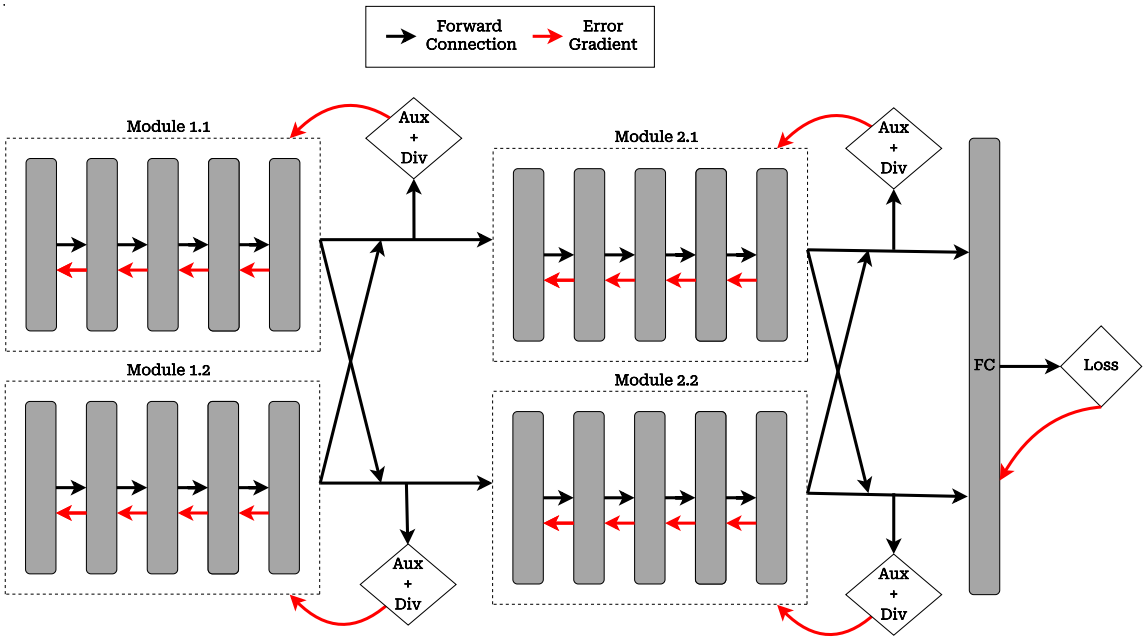


Figure 3.4: Illustration of our approach. Multi-layer modules are used in combination with a gradient decoupled output and diversity-promoting penalty.

3.3.2 Multi-layer Grouped Neuron DGL

In the field of deep learning, it is well-known that training a neural network using all of the layers at once can lead to better performance compared to training each layer independently (fully layerwise training) as shown in Belilovsky et al. (2020); Wang et al. (2021). According to the Wang et al. (2021), it has been shown that the initial layers of a the network are not able to effectively transmit enough task-relevant information from the input to the subsequent layers when they are trained independently. This can hinder the model's performance. To address this issue, in this section, we extend our splitting techniques to modules, which are defined as collections of layers, and sub-modules: sub-networks that can be grouped together to form a module as shown in Figure 3.4.

Unlike the previous section, for multi-layer versions of GN-DGL, we divide the network in such a way that we get K and G local modules depth-wise and width-wise respectively, resulting in a total of $K \times G$ local modules. Here, K corresponds to depth-wise splitting used in Wang et al. (2021), on whose experimental setup we base this experiment set. Each local module consists of approximately same number of layers and same number of channels in a particular layer and they are trained in isolation to each other as shown in Figure 3.4.

In addition to improving performance, we note that when performing multi-layer splits in the context of the CNN, the inference time of the overall network is decreased. This can provide an additional benefit in applying GN-DGL. By dividing the network into local modules, there are fewer cross-connections and computations, resulting in a faster evaluation time. This is particularly useful for applications where real-time performance is a priority, such as in image and video processing.

Training speed up comparison with end-to-end (Higher is better)										
KxG	**Infopro**		**DGL**		**Multi-layer GN-DGL (with Stop-Gradient & Diversity)**					
					G=2		G=4		G=8	
	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up
1	93.01	1	93.01	1						
2	92.35	1.833	91.47	1.894	91.41	3.674				
4	91.25	2.625	88.62	3.622	89.68	6.304	89.69	11.982		
8	88.5	3.856	85.53	6.456	87.94	10.667	89.27	17.445	88.86	28.531
16	87.02	5.218	83.87	11.609	85.37	16.426	88.11	24.522	88.96	33.736
32					83.38	23.072	85.09	30.47	87.86	39.333

Table 3.4: ResNet-32 on CIFAR-10 (Training speed up comparison)

Inference speed up comparison with end-to-end (Higher is better)										
KxG	**Infopro**		**DGL**		**Multi-layer GN-DGL (with Stop-Gradient & Diversity)**					
					G=2		G=4		G=8	
	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up	Test Acc.	Speed up
1	93.01	1	93.01	1						
2	92.35	1	91.47	1	91.41	1.961				
4	91.25	1	88.62	1	89.68	1.899	89.69	3.777		
8	88.5	1	85.53	1	87.94	1.785	89.27	3.448	88.86	7.03
16	87.02	1	83.87	1	85.37	1.612	88.11	2.936	88.96	5.824
32					83.38	1.338	85.09	2.324	87.86	4.336

Table 3.5: ResNet-32 on CIFAR-10 (Inference speed up comparison)

Discussion of Baseline Methods Here, we compare the proposed approach with other local-learning methods such as Decoupled Greedy Learning (*DGL*) Belilovsky et al. (2020) and local learning with Information Propagation (*InfoPro*) Wang et al. (2021). The other local-learning approaches only split the network across depth, unlike our approach of splitting across depth as well as width. Such splitting across both directions gives several advantages in performance and computation speed. We consider other local-learning methods as a special case of G = 1 for comparison. We note an inefficiency of the InfoPro method is its decoder model that tries to reconstruct the input to the network at each layer, leading to high computational cost at deeper layers

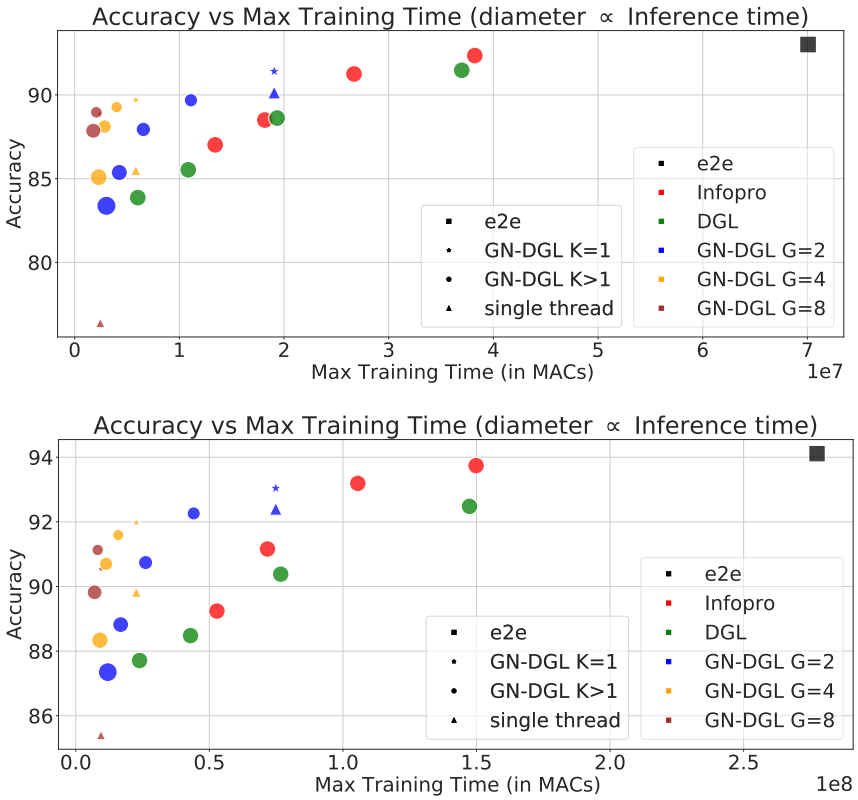


Figure 3.5: CIFAR-10 results. Accuracy versus maximum training time in MACs for a given node in a theoretical distributed scenario for various local learning methods. We observe that for both ResNet-32 (top) and a wider ResNet-32x2 (bottom), the GN-DGL leads to better tradeoffs in training time versus accuracy. The size of the bubbles is proportional to the inference time of the models.

(due to spatially upsampling a high number of channels). In addition we provide as reference the performance of a model when it is divided in width by G and trained end to end. We denote this baseline as *single thread*. We also report the performance of the model corresponding to G with K=1, which serves as a baseline of running individual sub-networks trained independently and recombined by a fully connected layer. Finally we denote the end to end training baseline as $e2e$.

Metrics We have carefully considered a number of different metrics for comparison in our setup. We believe that total training time, inference time, and final performance are the three most important factors to consider, as they all have a significant impact on the overall effectiveness of any local learning method. To accurately compare the training time of different methods, we have chosen to use the maximum computation time of any sub-component in multiply–accumulates (MACs) as

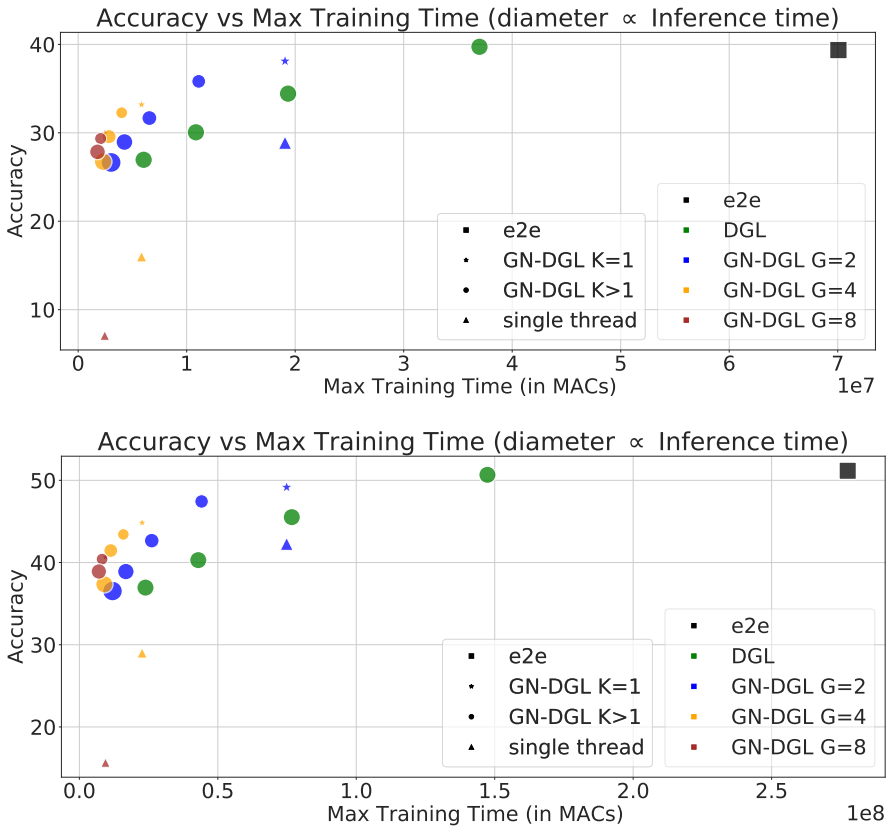


Figure 3.6: Imagenet32 results. Accuracy versus maximum training time in MACs for a given node in a theoretical distributed scenario for various local learning methods. We observe that for both ResNet-32 (top) and a wider ResNet-32x2 (bottom), the GN-DGL leads to better tradeoffs for this complex dataset. InfoPro is not shown as it yielded very poor performance on this dataset.

our measure. This allows us to fairly compare the training times of different methods, as it takes into account the full complexity of each method, and is the true measure of overall time required to complete training in local learning settings. In addition to training time, we have also considered the inference time of different methods. This is important because it can have a significant impact on the real-world performance of a method, particularly in situations where quick response times are required. Finally, we have also considered the overall performance of different methods as a key metric for comparison. This allows us to determine the overall effectiveness of a method, and to identify those methods that are likely to be the most useful in practice.

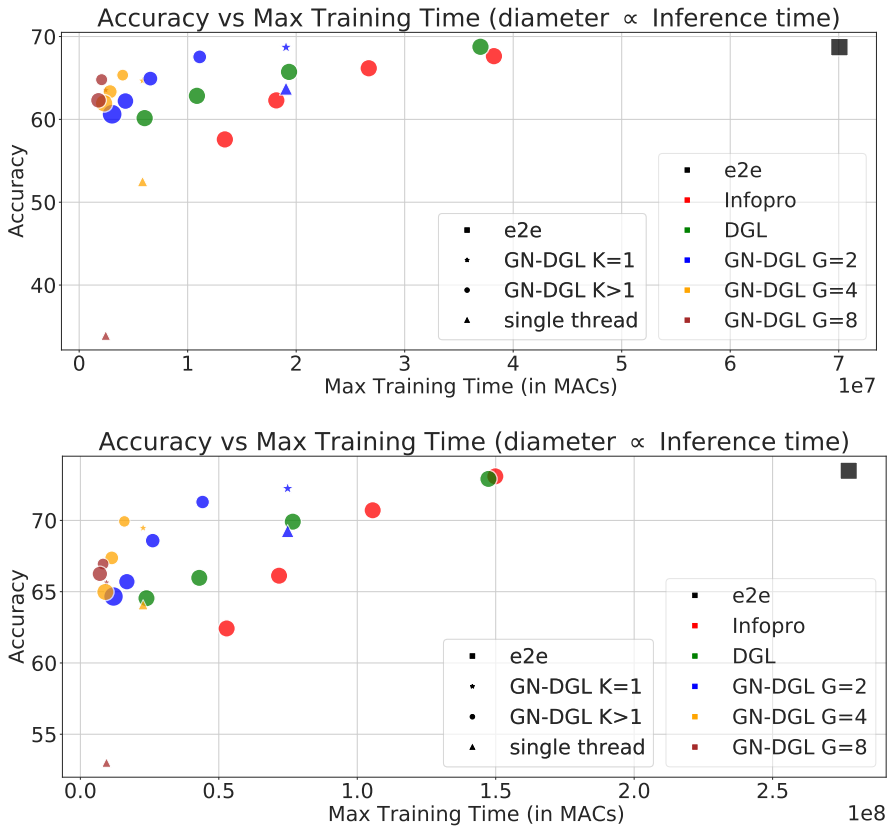


Figure 3.7: CIFAR-100 results. Accuracy versus maximum training time in MACs for a given node in a distributed scenario for various local learning methods. We observe that for both ResNet-32 (top) and a wider ResNet-32x2 (bottom), the GN-DGL leads to better tradeoffs in training time versus accuracy, with trends substantially improving over DGL and InfoPro.

Discussion of results For Multilayer GN-DGL, we conducted experiments on three datasets: CIFAR-10, CIFAR-100, and ImageNet-32 which are presented in Figures 3.5, 3.7, 3.6 respectively. We compared our method to DGL Belilovsky et al. (2020) and InfoPro Wang et al. (2021), two popular approaches for parallelizing deep neural network training, specifically CNNs. We used both ResNet-32 and a wider ResNet-32x2 for our experiments. In addition to accuracy, we also measured total training time and inference speed as metrics for evaluating the performance tradeoffs of each method as described in Sec 3.3.2. Additionally, we also present numerical tables corresponding to the results of the Figure 3.5 (ResNet-32 on CIFAR10) in Tables 3.4 & 3.5.

It can be observed that various configurations of our Grouped Neuron DGL method have better tradeoffs in many regimes in terms of performance, total training time and inference speed when

compared to DGL and InfoPro with ResNet-32. For example, our method achieves accuracy of 89.69% when K*G=4, compared to 88.62% of DGL with approx. 6 times training and 2 times inference speed up as shown in Tables 3.4 & 3.5. For the more complicated datasets CIFAR-100 and Imagenet32, we observe the tradeoffs are more significantly improved with the G>1 trending significantly higher than DGL or InfoPro. The naive baselines of single thread heavily underperforms in terms of training time and accuracy tradeoffs. We note that simple K=1 approach can perform reasonably well for wider models but degrades as G increases, particularly for CIFAR-100 dataset as shown in Figure 3.7.

InfoPro gives improved performance over DGL on CIFAR-10, however it does not provide substantial improvements for CIFAR-100. Furthermore, we were unable to obtain reasonable performance on Imagenet32 with the InfoPro method, either obtaining overly slow performance or low accuracy. We hypothesize that it requires a deep convolutional decoder model, which leads to drastic slowdown. We also observed it was more sensitive to hyperparameters than the DGL and GN-DGL method.

3.3.3 Ablations of Layerwise Ensembling

An ablation comparison between layerwise ensembling from Sec 3.3.2 is presented in Table 3.6. Here we compare the ensemble results to the naive result from just taking the last-layer. We observe that layerwise ensembling gives consistent improvement without any additional cost, validating our use of this approach in the experiments.

Evaluation Method	**G=2**				**G=4**			**G=8**	
	K=2	**K=4**	**K=8**	**K=16**	**K=2**	**K=4**	**K=8**	**K=2**	**K=4**
last-layer only	89.45	87.52	84.73	82.84	89.01	88.25	84.64	88.67	87.11
layerwise ensemble	89.68	87.94	85.37	83.38	89.27	88.11	85.09	88.96	87.86

Table 3.6: Ablating the effect of layerwise ensembling in GN-DGL on CIFAR-10 with ResNet-32. We observe that layerwise ensembling for GN-DGL yields better performance compared to last-layer evaluation. Performance gains are increased gradually as the K increases for a given G.

3.4 Conclusion and Future Work

In conclusion, our proposed GN-DGL method has shown promising results in outperforming existing local learning methods in certain scenarios. However, there is room for improvement in terms of the auxiliary network design to optimize the tradeoffs. As a future direction, efficient methods to promote diversity among neuron-groups could also be explored to further enhance the performance of the GN-DGL method. Additionally, the potential of the GN-DGL method can be evaluated in other complex tasks and architectures to determine its versatility and scalability.

Chapter 4

Decoupled Greedy Self-supervised Learning

4.1 Introduction

Self-supervised learning is a machine learning approach that involves training a model on a large dataset without providing explicit labels for each sample. Here, the model is given a task to perform, such as predicting a missing piece of information, and it is then left to learn from the data on its own. This approach has gained popularity in recent years because it allows for the efficient use of large amounts of unlabeled data, which is often easier to obtain than labeled data. Some examples of self supervised learning algorithms include SimCLR (T. Chen et al., 2020), which uses a contrastive loss function to learn visual representations from images, and GPT (Generative Pretrained Transformers) Brown et al. (2020), which is a autoregressive language model. It uses a transformer architecture to learn natural language processing tasks.

Given the widespread use of self supervised learning, it is natural to study how well DGL (Decoupled Greedy Learning) adapts to self-supervised learning tasks. DGL is a framework for learning in which the model architecture is divided in logical sub-components and trained in isolation using local loss functions. It has been used successfully on a variety of tasks in supervised learning such as large-scale image classification (Belilovsky et al., 2020), semantic segmentation and object detection (Wang et al., 2021). However, the performance of DGL on self supervised learning tasks

has not been extensively studied, and it is important to understand how well it adapts to this type of learning in order to fully leverage its capabilities and explore potential parallelization strategies. By studying the performance of DGL on self supervised learning tasks, we can better understand its capabilities and limitations, and potentially improve its performance in these scenarios. This would enable DGL to be more widely used in a variety of applications, including those that rely on self supervised learning techniques.

In this chapter, we will present a study on applying local-learning methods, such as DGL (Belilovsky et al., 2020), for the popular self-supervised learning method SimCLR (T. Chen et al., 2020). We will begin by examining the limitations of using DGL for SimCLR and then introduce a new local-learning method that builds on DGL, specifically tailored towards local learning of SimCLR. Our experiments demonstrates that the locally trained models using the proposed method are able to achieve the performance comparable to the end-to-end trained model.

4.2 Methods

In this section, we describe our methods studying local learning frameworks such as DGL for self-supervised learning tasks. First, we introduce the Decoupled Greedy Self-supervised Learning (DGSSL) method, which is described in detail in Sec 4.2.1. This method applies the SimCLR method, a popular self-supervised learning technique, within the DGL framework. We utilize linear probe evaluation method to understand the performance of each layer on the image classification task.

However, we also recognize the limitations of the DGSSL method and therefore introduce a new approach in Sec 4.2.2. This method, called DGSSL with Layerwise Loss Modification, modifies the local loss function for each layer (sub-module) such that the problem difficulty is gradually increased as the model progresses through the layers. This allows us to address the limitations of the original DGSSL method ultimately leading to improved performance on the self-supervised learning task.

4.2.1 Decoupled Greedy Self-supervised Learning (DGSSL)

To begin, we describe the local learning framework of DGL (Belilovsky et al., 2020) in the context of self-supervised learning and introduce notation. Consider a batch of samples x^k, first we

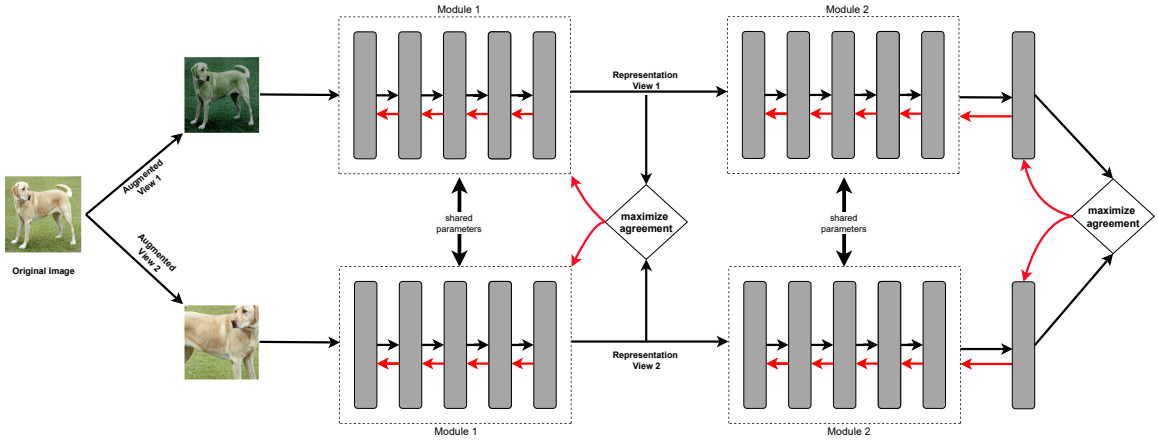


Figure 4.1: Decoupled Greedy Learning Framework for Self-supervised learning. The contrastive loss functions are applied locally on intermediate projections of feature representations using an auxiliary network. The main network modules along with their auxiliary modules share parameters to process two augmented views of samples. The error gradients only flow locally within corresponding modules.

apply data augmentation functions t and t' drawn from $\mathcal{T}$, where $\mathcal{T}$ is a distribution of stochastic data augmentations. Hence, we obtain a batch of transformed versions $\tilde{x}_0^{2k-1}$ and $\tilde{x}_0^{2k}$ by applying t and t' respectively on a batch of samples x^k as shown in Algorithm 3. Here, the sub-scripts and super-scripts indicate the local-module index and augmented-sample index respectively as shown in Algorithm 3.

Now, we denote operation of layer j as $f_{\theta_j}(x_{j-1})$, where θ_j corresponds to the parameters of the local module.We propose to learn the parameters θ_j in parallel online as described in Belilovsky et al. (2020).

DGSSL training

We apply a local-learning scheme of Belilovsky et al. (2020) to the contrastive learning framework of SimCLR (T. Chen et al., 2020) by splitting the encoder network depth-wise into multiple modules. These modules are trained in isolation from each other, along with their auxiliary modules, using local-loss functions. We note that the encoder network, represented by $f(\cdot)$ and its parameters θ, shown in Figure 2.3, is split into several local-modules, represented by $f_j(\cdot)$ with parameters θ_j, where j denotes a local-module. We understand the projection heads of SimCLR as the auxiliary networks. Therefore, in our Decoupled Greedy SSL approach, we have multiple auxiliary projection

Algorithm 3: Decoupled Greedy Self-supervised Learning (Training)

Input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
Initialize Parameters $\{\theta_j, \gamma_j$ for $f_j(\cdot), g_j(\cdot)\}_{j \leq J}$.
for *sampled minibatch* $\{\boldsymbol{x}^k\}_{k=1}^N$ **do**
  # prepare input-data
  for $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$ $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_0^{2k-1} = t(\boldsymbol{x}^k)$
    # the second augmentation
    $\tilde{\boldsymbol{x}}_0^{2k} = t'(\boldsymbol{x}^k)$
  end
  # iterate over J local-modules
  for $j \in 1, ..., J$ **do**
    # Forward propagate k sample-pairs in encoder-module $f_j(\cdot)$ and auxiliary-net $g_j(\cdot)$.
    for $k \in \{1, \ldots, N\}$ **do**
      $\boldsymbol{x}_j^{2k-1} = f_{\theta_j}(\tilde{\boldsymbol{x}}_{j-1}^{2k-1})$ # pre-auxiliary representation
      $\boldsymbol{z}_j^{2k-1} = g_{\gamma_j}(\boldsymbol{x}_j^{2k-1})$ # auxiliary projection
      $\boldsymbol{x}_j^{2k} = f_{\theta_j}(\tilde{\boldsymbol{x}}_{j-1}^{2k})$ # pre-auxiliary representation
      $\boldsymbol{z}_j^{2k} = g_{\gamma_j}(\boldsymbol{x}_j^{2k})$ # auxiliary projection
    end
    # calculate pairwise-similarity on auxiliary projections from module j
    for $a \in \{1, \ldots, 2N\}$ *and* $b \in \{1, \ldots, 2N\}$ **do**
      $S_j(a,b) = (z_j^a)^\top \boldsymbol{z}_j^b / (\|\boldsymbol{z}_j^a\| \|\boldsymbol{z}_j^b\|)$
    end
    define $\hat{\mathcal{L}}$ **as** $\hat{\mathcal{L}} = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$,
    where $\ell(a,b) = -\log \frac{\exp(S_j(a,b)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq a]} \exp(S_j(a,k)/\tau)}$
    # calculate parameter-gradients for encoder-module $f_j(\cdot)$ and auxiliary-net $g_j(\cdot)$.
    Compute $\nabla_{(\gamma_j, \theta_j)} \hat{\mathcal{L}}(S_j; \gamma_j, \theta_j)$.
    $(\theta_j, \gamma_j) \leftarrow$ Update parameters (θ_j, γ_j).
  end
end
return encoder network $\{f_j(\cdot)\}_{j \leq J}$, and throw away auxiliary networks $\{g_j(\cdot)\}_{j \leq J}$

heads, represented by $g_j(\cdot)$, and their parameters are γ_j.

As described in Algorithm 3, during training of Decoupled Greedy SSL, we obtain two augmented versions of input by applying series of stochastic augmentations on input batch x^k. Now, we train each local module in-parallel online using contrastive loss on intermediate projections z_j of these two different views of samples. Note that intermediate projection here refers to the output of auxiliary projection heads $g_j(\cdot)$ for module j.

Here, both transformed versions of samples are propagated forward using a shared local encoder module $f_j(\cdot)$ as shown in Figure 4.1. Then, at each local-module boundary, we have an auxiliary network which performs non-linear transformations on feature representations of both augmented

views, essentially performing similar operations as SimCLR projection heads. The auxiliary module parameters γ_j are also shared among two augmented views of intermediate representations. As mentioned earlier, in SimCLR, the contrastive loss is evaluated on the outputs of the projection heads to maximize the agreement between positive pairs of samples. Similarly, the local contrastive loss is evaluated on the outputs of auxiliary projection heads and it propagates the error signal back to corresponding local encoder module as shown in Figure 4.1. Simultaneously, gradient decoupled feature representations are propagated forward to train the subsequent local encoder modules. The last encoder module is directly coupled with a fully connected layer similar to SimCLR projection head. Hence, it can be seen as an auxiliary network for last encoder module. We note that local loss functions used to train the encoder modules are similar to the contrastive loss functions used by original SimCLR method (T. Chen et al., 2020).

Linear Evaluation of Decoupled Greedy SSL

In Decoupled Greedy SSL, during linear evaluation, we discard all auxiliary modules along with the projection heads of each local-modules. To evaluate a categorical cross-entropy loss, we attach an MLP module which performs a linear or non-linear transformation of representations x_j, where j indicates the last encoder module. As described in Algorithm 4, Θ_l are the parameters of the attached MLP module and similar to SimCLR, the parameters θ_j are only utilized during forward propagation and not updated during linear evaluation training. Hence, each gradient update step is only responsible to update the parameters Θ_l of the MLP head attached to the encoder module as shown in Algorithm 4.

4.2.2 Decoupled Greedy SSL with Layerwise Loss Modification

As described by Wang et al. (2021), in fully layerwise training, where each layer of the network is trained in isolation along with corresponding auxiliary networks using local loss function, the initial layers of the network are not able to pass enough task-relevant information about the input to the subsequent layers, which results in performance degradation. They show that when the layers are trained in isolation, the learned features tend to discard the information about the input which is not useful for that particular layer but may be useful in improving overall performance of the

Algorithm 4: Decoupled Greedy Self-supervised Learning (Linear Evaluation)

Input: batch size N, structure of f, h.
Initialize Parameters Θ_l for $h(\cdot)$, $\{\theta_j$ for $f_j(\cdot)\}_{j \leq J}$
for *sampled minibatch* $\{\boldsymbol{x}^k, y^k\}_{k=1}^N$ **do**
  for $k \in \{1, \ldots, N\}$ **do**
    for $j \in 1, ..., J$ **do**
      $\boldsymbol{x}_j^k = f_{\theta_j}(\boldsymbol{x}_{j-1}^k)$ # intermediate representation
    end
    $\hat{y}^k = h_{\Theta_l}(\boldsymbol{x}_j^k)$ # class logits
  end
  define $\hat{\mathcal{L}} = \frac{1}{N} \sum_{k=1}^N$ cross-entropy $(\hat{y}^k, y^k)$,
  Compute $\nabla_{\Theta_l} \hat{\mathcal{L}}(\hat{y}, y; \Theta_l)$.
  $\Theta_l \leftarrow$Update parameters Θ_l.
end

network. As a solution to this issue, they propose to reconstruct the input at each layer from the raw representations using the auxiliary decoder network. The problem with this decoder network is that, it adds a computational burden on the overall training process because it becomes very expensive to reconstruct the original input from high dimensional representations during later layers of the network. Hence, this approach to address the issue of task-relevant information collapse is not scalable. Hence, we introduce a new method to address this issue in the next section.

Proposed Method

We propose to address the information collapse issue by choosing different samples to train each layer of the network based on the sample difficulty. In other words, we increase the problem difficulty gradually as we go deeper in the network by explicitly choosing the samples to be trained at each layer. Our hypothesis is that it will help initial layers to propagate task-relevant information to subsequent layers if initial layers are trained with most easy samples with increasing difficulty as we go deeper in the network.

There are multiple ways we could choose to decide the difficulty of the samples given the task. As this work is focused on SimCLR framework, the sample difficulty is chosen based on how similar or different augmented versions are from each other. This can be decided based on heuristics which describes the severity of the augmentations applied. In other words, the use of severe augmentations can lead to significant changes in the appearance of a sample, while still maintaining its underlying meaning. This can make it challenging for a model to maximize agreement between these different,

but semantically similar, versions of the sample. An alternative method for determining sample difficulty is to utilize the pretrained SimCLR model. This model is able to distinguish between semantically similar and different images by producing corresponding representations in the feature space. For example, it will generate similar representations for a pair of semantically similar images, but distinct representations for semantically different images. By utilizing the output of the pretrained SimCLR model, we can effectively gauge the difficulty of a given sample as shown in Figure 4.2. This method is based on the assumption that the pretrained SimCLR model has learned robust features and can accurately capture semantic differences (and similarities) in the images being compared. To use this method, we compute the cosine similarity of the features produced by the SimCLR model for pairs of augmented samples before training. We utilize an end-to-end trained model using the SimCLR method to calculate these similarity scores.

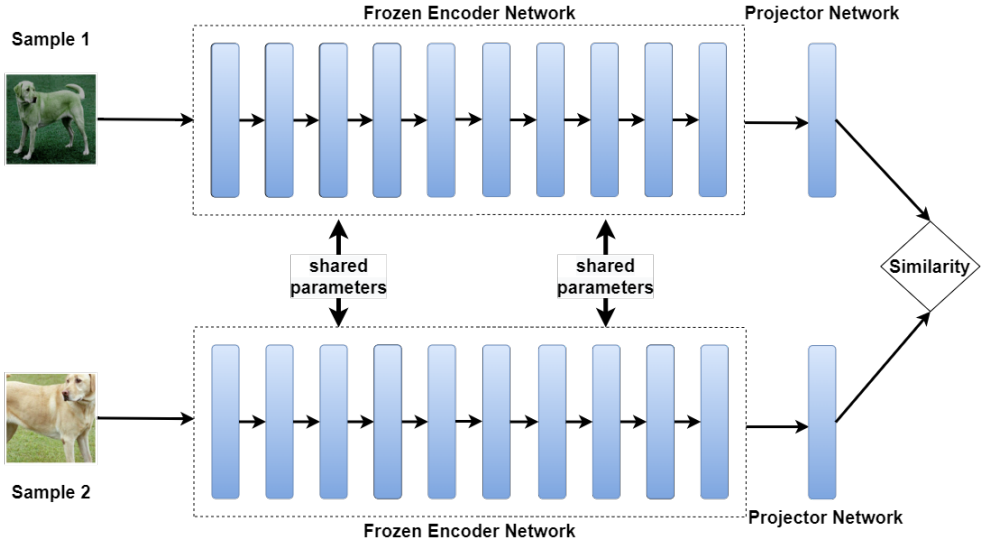


Figure 4.2: Sample-difficulty quantification: two augmented images are processed by the pretrained SimCLR model to obtain the embeddings to compute the cosine similarity. A positive pair with higher similarity score is considered easier than the one with lower similarity score. On the other hand, for negative pairs, lower similarity scores indicate an easy negative pair and vice-versa.

Difficulty thresholds In the SimCLR method, as described by T. Chen et al. (2020), two augmented versions of the same image are considered a positive pair, while negative pairs are created using augmented versions of other images in the same batch. However, in our DGSSL with Layerwise

Algorithm 5: DGSSL with Layerwise Loss Modification (DGSSL-LLM)

Input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.

Initialize Parameters $\{\theta_j, \gamma_j$ for $f_j(\cdot), g_j(\cdot)\}_{j \leq J}$, Thresholds $\{\alpha_j, \beta_j\}_{j \leq J}$.

for *sampled minibatch* $\{\boldsymbol{x}^k\}_{k=1}^N$ **do**

  # prepare input-data

  for $k \in \{1, \ldots, N\}$ **do**

    draw two augmentation functions $t \sim \mathcal{T}$ $t' \sim \mathcal{T}$

    # the first augmentation

    $\tilde{\boldsymbol{x}}_0^{2k-1} = t(\boldsymbol{x}^k)$

    # the second augmentation

    $\tilde{\boldsymbol{x}}_0^{2k} = t'(\boldsymbol{x}^k)$

  end

  # iterate over J local-modules

  for $j \in 1, ..., J$ **do**

    # Forward propagate k sample-pairs in encoder-module $f_j(\cdot)$ and auxiliary-net $g_j(\cdot)$.

    for $k \in \{1, \ldots, N\}$ **do**

      $\boldsymbol{x}_j^{2k-1} = f_{\theta_j}(\tilde{\boldsymbol{x}}_{j-1}^{2k-1})$ # pre-auxiliary representation

      $\boldsymbol{z}_j^{2k-1} = g_{\gamma_j}(\boldsymbol{x}_j^{2k-1})$ # auxiliary projection

      $\boldsymbol{x}_j^{2k} = f_{\theta_j}(\tilde{\boldsymbol{x}}_{j-1}^{2k})$ # pre-auxiliary representation

      $\boldsymbol{z}_j^{2k} = g_{\gamma_j}(\boldsymbol{x}_j^{2k})$ # auxiliary projection

    end

    # calculate pairwise-similarity on auxiliary projections from module j

    for $a \in \{1, \ldots, 2N\}$ *and* $b \in \{1, \ldots, 2N\}$ **do**

      $S_j(a, b) = (z_j^a)^\top \boldsymbol{z}_j^b / (\|\boldsymbol{z}_j^a\| \|\boldsymbol{z}_j^b\|)$

    end

    # prepare sets to modify the local-loss-function for module j

    $\mu =$ positive-sample-selection(S_j, N, α_j)

    # Ω is a list of sets with length $2N$

    $\Omega =$ negative-sample-selection(S_j, N, β_j)

    # calculate loss only for positive pairs present in μ

    define $\hat{\mathcal{L}}$ **as** $\hat{\mathcal{L}} = \frac{1}{2|\mu|} \sum_{k=1}^N \mathbb{1}_{[k \in \mu]} \left[\ell(2k-1, 2k, \Omega_{2k-1}) + \ell(2k, 2k-1, \Omega_{2k})\right]$,

    # ignore the negative pairs not present in ω

    where $\ell(a, b, \omega) = -\log \frac{\exp(S_j(a,b)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq a \wedge k \in \omega]} \exp(S_j(a,k)/\tau)}$

    # calculate parameter-gradients for encoder-module $f_j(\cdot)$ and auxiliary-net $g_j(\cdot)$.

    Compute $\nabla_{(\gamma_j, \theta_j)} \hat{\mathcal{L}}(S_j; \gamma_j, \theta_j)$.

    $(\theta_j, \gamma_j) \leftarrow$ Update parameters (θ_j, γ_j).

  end

end

return encoder network $\{f_j(\cdot)\}_{j \leq J}$, and throw away auxiliary networks $\{g_j(\cdot)\}_{j \leq J}$

Loss Modification method, we must carefully select positive and negative pairs for each layer or submodule based on the thresholds α_j and β_j. These thresholds determine which positive pairs (and negative pairs) are used for training at layer j. Specifically, positive pairs with a similarity score above α_j and negative pairs with a similarity score below β_j are chosen, with the values of α_j and β_j decreasing and increasing respectively. This process is designed to train the initial layers with

the easiest positive and negative pairs (indicated by higher and lower similarity scores, respectively) and gradually adding in more difficult pairs as the training progresses. The final layer or submodule will be trained on all positive and negative pairs in the batch, as it will have the lowest value for $\alpha_j(0.0)$ and the highest value for $\beta_j(1.0)$, representing the full difficulty of the problem. Note that the thresholds α_j and β_j are initialized separately for each layer/module j.

Algorithm 6: Positive Sample Selection

Input: Pairwise similarity matrix S, batch size N, Difficulty threshold for positive pairs α.

$\mu \leftarrow \{\}$
for $k \in \{1, \ldots, N\}$ **do**
  if $S(2k, 2k-1) \geq \alpha$ **then**
    $\mu \leftarrow \mu \cup \{k\}$
  end
end
return μ

Algorithm 7: Negative Sample Selection

Input: Pairwise similarity matrix S, batch size N, Difficulty threshold for negative pairs β.

1 $\forall i \in \{1, 2, \ldots, 2N\}, \omega_i \leftarrow \{\}$
2 $\Omega = [\omega_1, \omega_2, \ldots, \omega_{2N}]$ such that $\Omega_i \equiv \omega_i$
3 **for** $a \in \{1, \ldots, N\}$ **do**
4 **for** $k \in \{1, \ldots, 2N\}$ **do**
5 **if** $S(2a-1, k) \leq \beta$ **then**
     $\Omega_{2a-1} \leftarrow \Omega_{2a-1} \cup \{k\}$
6 **end**
7 **if** $S(2a, k) \leq \beta$ **then**
     $\Omega_{2a} \leftarrow \Omega_{2a} \cup \{k\}$
8 **end**
9 **end**
10 **end**
11 return Ω

Sample selection For selecting the positive samples, we simply iterate over the similarity matrix S_j of layer j, and select the positive pair (a, b) for which the pairwise similarity $S_j(a, b) \geq \alpha_j$ as shown in 6. Similarly, the negative samples are selected individually for each selected positive sample as shown in lines 5 and 7 of the Algorithm 7. The positive and negative sample pairs are selected into μ

and Ω respectively. Here, Ω is a list of sets ω_i where $i \in \{1, 2, \ldots, 2N\}$, and it indicates the sample index for which the negative samples are selected in set ω_i. Subsequently, the contrastive local loss function is modified based on the selected pairs of samples in sets μ and ω_i. It is important to note that the contrastive loss $\ell(a, b)$ is only calculated if the negative sample b is present in the set ω_i, where i is the index of sample a. The steps for calculating the contrastive local loss based on sets μ and ω is outlined in Algorithm 5.

Linear evaluation The evaluation method for DGSSL with Layerwise Loss Modification is same as vanilla DGSSL as described in the Algorithm 4.

4.3 Experiments and Results

In this work, we conducted experiments on two widely used deep learning architectures, VGGNet and ResNet, using the CIFAR-10 dataset. Our primary focus is on evaluating the performance of the Decoupled Greedy SSL method and understand the limitations.

To obtain the intuition behind the method, we design first set of experiments based on VGGNet, which follows training each layer of VGGNet in isolation using a local contrastive loss function. In addition, we also explore the use of this method as an application to model parallel training using ResNets. In experiments based on ResNet architecture, we use a local module that consists of multiple layers which are trained with the help of local contrastive loss function as described in 31. The contrastive loss function is similar to NT-Xent loss described in T. Chen et al. (2020).

The limitations of the vanilla DGSSL method has been studied using ResNet50 architecture, which led to the design of next set of experiments focused on studying and evaluating the second method (DGSSL with Layerwise loss modification) presented in Sec 4.3.3. For evaluation of all experiments, we follow a linear evaluation method after unsupervised training has completed as described at the end of Sec 4.2.1.

Hyperparameters For all our experiments with VGGNet and ResNets, we use batch size of 256. We adapt Adam optimizer (Kingma & Ba, 2014) with learning rate of 0.001. To obtain different views of samples, we use four types of data augmentations applied to each sample in sequence as

described in Appendix A.2. We use random resized crop, horizontal flip, color-jitter, and grayscale during self-supervised training of the models. The ℓ_2 weight decay of 1e-6 is also used for all experiments.

4.3.1 Layerwise Decoupled Greedy SSL

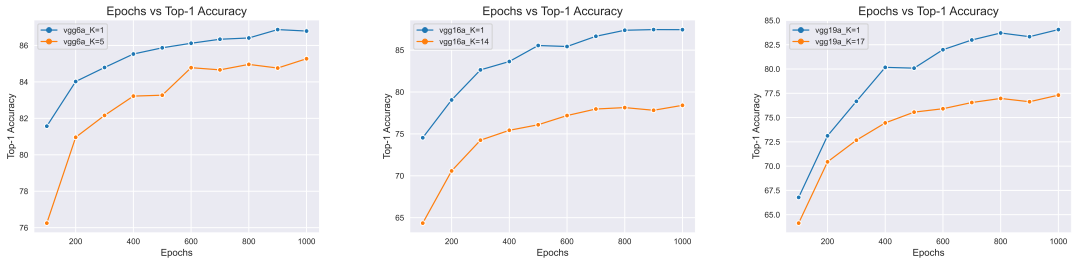


Figure 4.3: VGG6a (left), VGG16a (middle) and VGG19a (right): Comparison of linear evaluation top-1 accuracy for end-to-end self-supervised training ($K = 1$) and extremely local self-supervised training ($K = 5$ for VGG6a, $K = 14$ for VGG16a and $K = 17$ for VGG19a) using Decoupled Greedy Learning framework.

The VGGNet architectures used in this set of experiments are VGG6a, VGG16a and VGG19a. Here, we decouple each layer of the network and attach a separate auxiliary network for each layer. The local contrastive loss function is applied on the projections after the auxiliary network to train each layer and their auxiliary module in isolation. For VGGNet experiments, we use one hidden layer MLP as an auxiliary module which transforms the representations to a 128-dimension feature space where contrastive loss is applied. The results are shown in the Figure 4.3. We observe that VGG6a demonstrates better performance for extremely local training with $K = 5$, whereas deeper models such as VGG16a and VGG19a shows poor performance for parallelized models compared that of their end-to-end trained counterparts. Based on what Wang et al. (2021) concluded, we believe that the extremely local training of deeper networks suffer more from the issue of information collapse compared to models with less depth. This hypothesis can be verified further by more experiments. Also, more efficient auxiliary networks can be introduced to improve the overall performance.

4.3.2 Multilayer Decoupled Greedy SSL

Now, we extend the idea of layerwise training to multi-layer local modules as an application to model-parallel training. Here, instead of each layer being trained in isolation using a contrastive loss function, we form a local module with a set of layers and attach an auxiliary network at the end of last layer of each local module as shown in Figure 4.1. We demonstrate our results for ResNet-50 and ResNet-110 in Figures in Figure 4.4 and 4.5 respectively. We observe a trend of decrease in accuracy as the value of K increases, which is expected. Here, K refers to depth-wise splitting of the encoder module resulting in total K number of local-modules that are trained in parallel. Details about depth-wise splitting of the full encoder network is deferred in Appendix A.1.

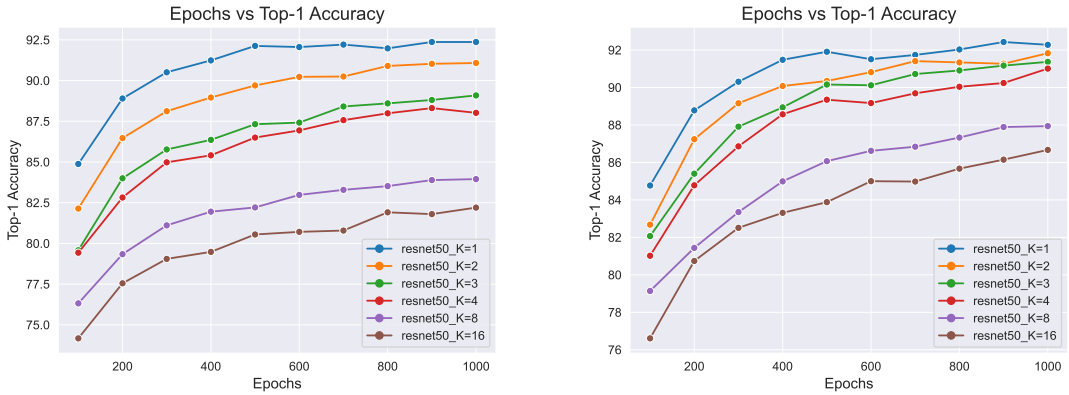


Figure 4.4: Decoupled Greedy SSL results for ResNet-50. Comparison of linear evaluation top-1 accuracy for different values of K, where $K = 1$ suggests end-to-end training. We observe that changing auxiliary network from '0c2f' (left) to '1c2f' (right) improves the performance significantly.

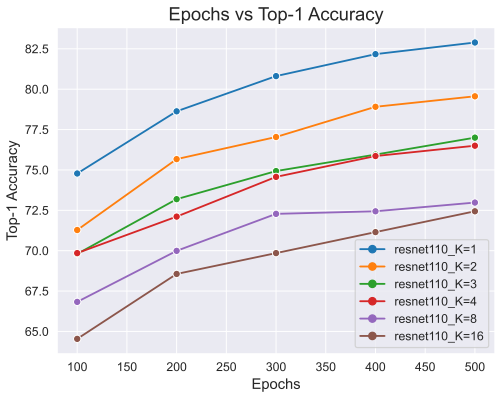


Figure 4.5: Results of Decoupled Greedy SSL for ResNet-110 with different values of K. We observe significant drop in performance as the value of K is increased. We believe this happens because of the increased depth of the network, which contributes towards information collapse as described in Wang et al. (2021). The auxiliary network design follows the '1c2f' architecture.

Auxiliary network design In our experiments with the ResNet models, we explored the impact of using different auxiliary network designs on the performance of parallelized models. Specifically, we compared a simpler design with just two hidden layer MLP and no convolutional layers (referred to as '0c2f') to a design with one convolutional layer and two hidden layer MLP (referred to as '1c2f'). Our results showed that the use of the '0c2f' design led to a decrease in performance as the degree of parallelization (represented by the value of K) increased, as shown in left of Figure 4.4. On the other hand, the '1c2f' design significantly improved the performance of the parallelized models compared to the '0c2f' design. We provide further evidence of this effect in Figures A.4 and A.5. Based on these results, we decided to only use auxiliary networks with the '1c2f' design in all of our subsequent experiments.

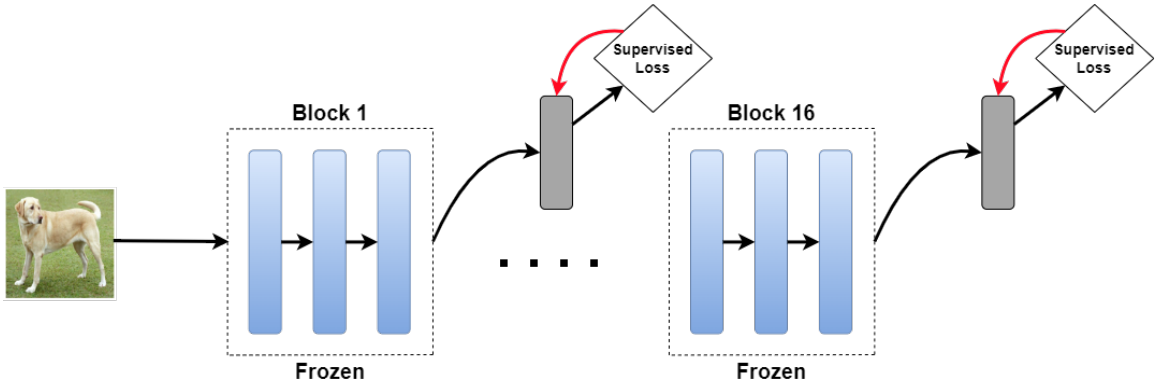


Figure 4.6: Depth-wise linear probe evaluation: intermediate representations from each block of ResNet-50 are evaluated using linear probes. The weights of the encoder blocks are frozen during linear probe training.

Depth-wise dynamics In order to better understand the performance of locally trained models, we conducted an evaluation over depth using a linear probe method on pretrained ResNet-50 models that were trained with various values of K. The evaluation involved attaching several MLP heads (linear probes) after each block of a pretrained ResNet-50 model as illustrated in Figure 4.6. Typically, a block is made up of three convolutional layers in ResNet-50 architecture. The specific boundaries are analogous to the local-module boundaries of K=16, which is specified in Figure A.1. These MLP heads were then trained on the CIFAR-10 image classification benchmark for 200 epochs, using the representations obtained at different depths of the main encoder network. We note that the parameters

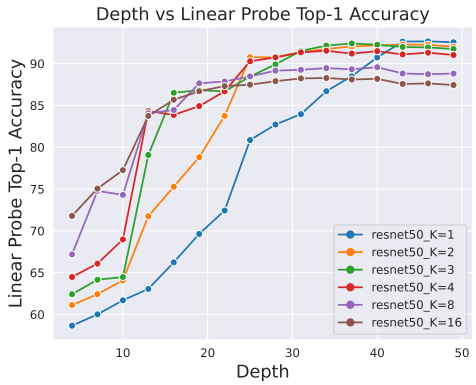


Figure 4.7: Depth vs linear probe top-1 accuracy for ResNet-50 with different values of K. We evaluate linear probe top-1 accuracy at different depths.

of the main encoder network are not updated during linear evaluation training process. This allowed us to compare the performance of the models with different values of K and understand how the learning dynamics differed between them.

The results are shown in Figure 4.7. We see an interesting trend of increase in accuracy over depth for different values of K. We observe that for the end-to-end training, increase in linear probe accuracy is more gradual as the depth increases. However, the DGSSL method appears to plateau early on in terms of accuracy.

One possible explanation for this difference is the information-collapse phenomenon described by Wang et al. (2021). In the DGSSL method, the early layers may be trying to fully solve the problem, leading to a collapse of information that prevents the model from performing as well as the end-to-end trained model. On the other hand, the end-to-end trained model is able to gradually solve the problem and achieve better overall performance, despite the initial layers not performing as well.

This effect is consistent as the value of K increases. For larger values of K, the initial layers tend to perform very well, as can be seen in the case of $K = 16$. However, as we move deeper into the model, the trend seems to change and the later layers perform worse on the task, resulting in lower overall performance. To address this problem we propose a new Structured DGL scheme (referred to as DGSSL-LLM) that modifies the loss function layerwise in order to gradually increase the problem difficulty, described in Sec 4.2.2.

4.3.3 Decoupled Greedy SSL with Layerwise Loss Modification

In this section, we present experiments to evaluate the DGSSL with Layerwise Loss Modification (DGSSL-LLM) method and compare it's performance with vanilla DGSSL. The results are presented in Table 4.1. We consider the end-to-end model a special case of DGSSL with K=1 for the purpose of comparison. We observe that for all values of K our DGSSL-LLM method outperforms vanilla DGSSL, nearly recovering end-to-end performance for K=4. Specifically, as the value of K increases the gain in the performance is larger. We also investigate the learning dynamics of the DGSSL-LLM method by applying linear probe evaluation method at different depths of the networks. As shown in the Figure 4.8, the overall performance of the model is increased but the evaluation over depth suggests that the initial layers are still trying to fully solve the task as in vanilla DGSSL.

Method	K=1	K=4	K=8	K=16
DGSSL	92.56	91.03	88.81	87.44
DGSSL-LLM		**91.96**	**90.51**	**89.8**

Table 4.1: Comparison of DGSSL-LLM vs vanilla DGSSL for different values of K. Linear evaluation accuracy evaluated at the last layer of the models is presented. DGSSL-LLM outperforms vanilla DGSSL in all cases, nearly recovering end-to-end performance for $K = 4$.

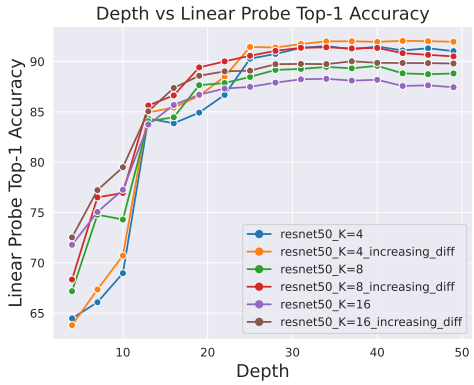


Figure 4.8: Depth vs linear probe top-1 accuracy. We evaluate linear probe top-1 accuracy at different depths. Evaluation of DGSSL-LLM (indicated by increased difficulty) vs vanilla DGSSL for different values of K.

Choosing difficulty thresholds As discussed in Sec 4.2.2, we propose to modify the local loss function in order to increase the difficulty of the task with depth. To achieve this, we must subselect

positive and negative sample pairs that represent the problem-difficulty using difficulty thresholds. These thresholds, α_j and β_j, are used to determine the selection of positive and negative pairs of samples based on the similarity scores obtained from a previously trained SimCLR model. By adjusting these thresholds, we can control the level of difficulty at each layer/module. We must carefully choose these difficulty thresholds for each layer j in the DGSSL-LLM method before training. If we choose very high or very low values for α_j and β_j respectively, there is a risk of only using a small subset of pairs to train the layer. This can lead to poor generalization at that particular layer due to overfitting on a small subset of data and ultimately negatively impacting the overall performance of the model.

To address this issue, we propose new hyperparameters, positive-sample-ratio (μ_{pos}) and negative-sample-ratio (μ_{neg}), as a way to select difficulty thresholds for our model. Rather than selecting these thresholds for each layer separately, we choose the sample ratios universally for the entire model, which change non-linearly for each layer or module. We use the sample ratios to determine the proportion of positive or negative pairs we select for any layer/module. For example, if we set a positive-sample-ratio of 0.50, we would choose at least half of the total number of positive sample pairs to train that particular layer/module. We define the positive and negative sample ratios for each layer/module j as follows:

$$\mu_{pos}^{j} = \mu_{pos}^{(K-j)}$$

$$\mu_{neg}^{j} = \mu_{neg}^{(K-j)}$$

Here, K denotes the total number of local-modules and j is the module-index ranging from 1 to K. In other words, if we have K=16, the first local module would have sample ratios of $\mu_{pos}^{(16-1)} = \mu_{pos}^{15}$ and $\mu_{neg}^{(16-1)} = \mu_{neg}^{15}$, and the last layer/module would always have sample ratios of 1, meaning it is trained with all the available positive and negative sample pairs, representing the highest level of problem difficulty. By using these sample ratios, we can search for difficulty thresholds that will help maintain the desired size of the subset as specified by the sample ratio. This approach allows for a more consistent and efficient selection of samples.

Sensitivity analysis We performed a sensitivity test to evaluate the effect of using different sample-ratios on our DGSSL-LLM method. The results of this test are presented in Table 4.2. The aim of this test was to investigate how sensitive our method is to variations in the sample-ratios. We have noticed that our method is highly resilient to a variety of sample-ratios. This means that regardless of the proportion of samples used, our method outperforms vanilla DGSSL in some cases or achieves comparable performance. This highlights the robustness and flexibility of our method.

Method	Sample-ratio	K=4	K=8	K=16
Vanilla DGSSL	-	91.03	88.81	87.44
DGSSL-LLM	0.81	89.9	88.12	86.63
	0.83	91.08	89.8	87.51
	0.85	90.55	88.75	87.7
	0.87	90.38	89.35	88.21
	0.89	91.43	90.34	88.44
	0.91	90.93	89.77	88.24
	0.93	91.26	89.92	88.93
	0.95	**91.96**	90.37	**89.8**
	0.97	91.54	**90.51**	89.66

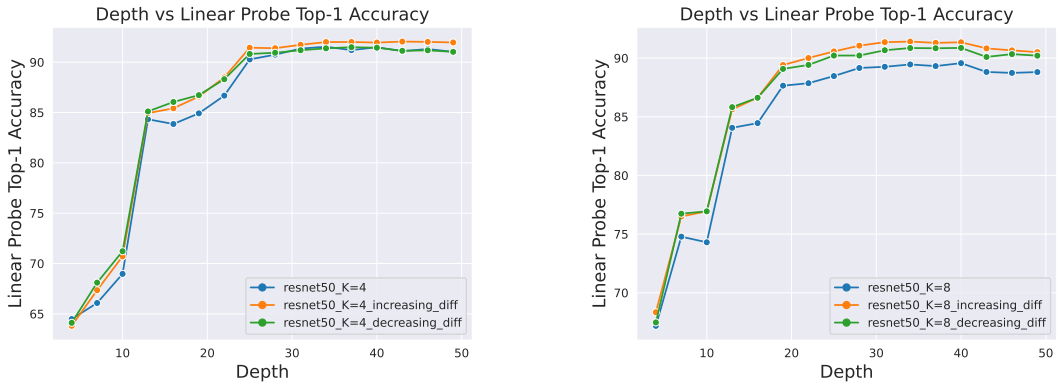
Table 4.2: Hyper-parameter sensitivity test: Effect of using different sample-ratios on DGSSL-LLM with different values of K. We observe that DGSSL-LLM with diverse sample-ratios achieve comparable performance to vanilla-DGSSL. The positive and negative sample-ratio is kept same for this test. Best results are in **bold**.

Ablating the effect of decreasing problem difficulty In this ablation study, we tested a variation of the DGSSL-LLM method that decreases the problem difficulty with depth, instead of increasing it as in the original method. The results, presented in Table 4.3, showed that both versions of DGSSL-LLM outperformed the vanilla DGSSL. However, the version that increased the difficulty with depth achieved the highest performance for all values of K. We believe that allowing each layer/module to operate at different difficulty levels improves the performance of the vanilla DGSSL method, but gradually increasing the difficulty is more effective. Furthermore, the depth-wise evaluation of both the versions of DGSSL-LLM along with vanilla DGSSL is presented in Figure 4.9.

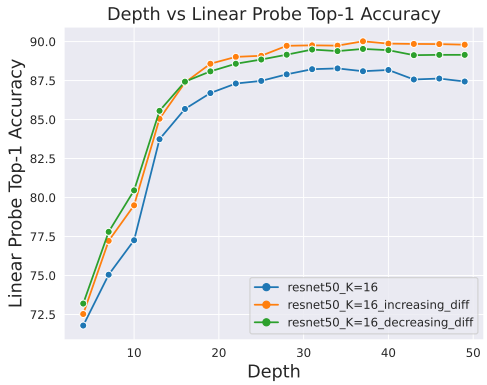
Ablating the effect of negative-pair selection The SimCLR method operates on the contrastive loss function, where the features of the model are learned by maximizing the agreement between

Method	K=1	K=4	K=8	K=16
DGSSL	92.56	91.03	88.81	87.44
DGSSL-LLM (increasing-diff.)		**91.96**	**90.51**	**89.8**
DGSSL-LLM (decreasing-diff.)		91.34	90.21	89.15

Table 4.3: Ablating the effect of decreasing problem-difficulty as opposed to increasing problem-difficulty in DGSSL-LLM method for different values of K. DGSSL-LLM with increasing difficulty achieves **best** results across all values of K.



(a) DGSSL-LLM with increasing vs decreasing problem-difficulty for K=4 (left) and K=8 (right).



(c) DGSSL-LLM with increasing vs decreasing problem-difficulty for K=16.

Figure 4.9: Linear Probe Evaluation over depth for different version of DGSSL-LLM and vanilla DGSSL. We observe DGSSL-LLM with increasing difficulty outperforms other methods for all values of K.

the positive pairs of samples and pushing the negative pairs of samples away from each other in the feature space. The DGSSL-LLM method described in Sec 4.2.2, modifies this contrastive local loss

function to increase the difficulty of the problem as we move deeper into the layers.

To achieve this, we subselect the positive and negative pairs of samples at each layer. This way, the initial layers work with easy pairs of positive and negative samples, while the deeper layers work with more challenging pairs. In this ablation study, we only modify the layerwise-local-loss to subselect the positive pairs of samples and keep the negative pairs of samples as they are. This allows us to maintain the desired level of difficulty throughout the layers. The results of this study are presented in Table 4.4. We found that both versions of DGSSL-LLM outperformed the vanilla DGSSL. However, the version of DGSSL-LLM that utilized both positive and negative pairs achieved the highest performance for K=16. This suggests that utilizing both positive and negative pairs in the DGSSL-LLM method improves the overall performance of the model.

Method	DGSSL	DGSSL-LLM (Pos.+Neg.)	DGSSL-LLM (Pos. Only)
K=16	87.44	**89.8**	88.66

Table 4.4: Ablating the effect of only using positive sample pairs as opposed to using both positive and negative sample pairs for layerwise loss modification in DGSSL-LLM with K=16.

4.4 Conclusion and Future Work

In conclusion, our study has highlighted the limitations of the existing local learning method, DGL, for a popular self-supervised learning algorithm, SimCLR. To address these limitations, we proposed a new method called Decoupled Greedy Self-supervised Learning with Layerwise Loss Modification (DGSSL-LLM) that significantly improves the performance of locally trained models. Our ablation studies showed that the use of increasing difficulty applied on both positive and negative pairs is the most effective variant of DGSSL-LLM. This finding is crucial in the development of local learning methods for self-supervised learning algorithms. In future work, we plan to extend our framework to other self-supervised learning methods such as Barlow-twins and Masked Autoencoders (MAE) to further validate the effectiveness of the approach.

Chapter 5

Conclusion and Future Work

In this thesis, we presented two methods to scale the current local learning methods in supervised learning and self-supervised learning settings, specifically focusing on image-classification task.

Scaling local learning for supervised learning We proposed a new method for training neural networks called the Grouped Neuron DGL (GN-DGL). This method builds on existing local learning methods such as DGL, and is based on the idea of using a purely local loss function to train independent neuron groups where there is no feedback between the independent groups. By breaking down the objective function in this manner, the optimization process can be parallelized, potentially leading to faster training. Specifically we studied a common image classification task, focusing on the convolutional neural network. The results from experiments demonstrate that this approach can allow for increased parallelization of local learning methods, unlocking substantial improvements for accuracy given training and inference time. In fact, in some cases, the GN-DGL method even outperformed the traditional end-to-end training approach.

We also discussed the potential implications of this approach on the practical considerations for distributed training. The GN-DGL method allows for the different model components to simultaneously learn their model parameters, which can be beneficial for distributed training scenarios where different processing units can handle different components of the network. Additionally, we explored ways to encourage diverse behavior in these decoupled neuron-groups based solely on forward communication. This proved to be important because it helped to avoid correlated features

among groups and improved the performance of the GN-DGL method.

However, it is important to note that the proposed method should be further investigated in more complex tasks and different types of data sets, to fully understand its capabilities and limitations. The current method of promoting diverse feature learning between neuron-groups requires communicating representations to different auxiliary nets, which may not be the most efficient method. Therefore, future work should look into more efficient methods of propagating such group-specific contexts and improve the overall efficiency of the GN-DGL method. Additionally, more efficient design of auxiliary networks will also affect the performance of this method significantly.

Overall, the GN-DGL method provides a new way to train neural networks and can be a useful approach for researchers, practitioners, and engineers who are working with large-scale data sets, distributed systems or neural networks. The method has the potential to improve the efficiency of neural networks by reducing the training time, and speeding up the inference time. However, further research is needed to fully understand its capabilities and limitations.

Local learning for self-supervised learning We presented a detailed study on applying Decoupled Greedy Learning (DGL) framework, for the popular self-supervised learning method SimCLR. We evaluated the performance of locally trained models at different depths and discovered that using the DGL for SimCLR has a limitation. DGL tends to focus on one specific layer at a time instead of the entire model, causing the early layers to try and fully solve the task. This leads to the issue of information-collapse in early layers and ultimately degrades the overall performance of the network.

To address this limitation, we introduced a new local-learning method, called Decoupled Greedy Self Supervised Learning with Layerwise Loss Modification (DGSSL-LLM). This method modifies the local loss function for each layer to gradually increase the problem difficulty as the model progresses through the layers. We showed that this approach allows the initial layers to propagate the task-relevant information to subsequent layers and results in better performance, while still leveraging the benefits of the DGL framework such as increased parallelization and low-memory footprint. By experimental analysis, we demonstrated that that locally trained models using the proposed method are able to achieve performance comparable to the end-to-end trained model on the image classification task using CIFAR-10 dataset. The results also indicate that this method is more

robust to different architectures and different number of local modules.

However, the main limitation of the DGSSL-LLM method is the use of a pretrained SimCLR model (potentially end-to-end for robustness) to quantify the sample difficulty in DGSSL-LLM method. Future work will focus on replacing the use of such pretrained models by introducing other simple and efficient methods for difficulty quantification. Additionally, this study is limited to the specific task of image classification using the SimCLR method, and it would be interesting to further study the generalization of the proposed method to other self-supervised learning methods and tasks such as Masked Auto-Encoders (MAEs), Barlow-twins etc.

Overall, this method is important for understanding the capabilities and limitations of local learning approaches in self-supervised learning tasks. It could potentially improve the efficiency of the models by reducing the training time and lead to significant computational advantages, making this method an attractive option for large-scale applications.

Appendix A

Decoupled Greedy SSL

A.1 Local Module Splitting

In this section, we show how local modules are obtained from the main encoder module for different values of K for experiments presented in Sec 4.3. We divide the main encoder module depth-wise to obtain K number of modules and define local module boundaries. As shown in Figure A.1, each pair of numbers is a local module boundary for different values of K. The pair of numbers describe the layer index and index of basic/bottleneck block of resnet within that layer respectively. This setup for ResNet-110 is taken from Wang et al. (2021).

```
'resnet50': {
    1: [],  # End-to-end
    2: [[3, 0]],
    3: [[2, 1],
        [3, 3]],
    4: [[2, 0],
        [3, 0], [3, 4]],
    8: [[1, 1],
        [2, 0], [2, 2],
        [3, 0], [3, 2], [3, 4], [4, 0]],
    16: [[1, 0], [1, 1], [1, 2],
         [2, 0], [2, 1], [2, 2], [2, 3],
         [3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5],
         [4, 0], [4, 1]],
}
```

```
'resnet110': {
    1: [],  # End-to-end
    2: [[2, 8]],
    3: [[1, 17],
        [2, 17]],
    4: [[1, 11],
        [2, 7],
        [3, 3]],
    8: [[1, 4], [1, 11],
        [2, 0], [2, 7], [2, 14],
        [3, 3], [3, 10]],
    16: [[1, 1], [1, 4], [1, 7], [1, 10], [1, 13], [1, 16],
         [2, 1], [2, 4], [2, 7], [2, 11], [2, 15],
         [3, 1], [3, 5], [3, 9], [3, 13]],
},
```

Figure A.1: Depth-wise split configuration for ResNet-50 and ResNet-110.

A.2 Data Augmentation Module

In following Figure A.2, we have shown the data augmentation functions used for all experiments presented in Chapter 4.

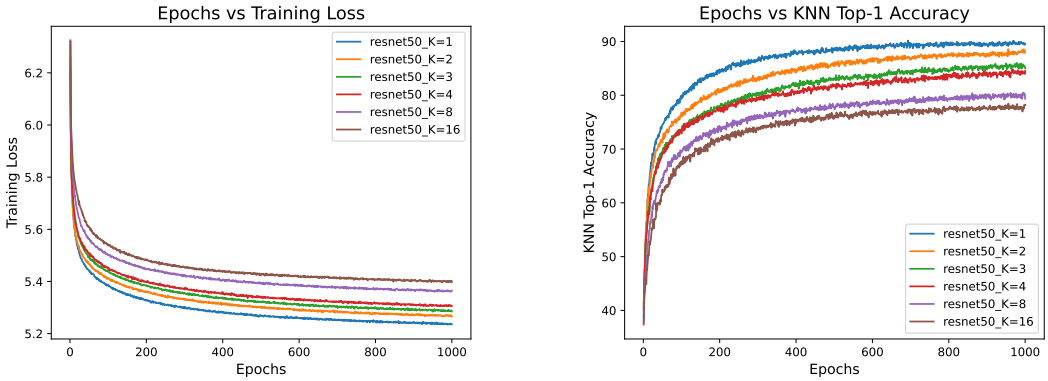
```
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(32),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomApply([transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010])])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010])])
```

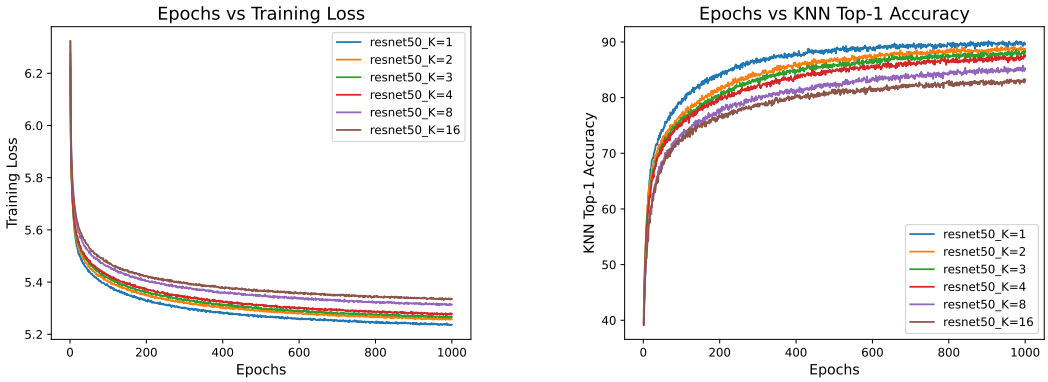
Figure A.2: Data augmentation functions used for the implementation of our experiments.

A.3 Additional Results

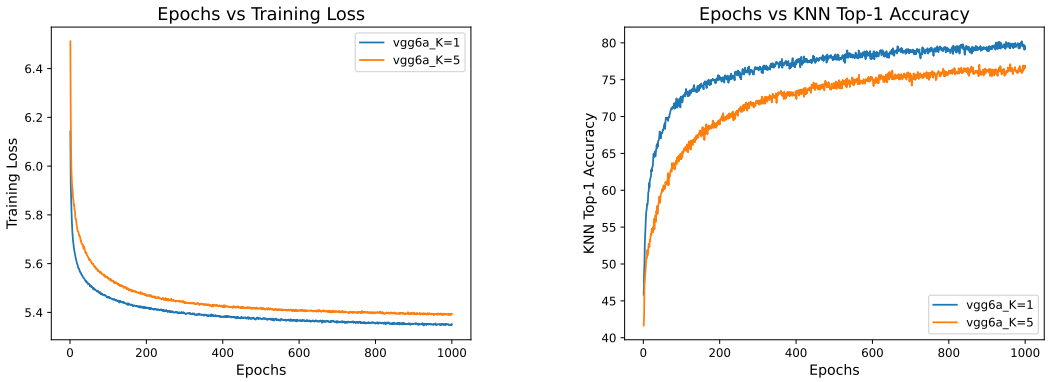
In this section, we present some additional results for our Decoupled Greedy SSL method presented in Chapter 4. The training and validation curves for experiments presented in Sec 4.3 are shown in Figure A.3. Furthermore, comparison between different auxiliary network architectures ('0c2f' and '1c2f') used in DGSSL method for ResNet-50 and ResNet-110 architectures is presented in Figures A.4 and A.5 respectively.



(a) ResNet-50 with '0c2f' auxiliary network for different values of K.

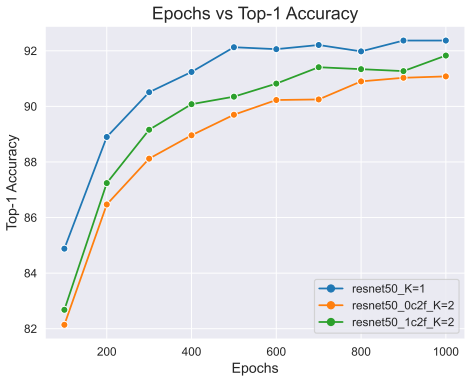


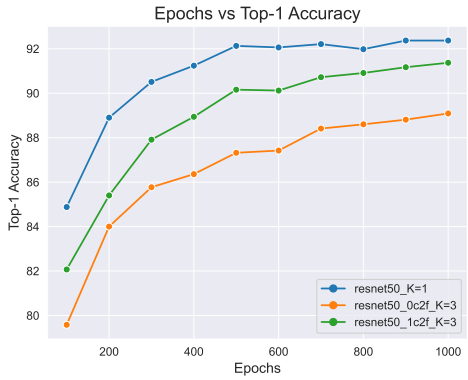
(b) ResNet-50 with '1c2f' auxiliary network for different values of K.



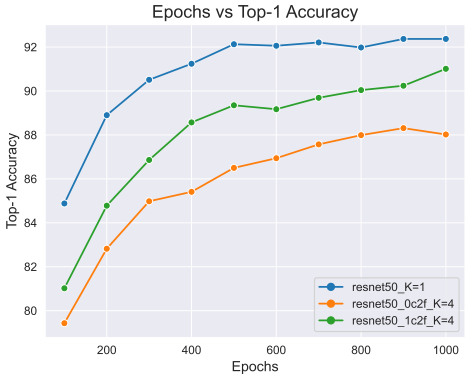
(c) VGG6a with '0c2f' auxiliary network for different values of K.

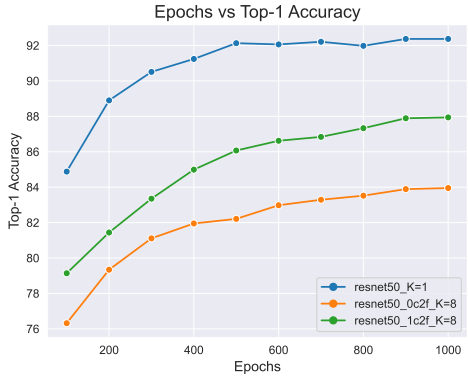
Figure A.3: Epochs vs Training Loss (left) and Epochs vs KNN Top-1 Accuracy (right), evaluated after each epoch for ResNet-50 with '0c2f' (a), ResNet-50 with '1c2f' (b), and VGG6a (c) for different values of K.



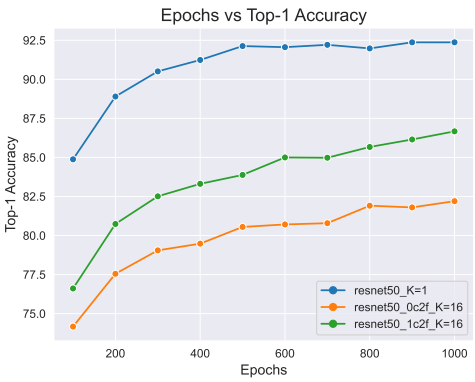


(a) ResNet-50 with different auxiliary networks for $K = 2$ (left) and $K = 3$ (right).



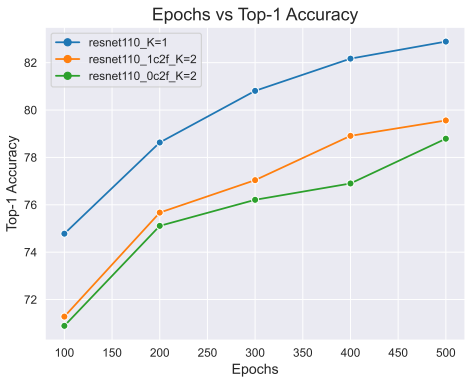


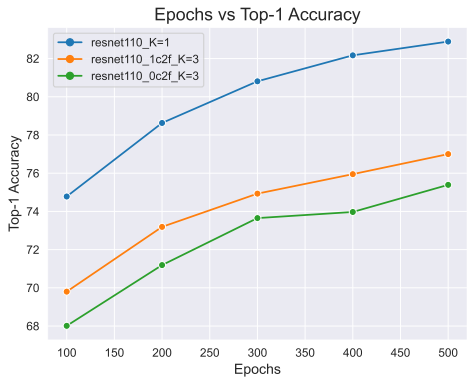
(b) ResNet-50 with different auxiliary networks for $K = 4$ (left) and $K = 8$ (right).



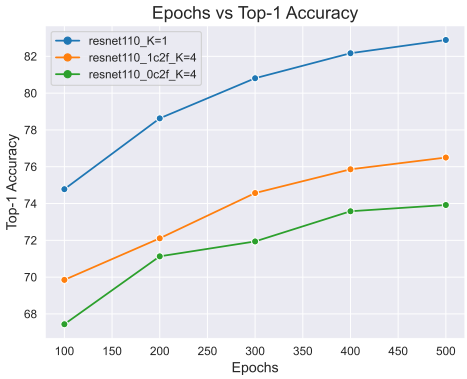
(c) ResNet-50 with different auxiliary networks for $K = 16$.

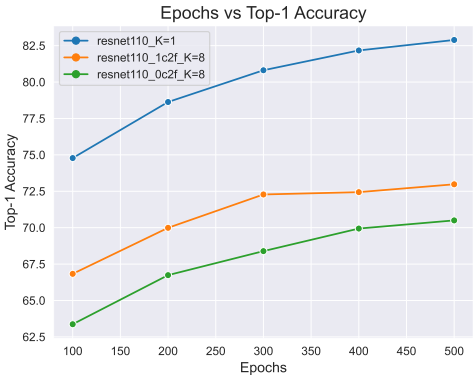
Figure A.4: Epochs vs Linear Evaluation Top-1 Accuracy, evaluated for ResNet-50 after every 100 epochs, trained using different auxiliary networks ('0c2f' and '1c2f') for different values of K.



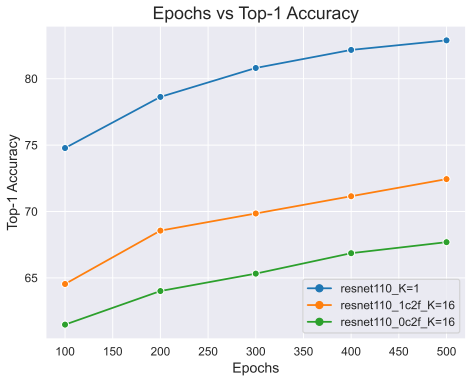


(a) ResNet-110 with different auxiliary networks for $K = 2$ (left) and $K = 3$ (right).





(b) ResNet-110 with different auxiliary networks for $K = 4$ (left) and $K = 8$ (right).



(c) ResNet-110 with different auxiliary networks for $K = 16$.

Figure A.5: Epochs vs Linear Evaluation Top-1 Accuracy, evaluated for ResNet-110 after every 100 epochs, trained using different auxiliary networks ('0c2f' and '1c2f') for different values of K.

References

Alain, G., & Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*.

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2019a). Greedy layerwise learning can scale to imagenet. *Proceedings of the 36th International Conference on Machine Learning (ICML)*.

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2019b). Greedy layerwise learning can scale to imagenet. In *International conference on machine learning* (pp. 583–593).

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2020, 13–18 Jul). Decoupled greedy learning of CNNs. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (Vol. 119, pp. 736–745). PMLR. Retrieved from https://proceedings.mlr.press/v119/belilovsky20a.html

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (pp. 153–160).

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... others (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Chen, C.-C., Yang, C.-L., & Cheng, H.-Y. (2018). Efficient and robust parallel dnn training through

model parallelism on multi-gpu platform. *arXiv preprint arXiv:1809.02839*.

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International conference on machine learning* (pp. 1597–1607).

Chilimbi, T., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. In *11th usenix symposium on operating systems design and implementation (osdi 14)* (pp. 571–582).

Choromanska, A., Cowen, B., Kumaravel, S., Luss, R., Rigotti, M., Rish, I., ... others (2019). Beyond backprop: Online alternating minimization with auxiliary variables. In *International conference on machine learning* (pp. 1193–1202).

Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... others (2012). Large scale distributed deep networks. *Advances in neural information processing systems*, *25*.

Doersch, C., Gupta, A., & Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the ieee international conference on computer vision* (pp. 1422–1430).

Dvornik, N., Schmid, C., & Mairal, J. (2019). Diversity with cooperation: Ensemble methods for few-shot classification. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 3723–3731).

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 201–208).

Fahlman, S., & Lebiere, C. (1989). The cascade-correlation learning architecture. *Advances in neural information processing systems*, *2*.

Gomez, A. N., Key, O., Gou, S., Frosst, N., Dean, J., & Gal, Y. (2020). Interlocking backpropagation: Improving depthwise model-parallelism. *arXiv preprint arXiv:2010.04116*.

Gomez, A. N., Key, O., Perlin, K., Gou, S., Frosst, N., Dean, J., & Gal, Y. (2022). Interlocking

backpropagation: Improving depthwise model-parallelism. *Journal of Machine Learning Research*, *23*(171), 1–28.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Guan, L., Yin, W., Li, D., & Lu, X. (2019). Xpipe: Efficient pipeline model parallelism for multi-gpu dnn training. *arXiv preprint arXiv:1911.04610*.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision* (pp. 630–645).

Hebb, D. (1949). *The organization of behavior. emphnew york.* Wiley.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., ... others (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, *29*(6), 82–97.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Huang, G., Liu, Z., Pleiss, G., Van Der Maaten, L., & Weinberger, K. (2019). Convolutional networks with dense connectivity. *IEEE transactions on pattern analysis and machine intelligence*.

Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., & Chen, Z. (2018). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*.

Huo, Z., Gu, B., & Huang, H. (2018a). Training neural networks using features replay. *Advances in Neural Information Processing Systems*.

Huo, Z., Gu, B., & Huang, H. (2018b). Training neural networks using features replay. *Advances in*

Neural Information Processing Systems, *31*.

Huo, Z., Gu, B., Huang, H., et al. (2018). Decoupled parallel backpropagation with convergence guarantee. In *International conference on machine learning* (pp. 2098–2106).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Ivakhnenko, A. G., & Lapa, V. G. (1965).

Cybernetic Predicting Devices. CCM Information Corporation..

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., & Kavukcuoglu, K. (2017a). Decoupled neural interfaces using synthetic gradients. *International Conference of Machine Learning*.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., & Kavukcuoglu, K. (2017b). Decoupled neural interfaces using synthetic gradients. *International Conference of Machine Learning*.

Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D., & Makedon, F. (2020). A survey on contrastive self-supervised learning. *Technologies*, *9*(1), 2.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1725–1732).

Keuper, J., & Preundt, F.-J. (2016). Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. In *2016 2nd workshop on machine learning in hpc environments (mlhpc)* (pp. 19–26).

Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., … Krishnan, D. (2020). Supervised contrastive learning. *Advances in Neural Information Processing Systems*, *33*, 18661–18673.

Kim, D., Woo, S., Lee, J.-Y., & Kweon, I. S. (2019). Deep video inpainting. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 5792–5801).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingsford, C., & Salzberg, S. L. (2008). What are decision trees? *Nature biotechnology*, *26*(9), 1011–1013.

Kohavi, R., et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, pp. 1137–1145).

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kukačka, J., Golkov, V., & Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*.

Laskin, M., Metz, L., Nabarro, S., Saroufim, M., Noune, B., Luschi, C., ... Abbeel, P. (2020). Parallel training of deep networks with local updates. *arXiv preprint arXiv:2012.03837*.

Laskin, M., Metz, L., Nabarro, S., Saroufim, M., Noune, B., Luschi, C., ... Abbeel, P. (2021). *Parallel training of deep networks with local updates.* Retrieved from https://openreview.net/forum?id=ufS1zWbRCEa

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995.

Lee, D.-H., Zhang, S., Fischer, A., & Bengio, Y. (2015). Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 498–515).

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, *42*, 60–88.

Liu, X., Zhang, F., Hou, Z., Mian, L., Wang, Z., Zhang, J., & Tang, J. (2021). Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*.

Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, *9*, 381–386.

Nøkland, A. (2016). Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, *29*.

Nøkland, A., & Eidnes, L. H. (2019). Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*.

Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015). Deep face recognition.

Patel, A., Eickenberg, M., & Belilovsky, E. (2022). Local learning with neuron groups. In *From cells to societies: Collective learning across scales.*

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2536–2544).

Pyeon, M., Moon, J., Hahn, T., & Kim, G. (2021). {SEDONA}: Search for decoupled neural networks toward greedy block-wise learning. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=XLfdzwNKzch

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California Univ San Diego La Jolla Inst for Cognitive Science.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks* (pp. 92–101).

Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., ... others (2022). Laion-5b: An open large-scale dataset for training next generation image-text models. *arXiv preprint arXiv:2210.08402*.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1–9).

Veness, J., Lattimore, T., Budden, D., Bhoopchand, A., Mattern, C., Grabska-Barwinska, A., ...

others (2019). Gated linear networks. *arXiv preprint arXiv:1910.01526*.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, *11*(12).

Wang, Y., Ni, Z., Song, S., Yang, L., & Huang, G. (2021). Revisiting locally supervised learning: an alternative to end-to-end training. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=fAbkE6ant2

Xiong, Y., Ren, M., & Urtasun, R. (2020). Loco: Local contrastive representation learning. *Advances in neural information processing systems*, *33*, 11142–11153.

Xu, N., Yang, L., Fan, Y., Yue, D., Liang, Y., Yang, J., & Huang, T. (2018). Youtube-vos: A large-scale video object segmentation benchmark. *arXiv preprint arXiv:1809.03327*.

You, Y., Gitman, I., & Ginsburg, B. (2017). Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, *6*(12), 6.

Zhai, X., Oliver, A., Kolesnikov, A., & Beyer, L. (2019). S4l: Self-supervised semi-supervised learning. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 1476–1485).