Efficient Neural Network Inference and Training Using Early Exit Strategies

BY

ALPEREN GÖRMEZ
B.S., Bilkent University, 2019

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2024

Chicago, Illinois

Dedicated to

My family

ACKNOWLEDGMENTS

I would like to thank several people for their help and support during my Ph.D. journey.

First and foremost, I would like to thank my advisor Prof. Koyuncu for his invaluable guidance. He has been very understanding since day one and I am grateful for that. I think his straightforward approach, offering clear solutions and precise suggestions significantly sharpened my problem-solving skills not only in technical sense but also in non-technical context. If I ever improved in this area, it is thanks to his mentorship.

Next, I would like to thank my dear mother, my father, my brother Onuralp and my sister Eylül. Being apart from them for an extended period has been challenging, perhaps even more so for them than for me. I deeply cherish their continuous support. Their unconditional love, constant encouragement, and motivating presence mean the world to me. I consider myself incredibly fortunate to have them by my side. I think I wouldn't have reached this point without their constant support.

Lastly, a special thanks to Afagh, who not only supported me through my Ph.D. but also navigated her own journey alongside. I am grateful for her patient listening and persistent reminders about the importance of maintaining a balance between work and life. Her unwavering belief in me, never doubting for a moment, and constant encouragement during challenging times matter more than words can say.

AG

CONTRIBUTION OF AUTHORS

A version of Chapter 2 has been published in the 2022 IEEE World Congress on Computational Intelligence: International Joint Conference on Neural Networks, and was presented in the 2022 Eastern European Machine Learning Summer School where it received the top-voted poster award. This work was conducted jointly with Drs. Dasari and Koyuncu. A version of Chapter 3 has been published in IEEE Signal Processing Letters. A version of Chapter 4 has been featured in the 2023 International Conference on Machine Learning, in the Localized Learning Workshop. A version of Chapter 5 has been featured in the 2024 International Conference on Machine Learning, in the Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization. Prof. Koyuncu advised me throughout the dissertation.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER PAGE

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

E^2CM	Early Exit via Class Means
CBT	Class Based Thresholding
EEPrune	Early Exit Prune
LLM	Large Language Model
AWGN	Additive White Gaussian Noise

SUMMARY

This work aims to reduce the inference and training costs of deep learning models by utilizing early exit networks. In particular, we introduce four algorithms:

1. E^{2}CM, a simple and lightweight early exit algorithm that reduces the inference cost. In a separate line of work, we also show how early exit networks can be combined with model pruning.
2. CBT, an algorithm to further decrease the inference cost of early exit semantic segmentation networks.
3. EEPrune, a novel dataset pruning algorithm that uses early exit networks to reduce training cost.
4. Class-aware EE LLM, a novel weight initialization algorithm for early exit large language models to accelerate pre-training.

CHAPTER 1

INTRODUCTION

Deep learning is advancing rapidly with the constant development of new, state-of-the-art models. The performance increase in the models are often attributed to increase in their size and training them for longer on larger datasets. However, these two factors result in a rise in the inference and training costs. This rise may not be sustainable in the long run and is thus a significant problem that needs to be addressed, especially for resource-constrained settings.

One way of reducing the inference cost is adding early exit layers to the model. By doing so, the computation will be terminated sooner for a sample that can be classified correctly at an early exit layer. However, existing early exit methods are not suitable for resource-constrained settings. The reason for that is three fold. First, existing early exit methods add layers that require gradient based training to the model, which increases the model complexity. Second, the added layers have to be trained jointly with the model and this increases the computational cost. Finally, existing methods introduce more hyper-parameters to the model, such as number, location and size of the exit layers. This makes hyper-parameter tuning more challenging and expensive. To overcome these limitations and reduce the inference cost of early exit models, we introduce Early Exit via Class Means (E^{2}CM), a simple and lightweight early exit algorithm. We compare E^{2}CM against various early exit methods on different models for numerous supervised and unsupervised tasks, and demonstrate its effectiveness in reducing the inference cost.

In a separate line of work, we demonstrate how early exit networks can be combined with other efficiency optimization techniques such as model pruning. We explore whether the order of pruning the backbone and the exit layers impacts performance, and find that pruning everything simultaneously yields the best outcome.

We also design Class Based Thresholding (CBT) to decrease the inference cost of early exit semantic segmentation networks further in settings where the number of classes is high and the classes have different intrinsic difficulties. Specifically, CBT utilizes the neural collapse phenomenon by calculating the mean of the prediction probabilities of pixels in the training set, for each class. Then, CBT transforms the probabilities to thresholds using a set of formulas, resulting in a different threshold for each class. We demonstrate the effectiveness of CBT by comparing it to existing methods on various datasets.

In order to reduce the training costs, we resort to dataset pruning. Dataset pruning addresses the high training costs of deep learning models by discarding redundant samples and keeping, for example, only the difficult samples for training. Existing dataset pruning methods typically train an ensemble of models or train a single model fully for assessing the redundancy of a training sample. This is particularly undesirable for resource-constrained devices due to the large memory footprint of the ensemble and high computational demands of full training. For this reason, we introduce Early Exit Prune (EEPrune), a novel dataset pruning algorithm that utilizes early exit networks to discard redundant samples from the dataset. We demonstrate EEPrune's superiority against existing dataset pruning methods using various models on different datasets.

Finally, we propose a novel class-aware weight initialization technique for early exit large language models with the purpose of accelerating pre-training. Our design utilizes the neural collapse phenomenon combined with a Gaussian mixture model for the distribution of feature vectors at a given layer. Specifically, we calculate the average of token representations at the early exit point and use the resulting vectors together with class probabilities for initializing the early exit vectors. The next token prediction accuracy of our class-aware initialization technique is up to five times higher than other baselines at epoch zero and matches or surpasses them in later epochs throughout the pre-training process.

1.1 Thesis Contributions

1.1.1 E^{2}CM: Early Exit via Class Means for Efficient Supervised and Unsupervised Learning

- We design E^{2}CM, a simple and lightweight early exit algorithm to reduce inference cost of deep learning models. E^{2}CMdoes not add trainable layers, does not need gradient based training, and does not need any additional hyper-parameters.
- We show E^{2}CM's feasibility for unsupervised learning tasks, whereas the existing methods focus only on supervised learning tasks.
- We compare E^{2}CMagainst various early exit methods on different models for numerous supervised and unsupervised tasks and demonstrate its effectiveness in reducing the inference cost.

1.1.2 Pruning Early Exit Neural Networks

- We combine two methods to reduce the computational cost: Pruning and early exit networks.
- We explore whether the order of pruning the backbone and the exit layers impacts performance, and find that pruning everything simultaneously yields the best outcome.
- We show the processes of pruning and early exit can potentially be separated without significant penalty in performance.

1.1.3 Class Based Thresholding in Early Exit Semantic Segmentation Networks

- We design CBT, a new algorithm to further decrease the inference cost of early exit semantic segmentation networks.
- We demonstrate how CBT performs better in settings where the number of classes is high and the classes have different intrinsic difficulties.
- We demonstrate the effectiveness on CBT by comparing it to existing methods on various datasets.

1.1.4 Dataset Pruning Using Early Exit Networks

- We use early exit networks, an inference time reduction technique, for the task of dataset pruning, a training time reduction technique.
- We introduce EEPrune, a novel dataset pruning algorithm capable of maintaining baseline accuracy and occasionally surpassing it, while consuming significantly less energy compared to other dataset pruning methods.

- We shed light on the impact of exit location on the performance of identifying easy samples.
- We conduct an extensive evaluation of dataset pruning methods and assess their ability to maintain a balanced representation across different classes during the pruning process.

1.1.5 Class-aware Initialization of Early Exits for Pre-training Large Language Models

- We propose a novel class-aware weight initialization technique for early exit large language models with the purpose of accelerating pre-training.
- We make connections to the optimal detection problem for the vector AWGN channel from the digital communications domain.
- We show that our method performs better than baselines in both "no freezing" and "freezing" settings for various model families and datasets.

CHAPTER 2

E^{2}CM: EARLY EXIT VIA CLASS MEANS FOR EFFICIENT SUPERVISED AND UNSUPERVISED LEARNING

Part of this chapter has been pusblished as A. Görmez, V. R. Dasari and E. Koyuncu, "E2CM: Early Exit via Class Means for Efficient Supervised and Unsupervised Learning," 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 2022.

Overview: State-of-the-art neural networks with early exit mechanisms often need considerable amount of training and fine-tuning to achieve good performance with low computational cost. We propose a novel early exit technique, *Early Exit Class Means (E^{2}CM)*, based on class means of samples. Unlike most existing schemes, E^{2}CM does not require gradient-based training of internal classifiers and it does not modify the base network by any means. This makes it particularly useful for neural network training in low-power devices, as in wireless edge networks. We evaluate the performance and overheads of E^{2}CM over various base neural networks such as MobileNetV3, EfficientNet, ResNet, and datasets such as CIFAR-100, ImageNet, and KMNIST. Our results show that, given a fixed training time budget, E^{2}CM achieves higher accuracy as compared to existing early exit mechanisms. Moreover, if there are no limitations on the training time budget, E^{2}CM can be combined with an existing early exit scheme to boost the latter's performance, achieving a better trade-off between computational cost and network accuracy. We also show that E^{2}CM can be used to decrease the computational cost in unsupervised learning tasks.

Keywords: Neural networks, early exit, class means.

2.1 Introduction

Modern deep learning models require a vast amount of computational resources to effectively perform various tasks such as object detection [3], image classification [4], machine translation [5] and text generation [6]. Deploying deep learning models to the edge, such as to mobile phones or the Internet of Things (IoT), thus becomes particularly challenging due to device computation and energy limitations [7,8]. Moreover, the law of diminishing returns applies to the computation-performance trade-off [9]: The increase in the performance of a deep learning model is often marginal as compared to the increase in the amount of computation.

One of the primary reasons behind traditional deep learning models' high computation demand is their tunnel-like design. In fact, traditional models apply the same sequence of operations to any given input. However, in many real world datasets, certain inputs may consist of much simpler features as compared to other inputs [9]. In such a scenario, it becomes desirable to design more efficient architectures that can exploit the heterogeneous complexity of dataset members. This can be achieved by introducing additional exit points to the models [10–15]. These exit points prevent simple inputs to traverse the entire network, reducing the computational cost of inference.

Despite reduced inference time, existing early exit neural network architectures require additional training and fine-tuning for the early exit points, which increases the training time [9, 11, 12]. This side-effect is undesirable for scenarios in which the training has to be done in a low-power device. An ideal solution is a plug-and-play approach that does not require gradient-based training and performs well. In this work, we propose such an early-exit mechanism,

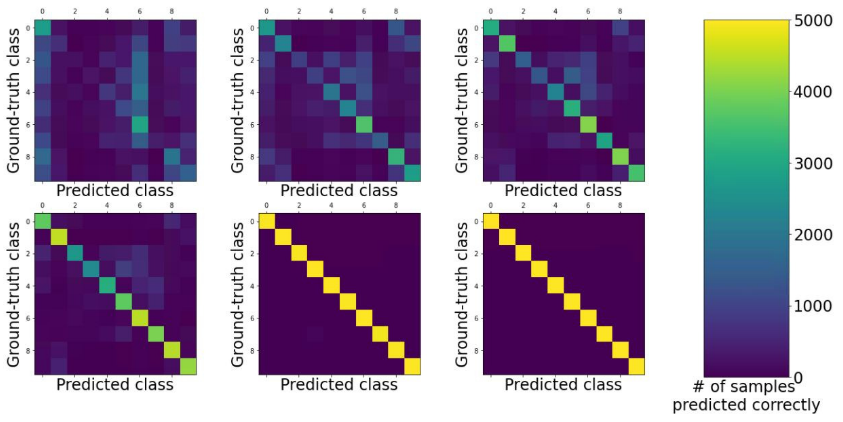


Figure 1. Confusion matrix of the classifications done according to the nearest class mean on CIFAR-10 training set. From left to right, top to bottom; the results belong to the first convolutional layer, 1^{st}, 3^{rd}, 10^{th}, 15^{th} and 30^{th} residual block of ResNet-152.

Early Exit Class Means (E^2CM), based on the *class means* of input samples for the image classification task. By averaging the layer outputs for each class at every layer of the model, class means are obtained. During inference, output of a layer is compared with the corresponding class means using Euclidean distance as the metric. If the output of the layer is close enough to a class mean, the execution is stopped and the sample exits the network. In fact, as seen in Figure 1, some samples can be classified easily at early stages of the network by just considering a "nearest class mean" decision rule, suggesting the potential effectiveness of our method for reducing computational cost.

A practical use case for E^2CM is when a large, expensive-to-train model is broadcast to edge devices with limited and heterogeneous computation capabilities. In such a scenario, different devices may train the E^{2}CM model at different FLOP operating points depending on their computation capabilities or power limitations.

One other practical scenario for E^{2}CM is transfer learning and fine tuning, where the training has to be done on low-power edge device with a local dataset that is different from the dataset the base network was trained on. In order to keep the battery usage minimum on the edge device, the training time may be limited. The advantage of E^{2}CM is that it does not need any gradient-based training like Shallow-Deep Networks [12], therefore it is more suitable for transfer learning and fine tuning on low-power devices.

To the best of our knowledge, E^{2}CM is the first early exit mechanism that does not require gradient-based training and does not modify the original network by any means. Moreover, E^{2}CM does not have a hyper-parameter for early exit locations unlike existing schemes. These features make E^{2}CM simpler to use and easier to deploy on low-power devices. While using class means as the only early exit mechanism requires just a single feed forward pass, existing early exit methods reaches the same performance after training for multiple epochs (i.e. multiple forward and backward passes), which suggests our method is more agile and powerful yet simpler. Moreover, combining E^{2}CM with the existing mechanisms that require gradient-based training achieves a better trade-off in terms of computation cost and network accuracy. In addition to its benefits in the supervised learning setting, E^{2}CM is also the first technique that shows the feasibility of early exits in the unsupervised learning setting.

We show the effectiveness of E^{2}CM on CIFAR-10 [16], CIFAR-100 [16], ImageNet [17], Tiny ImageNet [18], KMNIST [19], Fashion-MNIST [20] and MNIST [21] datasets using ResNet-18 [4], ResNet-152 [4], WideResNet-101 [22], MobileNetV3Large [23] and EfficientNet-B0 [24] models. Using class means as the sole decision mechanism results in 50% better accuracy or 50% faster inference time compared to the existing early exit techniques in the literature. Also, when combined with the state-of-the-art, we increase the accuracy by 6% without doing further computation; or decrease the inference time by 33% with a negligible loss in accuracy. We also show that it is possible to decrease the computational cost while doing clustering with autoencoders on MNIST and Fashion-MNIST [20,21]. In particular, E^{2}CM saves on computation by 60%, while the loss in unsupervised clustering accuracy [25] is marginal. We also evaluate the computational and memory overheads of E^{2}CM, and show that they are practically low even for large datasets and/or models.

2.2 Related Work

Conditional computation: Our work is related to the area of conditional computation [26–28], where several small networks are trained to control the computation flow of one deep neural network. For this purpose, adding gates between the blocks of residual networks have been proposed [28, 29]. During inference, these gates allow the input to skip unnecessary blocks, thus saving computation time. However, the gates have to be trained from scratch jointly with the base network. Also, the locations of the gates have to be determined explicitly, which result in increased number of hyper-parameters. Unlike these methods, E^{2}CM does not modify the base network and does not require gradient-based training.

Early exit networks: One of the earliest works that explicitly propose the idea of early exiting is [10], where the authors consider adding a cascade of linear layers after convolutional layers as control blocks. Rather than just linear layers, adding *branches* consisting of convolutional layers to the original model has also been studied [11]. A significant drawback of this method is that the branches increase the computational cost due to convolutional layers. In a more recent study, *internal classifiers (ICs)* consisting of a feature reduction layer and a single linear layer are added after certain layers in the network [12, 15]. The methods presented in these studies modify the original network by adding linear or convolutional layers. Moreover, they require gradient updates to train those layers. Also, an implicit hyper-parameter is the locations of the early exit points. E^2CM is better suited for low-power applications compared to existing studies since we do not modify the original model, do not require gradient based training and additional hyper-parameters. Another novelty of E^2CM is the ability to extend to unsupervised learning tasks. Existing early exit methods focus only on supervised learning.

In addition to layer level early exits, a network level early exit mechanism has been introduced in [9]. Both the layer level exit and the network level exit require decision functions to be inserted between the layers and networks. This type of architecture freezes the weights of the original model, and then trains the decision functions one by one using weighted binary classification. The drawback of this approach is such an alternative optimization, which may consume a lot of time and energy when there are many decision functions to optimize.

Multi-resolution networks: One other idea to reduce the inference time computational cost of neural networks is the usage of multi-resolution features to facilitate early exiting [30–32]. While this idea works well, these methods focus only on supervised learning tasks and they design new architectures for that purpose. Moreover, the locations of early exits are restricted to the last few blocks of the sub-network [31]. Rather than designing new architectures, E^{2}CM is a plug-and-play method: We focus on taking off-the-shelf networks such as MobileNetV3Large [23] and reducing the inference time without modifying and retraining the model for both supervised and unsupervised learning tasks.

Few-shot learning: Intermediate layer outputs have been used for classification in few-shot and one-shot learning settings in the past [33–36]. These studies are closely related to the area of *metric learning*. The closest work to E^{2}CM is *prototypical networks*, in which *prototypes* for each class are computed [35, 37]. However, none of those approaches aim to reduce the computation cost, rather they either try to remedy the problem of classifying unseen classes or explore unsupervised domain adaptation [37, 38].

Neural collapse: E^{2}CM is also related to the phenomenon of neural collapse [39]. It is known that as the inputs go deeper in a neural network, the classes are separated better from each other as a result of multiple nonlinearities, and the samples begin to concentrate [40, 41]. E^{2}CM exploits this phenomenon with the main idea of stopping the execution as soon as the sample is close enough to a concentration point i.e., a class mean.

2.3 Early Exit Class Means

In this section, we introduce E^{2}CM in the context of image classification. The scheme extends to different classification tasks in the same manner. Let $(x_0^{(i)}, y^{(i)}) \in D$ be an image-label pair from the dataset D consisting of N samples and K distinct classes, where $y^{(i)} \in \{1, 2, \ldots, K\}$ and $i \in \{1, 2, \ldots, N\}$. We denote the network F with M layers as a sequence $l_1, l_2, \ldots, l_M$. Let $\hat{y}^{(i)}$ denote the prediction of the network, $x_j^{(i)}$ denote the output of layer j, and $\hat{y}_j^{(i)}$ denote the prediction in case the input exits the network after layer j, for $j = 1, 2, \ldots, M$. The input-output relationships of the network F can be expressed as

$$x_j^{(i)} = l_j(x_{j-1}^{(i)}),\ j = 1, 2, \ldots, M. \tag{2.1}$$

2.3.1 Class Means

The input to E^{2}CM is the network F trained on D. The network F is not modified by any means. Therefore, we can obtain the class means for each class at each layer easily by just a forward pass. This is especially useful when the training time budget is fixed. Let S_k denote the set of samples whose ground-truth label is k, and c_j^k denote the mean of the output of layer j for class k. In other words, let

$$c_j^k = \tfrac{1}{|S_k|} \textstyle\sum_{n \in S_k} x_j^{(n)}. \tag{2.2}$$

The Euclidean distance between a layer output $x_j^{(i)}$ and K class means c_j^k is then computed via

$$d_j^{k^{(i)}} = ||x_j^{(i)} - c_j^k||_2,\ k \in \{1, 2, \ldots, K\}. \tag{2.3}$$

After calculating $d_j^{k^{(i)}}$ at each layer for every sample in the dataset, we normalize the distances for each class as

$$d_j^{k^{(i)}} := \frac{d_j^{k^{(i)}}}{\frac{1}{N}\sum_{i=1}^{N} d_j^{k^{(i)}}},\ k \in \{1, 2, \ldots, K\}. \tag{2.4}$$

Finally, the normalized distances are converted to probabilities of input belonging to a class in order to perform inference. This is done using the softmax function as

$$P(\hat{y}_j^{(i)} = k) = \text{softmax}(-d_j^{k^{(i)}}). \tag{2.5}$$

During inference, the decision of exiting after l_j or moving forward to l_{j+1} is made according to a threshold value T_j. If the largest softmax probability is greater than the specified threshold T_j, execution is stopped and the class with the largest softmax probability is predicted. In other words, if

$$\max(\text{softmax}(-d_j^{k^{(i)}})) > T_j, \tag{2.6}$$

then the network predicts

$$\hat{y}_j^{(i)} = \arg\max_k(\text{softmax}(-d_j^{k^{(i)}})). \tag{2.7}$$

Otherwise, the input moves forward to the next layer. In the worst case, execution ends at the last layer of the network. The full procedure of our method is shown in Algorithm 1.

E^{2}CM performs differently according to different set of threshold values. We use binary search to reach target number of FLOPs on training set. If the thresholds result in a higher number of FLOPs than the target number of FLOPs, they are decreased to encourage early exits. Else, they are increased to encourage moving deeper in the layers. Later, same threshold values are used on the test set during the inference phase for that target number of FLOPs.

Algorithm 1 Early Exit Class Means (E^{2}CM)

```
Input: Trained network layers $l_j$, dataset $D$, thresholds $T_j$
if training then
    for $j = 1$ to $M$ do
        $x_j^{(i)} = l_j(x_{j-1}^{(i)})$
        Calculate class means $c_j^k$, $k \in \{1, 2, \ldots, K\}$
    end for
end if
if inference then
    for $j = 1$ to $M$ do
        $x_j^{(i)} = l_j(x_{j-1}^{(i)})$
        Compute $d_j^{k^{(i)}} = ||x_j^{(i)} - c_j^k||_2$
        Normalize $d_j^{k^{(i)}}$ as in (Equation 2.4).
        if $\max(\text{softmax}(-d_j^{k^{(i)}})) > T_j$ then
            Early exit with $\arg\max_k(\text{softmax}(-d_j^{k^{(i)}}))$.
        end if
    end for
end if
```

2.3.2 Combination of Class Means with Existing Schemes

E^{2}CM can boost the performance of existing early exit schemes. Existing methods use only $x_j^{(i)}$ as the input to the internal classifiers and decide to exit early based on either the entropy of class probabilities or the largest class probability [11,12]. We propose feeding the distances to the class means ($d_j^{k^{(i)}}$) as additional inputs to the internal classifiers by simple concatenation, which improves the performance. In addition, during inference, if the j^{th} internal classifier decides to move to the next layer, E^{2}CM is consulted. If E^{2}CM suggests exiting early, the prediction of the j^{th} internal classifier is returned and the input exits early. Hence, an input can move to the next layer if and only if it receives approval from both the internal classifier (which can be based on any existing scheme) and our simple E^{2}CM.

2.3.3 Extension to Unsupervised Learning

E^{2}CM can be used for clustering as well. As an example, we consider Deep Embedding Clustering (DEC) [25] and focus on the task of jointly learning representations and cluster assignments. In DEC, there is only one clustering layer, and it is at the end of the encoder layers. This makes it impractical for low-power clustering, because the architecture is like a tunnel with only one exit at the end. We propose adding multiple clustering layers as early exits in order to decrease the computational cost.

2.4 Results

We validate the effectiveness of E^{2}CM on CIFAR-10 [16], CIFAR-100 [16], ImageNet [17], Tiny ImageNet [18], KMNIST [19], Fashion-MNIST [20] and MNIST [21] datasets using ResNet-18 [4], ResNet-152 [4], WideResNet-101 [22], MobileNetV3Large [23] and EfficientNet-B0 [24] models.

Datasets: CIFAR-10 and CIFAR-100 datasets consist of 50000 training and 10000 test images, and have 10 and 100 classes respectively with equal amount of samples for each class. ImageNet dataset has 1000 classes and consists of 1.2 million training images and 50000 validation images. Tiny Image-Net dataset consists of 100000 training, 10000 validation and 10000 test images of 200 classes. KMNIST, Fashion-MNIST and MNIST datasets consist of 60000 training and 10000 test images. While MNIST is a dataset of handwritten digits, KMNIST consists of 10 Hiragana characters and Fashion-MNIST consists of clothing images. KMNIST, Fashion-MNIST and MNIST datasets consist of grayscale images unlike other datasets described above.

Models and training: We use ResNet-18, ResNet-152, WideResNet-101, MobileNetV3Large and EfficientNet-B0 models for our supervised learning experiments. For all models, we use the same data augmentation scheme and the hyper-parameter values stated in [28] for training. For our unsupervised learning experiments, we use the same network architecture and training parameters stated in [25].

Experiments: We run experiments in four settings. First, we fix the training time budget and compare E^{2}CM with the existing early exit methods in terms of network accuracy and floating point operations (FLOPs) performed during inference. The second experiment is similar to the first one: Training time budget is fixed, but now for a fine tuning task. In the third experiment, we remove the training time budget. We allow the full training of the internal classifiers, and we combine E^{2}CM with internal classifiers and compare it with existing methods. The fourth experiment is for unsupervised learning, and we generate the accuracy-FLOPs curve to evaluate the effectiveness of E^{2}CM.

2.4.1 E^{2}CM Under a Fixed Training Time Budget

In the fixed training time budget setting, we compare E^{2}CM with existing methods in two ways. First, E^{2}CM is compared with Shallow-Deep Networks [12] and BranchyNet [11], which are trained for only one epoch since E^{2}CM requires only a single forward pass. We do not include here the Bolukbasi-Wang-Dekel-Saligrama (BWDS) method [9], because in this method, each decision function requires a separate training. Hence, we cannot train an entire BWDS network with many decision functions using only one epoch.

We use the procedure described in [12] and add 6 ICs after the layers which correspond to the 15%, 30%, 45%, 60%, 75%, 90% of the entire network in terms of FLOPs for Shallow-Deep Networks. For BranchyNet, we add 2 branches to the original network as in [11]. The first branch is after the first convolutional layer, and the second branch is after the layer that corresponds to 1/3 of the whole network in terms of FLOPs. For E^2CM, we allow early exiting after every ResNet block for simplicity.

To reach a target number of FLOPs, we first initialize the threshold vector by drawing M numbers uniformly at random from $[0, 1]$. The j^{th} component of the threshold vector corresponds to T_j in Algorithm 1 and is utilized at layer l_j of the neural network. Then, the softmax values are obtained for each layer on training set as in Equation 2.5. Therefore, only one pass on training set is needed to optimize the thresholds. We update the threshold vector until we reach the target number of FLOPs using binary search at each component and alternating optimization: If the thresholds give larger number of FLOPs than the target FLOP, the thresholds are decreased for the next iteration. Otherwise, they are increased. Binary search is guaranteed to converge as the (average) FLOPS are monotonic with respect to the thresholds.

In this paper, we normalize the FLOPs to the base model. Thus, the base model without any early exits is assumed to cost 1 FLOP. To obtain the trade-off between accuracy and FLOPs for early exit models synthesized from a base model, the above threshold optimization process is repeated on the training set for target FLOPs ranging from 0.0 to 1.0 with 0.001 granularity. We then compute the convex hull of the resulting 1000 points in the FLOPs-Accuracy plane. This yields the best performing thresholds on the training set. We use the best thresholds on

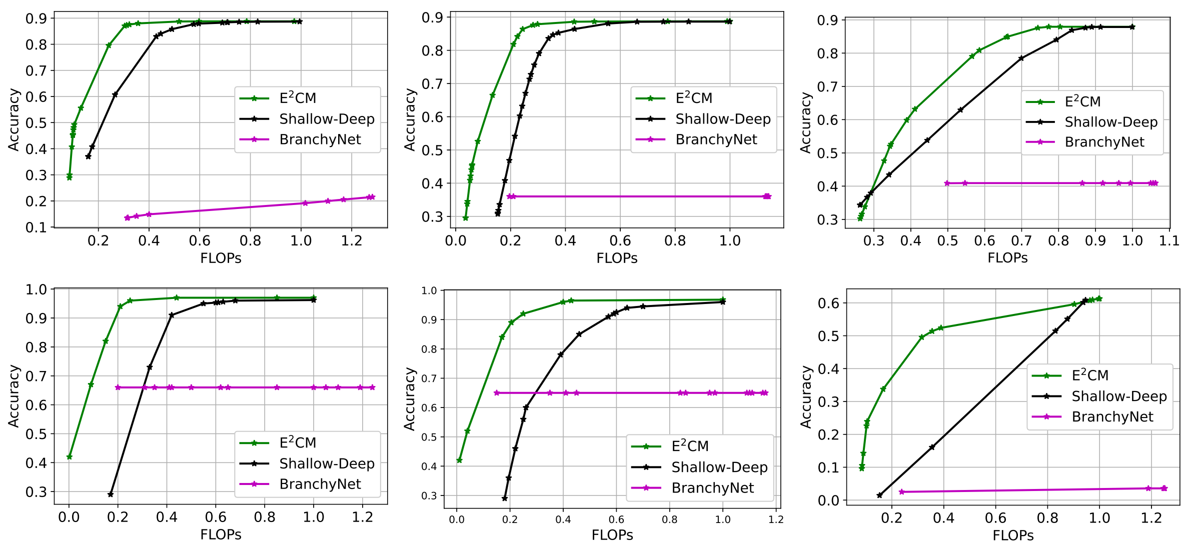


Figure 2. Comparison of E^{2}CM with existing methods under fixed training time budget of one epoch for ResNet-152 (left) and WideResNet-101 (middle) on CIFAR-10 (top) and KMNIST (bottom). ResNet-18 for CIFAR-10 is shown at top right, and ResNet-152 for CIFAR-100 is shown at bottom right.

test set and form the FLOPs-Accuracy curves. Each point on a given curve represents one threshold vector. Points are connected via lines: The performance of any point on any given line is achievable simply via time sharing of the models that correspond to line endpoints.

We show the performance of E^{2}CM for a fixed training time budget in Figure 2. The horizontal axis represents the FLOPs normalized the base model, and the vertical axis represents the accuracy. As shown in the figure, E^{2}CM generally outperforms all existing schemes. Specifically, using E^{2}CM as the only decision mechanism achieves 50% better accuracy or 50% faster inference time for certain cases. This is because E^{2}CM uses the trained weights of the original vanilla model, which generalizes well to the dataset and can be used for classification. On the other hand, internal classifiers require further training. Therefore, they require more time to be ready for the task of classification.

Secondly, we still consider a fixed training time budget, but this time we allow the separate training of decision functions in the BWDS method. As suggested by the authors, there are 6 early exit points, hence 6 decision functions. We train each decision function for 1 epoch, resulting in 6 separate epochs. To make a fair comparison, we train Shallow-Deep Networks and BranchyNet for 6 epochs as well. We combine E^{2}CM with Shallow-Deep Networks. By doing so, E^{2}CM rectifies the decisions made by Shallow-Deep Networks. At every exit point of Shallow-Deep Networks, if the decision is made in favor of exiting early at the exit point, E^{2}CM is consulted. If E^{2}CM disagrees with Shallow-Deep Networks, early exit does not happen at that exit point.

From Table I, it can be seen that combining E^{2}CM with Shallow-Deep Networks (SDN) performs better than other methods and can increase the performance by 25% when the number of classes is large. Table I shows accuracies at $\phi \in \{0.15, 0.20, 0.25, 0.30\}$ FLOPs only, because after 0.30 FLOPs, the accuracy stays the same, which indicates that we do not need the tunnel-like design of traditional networks.

Overheads: There are two main sources of overhead for our E^{2}CM scheme that should be considered. One is the extra computation overhead to calculate the distances to class means. In this context, we would like to note that the results in Figure 2 already include the computation overhead as the FLOPs are normalized to the base model. The superior performance of E^{2}CM suggest that the computation overhead is very low. In fact, for the WideResNet-101 and

TABLE I

COMPARISON OF E^{2}CM WITH EXISTING METHODS UNDER A FIXED TRAINING TIME BUDGET OF SIX EPOCHS. SYMBOL ϕ REPRESENTS THE FLOP CONSTRAINT.

Model, Dataset	**Method**	**Accuracy**			
		ϕ=0.15	ϕ=0.20	ϕ=0.25	ϕ=0.30
ResNet-152, CIFAR-10	**E^{2}CM+SDN**	**77%**	**85%**	**87%**	**87%**
	SDN	66%	75%	85%	87%
	BWDS	77%	82%	84%	86%
	BranchyNet	57%	57%	57%	57%
ResNet-152, CIFAR-100	**E^{2}CM+SDN**	**34%**	**46%**	**52%**	**54%**
	SDN	20%	24%	27%	30%
	BWDS	33%	40%	42%	44%
	BranchyNet	24%	25%	26%	26%
WideResNet-101, CIFAR-10	**E^{2}CM+SDN**	81%	**87%**	**88%**	**88%**
	SDN	70%	79%	86%	88%
	BWDS	**82%**	85%	87%	88%
	BranchyNet	56%	56%	57%	57%
WideResNet-101, KMNIST	**E^{2}CM+SDN**	85%	**92%**	**96%**	**96%**
	SDN	63%	74%	96%	96%
	BWDS	**88%**	90%	92%	94%
	BranchyNet	80%	82%	83%	84%

ResNet-152 models and CIFAR-10 dataset, our experiments have shown that the computational complexity of an early exit block is only around 0.007 FLOPs. For CIFAR-100 dataset, the overhead is about 0.057 FLOPs. An issue in the case of a very large number of classes is that the computational complexity of E^{2}CM scales linearly with the number of classes. This is also evident with the roughly 10-fold scaling (from 0.007 to 0.057) of the computational complexity as soon as one considers CIFAR-100 instead of CIFAR-10. One solution that we will explore in the next set of experiments is to utilize max pooling to reduce the class means' dimensions.

We will show that this technique can effectively reduce the computation overhead while still preserving the advantages of E^{2}CM over existing schemes.

We have also evaluated the memory overhead of our E^{2}CM scheme, which is important for edge computing devices with limited resources. Note that, as compared to the base model, E^{2}CM requires extra memory for storing the class means vectors. For ResNet-152, WideResNet-101 and ResNet-18 on CIFAR-10 models, we have found that our E^{2}CM needs 0.417 MB, 0.43 MB and 0.242 MB of space to store the class means for an exit location respectively. These numbers are averages over different layers of the network. Considering there are respectively 50, 35 and 8 ResNet blocks for these models, E^{2}CM results in a memory overhead of 20.85 MB, 15.05 MB and 1.94 MB respectively. The models themselves respectively require 222 MB, 476 MB and 42.17 MB, so the fractional memory overhead is at most 9.4%. Similar to the computation overhead, the memory overhead of E^{2}CM increases linearly with the number of classes. Thus, for ResNet-152 on CIFAR-100, one will need $0.417 \times 10 = 4.17$ MB to store the class means for a layer on average, increasing the overhead to $9.4\% \times 10 = 94\%$. We show in the following that the aforementioned max pooling method can also alleviate memory requirements.

2.4.2 E^{2}CM Under a Fixed Training Time Budget: Fine Tuning

In this experiment, we consider a scenario where a model has to be trained on a low-power edge device. Also, the local training dataset is different from the dataset that the base model was trained on. A key motivation for this widely-used transfer learning setup is user privacy. We compare the early exit methods using EfficientNet-B0 and MobileNetV3Large as base models on ImageNet and Tiny ImageNet datasets. We randomly sample 50 and 10 images per class for

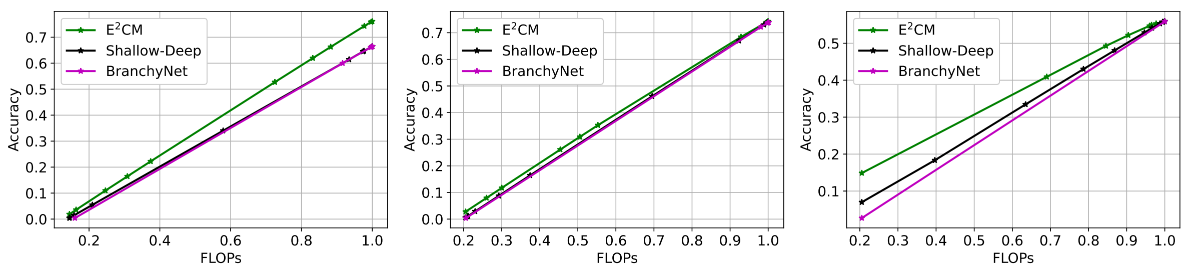


Figure 3. Comparison of E^{2}CM with existing methods under fixed training time budget of one epoch for the fine tuning task on ImageNet (left, middle) and Tiny ImageNet (rightt) datasets using EfficientNet-B0 (left), MobileNetV3Large (middle, right) models.

ImageNet and Tiny ImageNet and use the resulting subsets for fine tuning. For all methods, only one early exit location is used that corresponds to roughly 20% of the entire network in terms of FLOPs. To overcome the memory overhead of E^{2}CM, the output dimensions of the exit layer is downsampled from 14x14x80 to 7x7x10 via max pooling. We follow the same threshold selection procedure as described in 2.4.1.

As seen from Figure 3, E^{2}CM outperforms all competing methods under a fixed training time budget for the fine tuning task. This supports the hypothesis that E^{2}CM generalizes better to the dataset compared to other early exit techniques under a training time budget. This result is important because the networks on edge devices may have to train their own version of the base model for data/model privacy reasons. Moreover, E^{2}CM requires the fewest FLOPs for the same accuracy, which translates to reduced battery usage. Moreover, the cost of communication to the cloud may be very high.

Thanks to the max pooling technique, the overall computation overhead of E^{2}CM is only around 0.001 FLOPs on the ImageNet dataset for both EfficientNet-B0 and MobileNetV3Large. In terms of memory overhead, if MobileNetV3Large is used, E^{2}CM needs 1.87 MB and 0.87 MB to store the class means for ImageNet and Tiny ImageNet datasets respectively. For EfficientNet-B0 on ImageNet, the overhead is 1.87 MB. Considering MobileNetV3Large requires 21.5 MB and 17.2 MB of memory for ImageNet and Tiny ImageNet datasets, respectively, and EfficientNet-B0 needs 16.9 MB for ImageNet, the memory overhead is at most 11%. These results show that E^{2}CM does not have a large memory or computation footprint even for datasets with a large number of classes. We believe that the memory and computation costs can be further reduced by complementary methods such as quantization and pruning [42, 43].

2.4.3 E^{2}CM with Unlimited Training

In this experiment, we remove the training time budget. We combine E^{2}CM with Shallow-Deep Networks and compare this combination against Shallow-Deep Networks, the BWDS method, and BranchyNet.

We train the internal classifiers of our merger of E^{2}CM and Shallow-Deep for 100 epochs. The corresponding high computational complexity may not be desirable for low-power devices. We follow the same threshold selection procedure described above. During inference, the decision of early exit is made according to both the E^{2}CM and the ICs.

We train BranchyResNet-152 and BranchyWideResNet-101 for 300 and 250 epochs respectively. We use stochastic gradient descent with batch size of 256. The loss of the branches are added up to the loss of the final layer and the weighted average is taken as prescribed

TABLE II

COMPARISON OF EARLY EXIT METHODS

Model, Dataset	**Method**	**Accuracy**			
		ϕ=0.15	ϕ=0.20	ϕ=0.25	ϕ=0.30
ResNet-152, CIFAR-10	**E^{2}CM+SDN**	**82.4%**	**86.4%**	**88.5%**	**88.8%**
	SDN	80%	86%	88.4%	88.8%
	BWDS	82%	84.1%	86%	87.7%
	BranchyNet	80%	80%	80%	80%
ResNet-152, KMNIST	**E^{2}CM+SDN**	94.1%	**96.2%**	**96.8%**	**96.8%**
	SDN	83.9%	94%	96.4%	96.6%
	BWDS	**94.5%**	96%	96.5%	96.6%
	BranchyNet	86%	87.5%	88.2%	89.5%
WideResNet-101, CIFAR-10	**E^{2}CM+SDN**	84%	**88%**	**88.6%**	**88.8%**
	SDN	82%	88%	88.5%	88.8%
	BWDS	**85.8%**	86.5%	87.8%	88.2%
	BranchyNet	81.9%	82.5%	84.9%	85%
WideResNet-101, KMNIST	**E^{2}CM+SDN**	**94.1%**	**96.5%**	**97.1%**	**97.2%**
	SDN	84.1%	94%	96%	97%
	BWDS	93.8%	94.2%	95%	96%
	BranchyNet	89%	90.1%	90.2%	90.2%

in [11]. We have trained multiple combination of weights, and found that $[1/6, 1/4, 1]$ achieves the best performance. For thresholds, we follow the same procedure, but this time the range of values is $[0, \log K]$ where $K = 10$, because we consider the entropy of the predictions to make a decision [11].

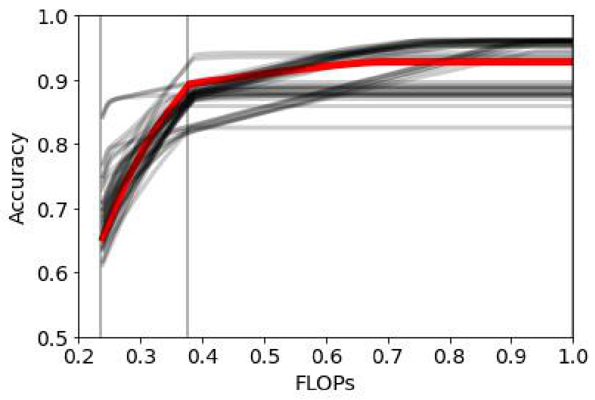
Since the task of training decision functions in the BWDS method is a binary classification task (i.e., early exit or not) they converge rather quickly. We separately train the classifiers that come after the decision functions as well. We use pooling and single linear layer for these, as suggested by the authors.

As shown in Table II, combining E^{2}CM with internal classifiers achieves a better trade-off between the computational cost and network accuracy. For low computational budget, E^{2}CM improves the accuracy by up to 6%. We also observe that BranchyNet suffers from training the network with the branches jointly. These branches hurt the overall performance. Moreover, convolutional layers in the branches add a significant computational cost without a considerable gain in accuracy. Also, although early exit mechanisms with decision functions like the BWDS method provide decent performance, experimental results show that threshold based early exit mechanisms perform better. In other words, making a decision about early exit and then performing classification fares worse than the threshold-based strategies where classification and exiting decisions are melted into the same pot.

According to Table II, we can conclude that the consulting mechanism between the E^{2}CM and the internal classifiers is generally beneficial. The common decision that is reached by the two classifiers can rectify possible misclassifications and avoid unnecessary computation. This can be seen as an example of ensembles, in which multiple classifiers are used to make a decision. Interestingly, our ensemble reduces the total computational cost unlike ordinary ensemble methods.

2.4.4 E^{2}CM for Unsupervised Learning

We follow the same experimental setup as in [25] for MNIST and Fashion-MNIST datasets. Namely, we use an encoder with 4 layers, which have 500, 500, 2000, 10 neurons, with a clustering layer (CL) at the end. Let this encoder and clustering layer be DEC_{large}. After pretraining DEC_{large}, we train the clustering layer as in [25]. Then, we create another encoder with the



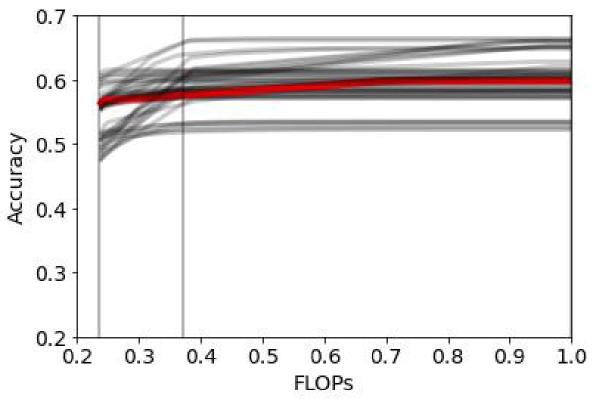


Figure 4. Accuracy-FLOPs curve on MNIST (left) and Fashion-MNIST (right) for unsupervised learning using E^{2}CM. Vertical lines indicate the individual FLOPs of DECs. Each black curve is for one experiment, and the red curve is the average of all experiments.

500-500-10-CL architecture, where the weights of first two layers are copied from DEC_{large} and frozen. We name this encoder DEC_{middle} and follow the same procedure as in DEC_{large}. Finally, we repeat the same procedure for DEC_{small}, which has the 500-10-CL architecture, where the weight of the first layer is frozen and copied from DEC_{large}.

After the training is complete, we take the intermediate outputs, i.e., the outputs of the layers with 10 neurons. Then, using the cluster centers from each clustering layer, we follow Algorithm 1. To measure the accuracy, we use the same technique described in [25].

During the experiments, we noticed that the process of pretraining affected the final result significantly. We have thus run multiple experiments, and the performance corresponding to each experiment is illustrated as one gray curve in Figure 4. The average of individual experiments is shown as the solid red curve. As can be observed in Figure 4, by adding early exits to the architecture, it is possible to save 60% of the computation while losing only 6% in unsupervised clustering accuracy on MNIST dataset. On Fashion-MNIST, the accuracy loss is 1%. Also, thresholds make it possible to adjust the model according to various computational needs.

2.5 Pruning Early Exit Networks

In this separate line of work, we combine two approaches that try to reduce the computational cost while keeping the model performance high: Pruning and early exit networks.

2.5.1 Introduction

Most early exit networks have two components: an off-the-shelf base network, and additional small classifiers [11, 12, 44]. Until now, no work has been done to prune early exit networks. This work aims to fill this gap. In particular, we evaluate the performance of two strategies. In the first strategy, we prune and train the additional internal classifiers that enable early exit jointly with the base network. In the second strategy, we first prune and train the base network, and then the classifiers. The significance of the second strategy is that its optimality implies one can separate the processes of early exit and pruning without loss of optimality.

2.5.2 Experiment

We compare the following two approaches using a ResNet-56 and CIFAR-10 [4, 16]:

1. Multiple linear layers are added to a ResNet-56 (the resulting network is a Shallow-Deep Network [12]) and the entire model is pruned.
2. A ResNet-56 is pruned, multiple linear layers are added to the resulting network and then these linear layers are pruned.

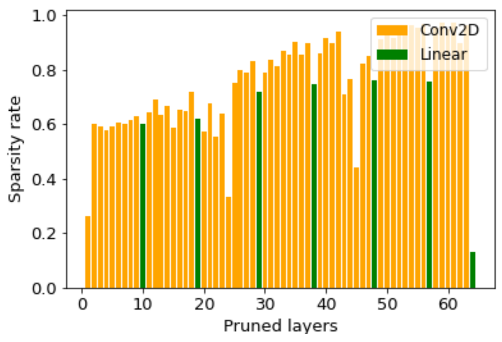
The number, locations, and the architectures of the additional linear layers are as described in [12]. For both approaches, pruning is followed by fine tuning. The procedure of pruning and fine tuning is repeated 20 times. At each pruning phase, 10% of the weights are pruned using global unstructured l_1 norm based pruning. At each fine tuning phase, the model is trained for 10 epochs. Accuracy vs. FLOPs graphs are obtained using different confidence thresholds and applying time sharing as in [12, 44].

2.5.3 Results

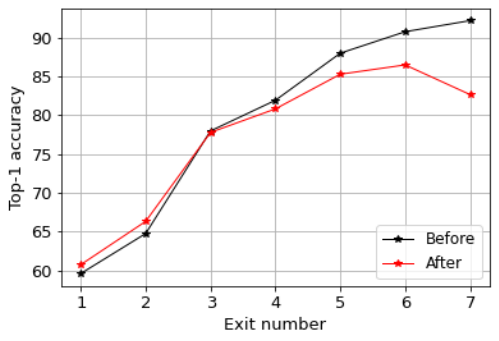
2.5.3.1 Approach 1

The sparsity rate of each layer's weights and the exit performances after 20 iterations of pruning and fine tuning are shown in Figure 5. We can make the following observations:

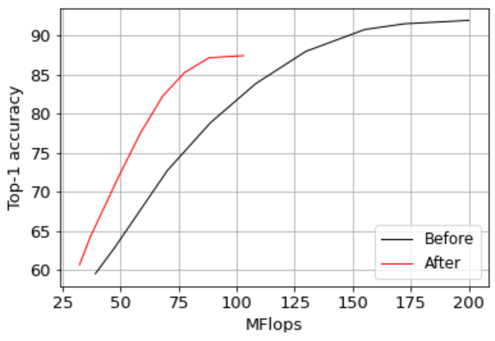
1. Deeper layers are pruned slightly more than earlier layers, and final exit is pruned much less than earlier exits as seen from Figure 5a.
2. Despite extensive pruning, exit performances at earlier exits are comparable with the unpruned baseline, and even better at the first and second exit points as seen from Figure 5b. This suggests pruning can improve model performance.
3. Performance at the last exit is most likely hurt by the joint training of all linear classifiers.
4. Pruning reduces the computational cost up to around 20% without loss of performance. Best accuracy obtained with pruning is around 4% less than the unpruned baseline, but the computational cost is reduced to half as seen from Figure 5c.



(a) Sparsity rates of the weights of each layer after 20 iterations of pruning and fine tuning.



(b) The exit performances after 20 iterations of pruning and fine tuning. Each point corresponds to the scenario where all samples exit from that exit location. Black curve represents the performance before any pruning.



(c) Accuracy-FLOPs plot. Samples exit from different exit points due to confidence thresholds. Black curve represents the performance before any pruning.

Figure 5. The sparsity rates and exit performances for Approach 1.

2.5.3.2 Approach 2

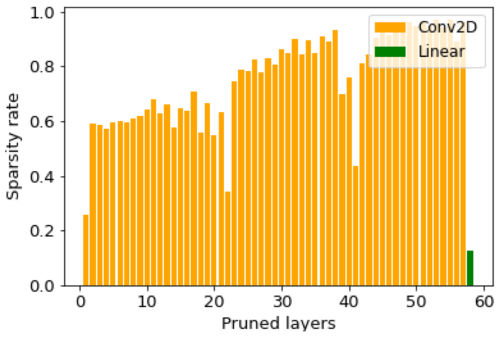
In this approach, the ResNet-56 is pruned and fine tuned first. Then, linear layers are added to the model and these layers are pruned and fine tuned. The sparsity rate of each layer's weights and the exit performances after 20 iterations of pruning and fine tuning are shown in Figure 6. We can make the following observations:

1. Figures 6a and 6b shows that although the final exit went through two pruning phases (one with base network, one with additional linear classifiers), it is not pruned much.
2. Except the last exit, deeper exits are pruned more than earlier exits as seen from Figures 6a and 6b.
3. Compared to Figure 5, linear layers become more sparse as seen from Figure 6b.
4. Unpruned baseline in Figure 6c has high accuracy only at the last exit because earlier exits are attached after the base network was pruned and fine tuned. The additional linear classifiers were not trained at this point.
5. Fine tuning linear layers pushes the computational cost up as seen in Figure 6d, which is interesting.

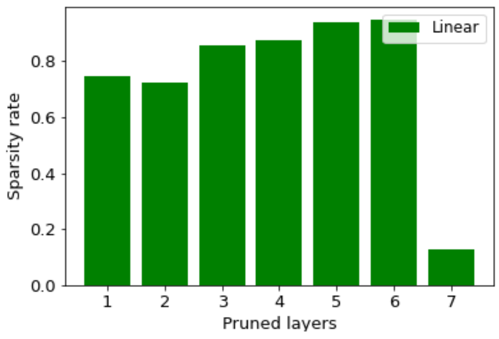
2.5.3.3 Comparison

The exit performances of the two approaches are shown in Figure 7. From Figure 7a, it can be seen that that pruning all layers at the same time performs better at earlier exit points compared to pruning the layers in an ordered fashion. However, ordered pruning approach performs better at deeper exit points which give the best performance.

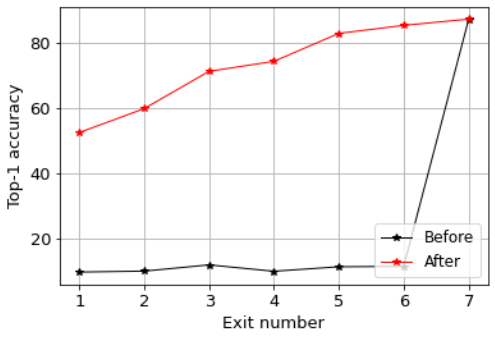
In terms of computational cost, pruning all layers at once reduces the computational cost by up to 15% compared to ordered pruning, but the performances are close at high accuracy rates as seen from Figure 7b. Both approaches are able to reduce the computational cost by half at the cost of 4% decrease in the accuracy.



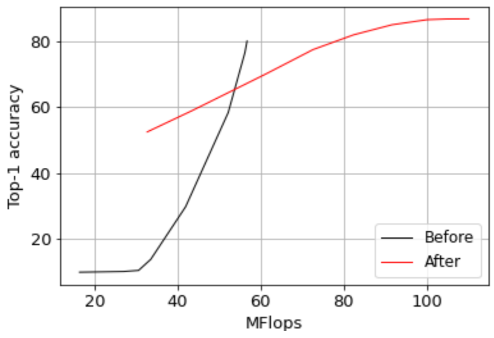
(a) Sparsity rates for the base network's layers after 20 iterations of pruning and fine tuning.



(b) Sparsity rates for the additional classifiers after 20 iterations of pruning and fine tuning.

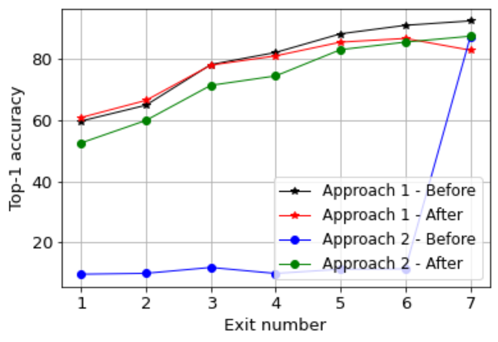


(c) The exit performances after 20 iterations of pruning and fine tuning. Each point corresponds to the scenario where all samples exit from that exit location. Black curve represents the performance before any pruning.

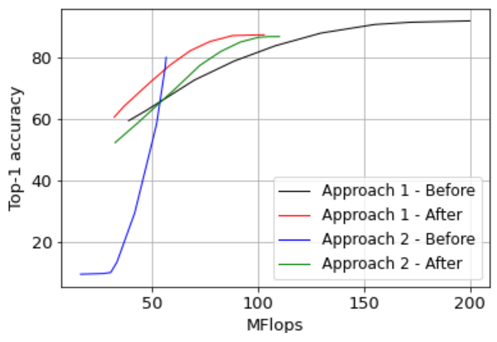


(d) Accuracy-FLOPs plot. Samples exit from different exit points due to confidence thresholds. Black curve represents the performance before any pruning.

Figure 6. The sparsity rates and exit performances for Approach 2.



(a) The exit performances of the approaches.



(b) Accuracy-FLOPs plots of the approaches.

Figure 7. Comparison of the approaches.

CHAPTER 3

CLASS BASED THRESHOLDING IN EARLY EXIT SEMANTIC SEGMENTATION NETWORKS

Part of this chapter has been pusblished as A. Görmez and E. Koyuncu, "Class Based Thresholding in Early Exit Semantic Segmentation Networks," in IEEE Signal Processing Letters, vol. 31, pp. 1184-1188, 2024.

Overview: We consider semantic segmentation of images using deep neural networks. To reduce the computational cost, we incorporate the idea of early exit, where different pixels can be classified earlier in different layers of the network. In this context, existing work utilizes a common threshold to determine the class confidences for early exit purposes. In this work, we propose Class Based Thresholding (CBT) for semantic segmentation. CBT assigns different threshold values to each class, so that the computation can be terminated sooner for pixels belonging to easy-to-predict classes. CBT does not require hyperparameter tuning; in fact, the threshold values are automatically determined by exploiting the naturally-occurring neural collapse phenomenon. We show the effectiveness of CBT on Cityscapes, ADE20K and COCO-Stuff-10K datasets using both convolutional neural networks and vision transformers. CBT can reduce the computational cost by up to 23% compared to the previous state-of-the-art early exit semantic segmentation models, while preserving the mean intersection over union (mIoU) performance.

Keywords: Semantic segmentation, early exit networks.

3.1 Introduction

As deep learning advances rapidly, new, larger models are frequently introduced, leading to improved performance [6, 45, 46]. However, larger models, while capable of learning complex patterns, come with higher inference costs. In the era of decentralized computing on edge devices (e.g., IoT), minimizing the inference cost of large models becomes crucial for deployment on resource-constrained devices [47, 48].

To reduce the inference cost without compromising performance, early exit networks are proposed [10, 11]. Early exit networks capitalize on the heterogeneity of real world data. Since not all data samples have the same "difficulty", "easy" data samples can be allowed to exit the model early to save computation [10–12, 44, 49]. Early exit networks have been studied in conjunction with network pruning [50, 51]. They also have close ties with the phenomenon of *neural collapse* [39, 44].

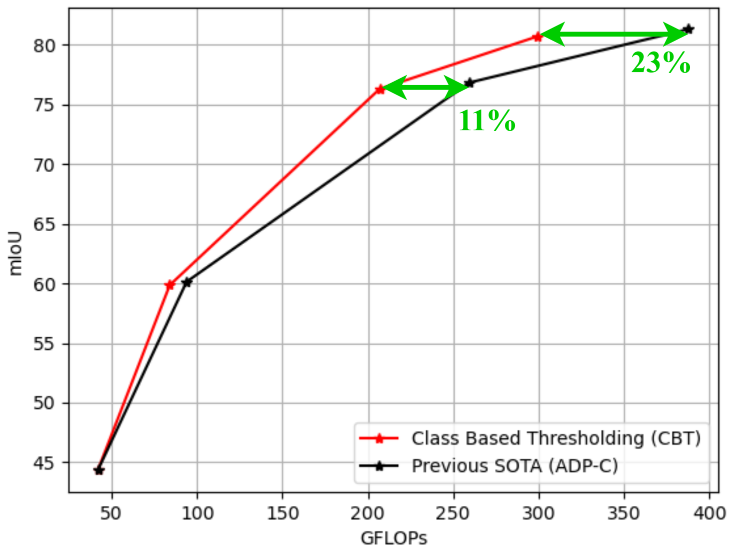


Figure 8. Comparison of CBT with the previous state-of-the-art on the Cityscapes dataset for HRNetV2-W48 model.

The neural collapse phenomenon states that as one travels deeper in a neural network, the intermediate representations become more disentangled, forming distinct clusters at the last layer, which makes classification easier [39]. Recent works expand on this phenomenon and show that clusters begin to form even at earlier layers [44, 52], resulting in a so-called *cascading collapse*. In the supervised setting, each cluster corresponds to a class where the model is trained on, and the mean of the cluster is referred to as simply a *class mean*.

In [44], the authors propose an early exit mechanism utilizing the neural collapse phenomenon, outperforming various existing schemes. Specifically, a representation that is sufficiently close to a class mean at any given layer can be allowed an early exit, without significant penalty in classification performance. However, the idea of using the nearest class mean decision rule is not immediately applicable to the task of semantic segmentation since one now needs to perform pixel-wise classification. In fact, in the image classification task, there is one input and it belongs to one class. Therefore, the representations of the images with the same label can be averaged, and class means can be calculated. The intermediate layer outputs will be close to only one class mean, and a meaningful prediction can be performed based on the distances to the class means [44]. On the contrary, in semantic segmentation, one input has many pixels, each of which belong to different classes. One image has one representation at each layer, and components of a representation corresponds to many pixels. Hence, class means cannot be immediately calculated from the representations for individual pixels for the task of semantic segmentation.

Having too many pixels in an image results in the curse of dimensionality, which presents an additional complication. In fact, even if the class means could be obtained, using the nearest class mean decision for the pixels would be too costly since there are thousands of pixels in an image. These make it infeasible to calculate the class means for the pixels using existing algorithms (e.g. [44]). Nevertheless, utilizing the neural collapse phenomenon for semantic segmentation would be particularly useful because the amount of computation can be reduced significantly for the state-of-the-art semantic segmentation models [53–61].

We propose "Class Based Thresholding (CBT)", a novel algorithm that reduces the computational cost while preserving the model performance for the semantic segmentation task. Leveraging the neural collapse phenomenon, CBT calculates the mean of the prediction probabilities of pixels in the training set, for each class. Then, the thresholds for each class are calculated via a simple transformation of the class means. These thresholds are then employed to allow the early termination of the computation for confidently predicted pixels at inference time. We show the effectiveness of CBT on the Cityscapes [62], ADE20K [63] and COCO-Stuff-10K [64] datasets using the HRNetV2-W18, HRNetV2-W48 and vision transformer models [65, 66]. By efficiently utilizing the neural collapse phenomenon, CBT can reduce the computational cost by up to 23% compared to the previous state-of-the-art method while preserving the model performance as shown in Figure 8.

3.2 Class Based Thresholding

We build on the state-of-the-art early exit semantic segmentation method, "Anytime Dense Prediction with Confidence Adaptivity (ADP-C)" [1]. ADP-C adds early exit layers to the base semantic segmentation model and introduces a masking mechanism based on a single user-specified threshold value t to reduce the computational cost. If a pixel is predicted confidently

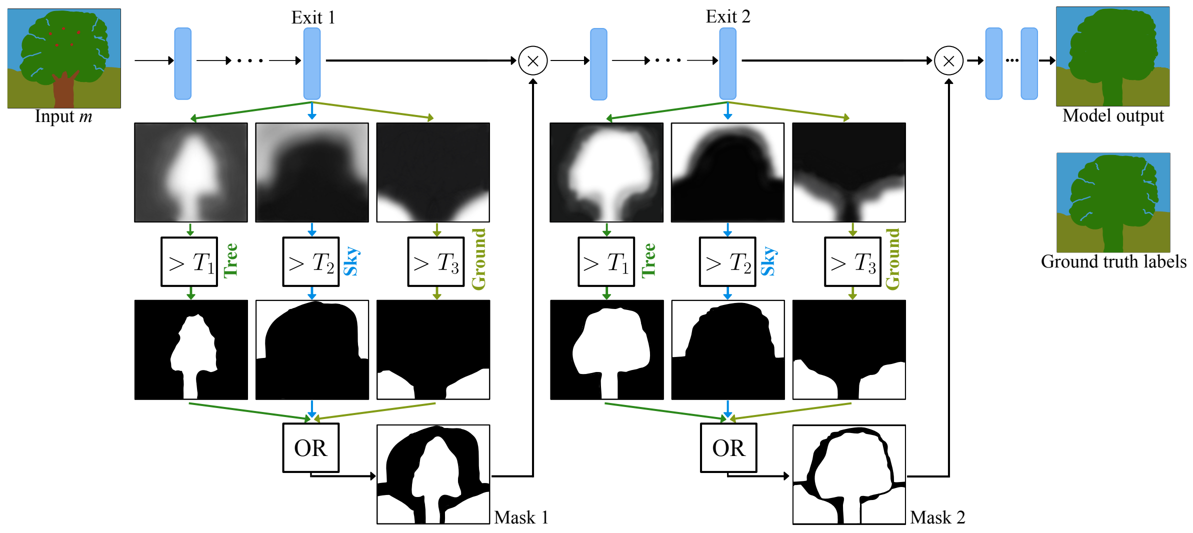


Figure 9. Overview of our Class Based Thresholding (CBT) scheme at the inference time for an example network with $N = 2$ exit layers and $K = 3$ classes: Tree, ground, and sky. In contrast to [1] and [2], CBT utilizes different thresholds for different classes, considering their varying levels of inherent difficulty. The thresholds are determined as a function of only two non-trainable hyperparameters, independent of the number of classes or exits, thanks to our neural collapse inspired design. At each exit, the layer output is split into $K = 3$ channels, where each channel corresponds to one of the tree, ground, or sky classes. The channels are then transformed into masks using their corresponding distinct thresholds, and the resulting masks are merged. The methods presented in [1] and [2] thus become a special case of CBT where the thresholds for every class is the same. Mask 1 illustrates the confident (white) pixels after the merger at Exit 1, which is subsequently integrated into the following layers through multiplication. This integration ensures that the model avoids unnecessary computations for these confident pixels in subsequent layers. Exit 2 follows the same mechanism for inference. Mask 2 exhibits a greater number of confident pixels due to the input image passing through layers between Exit 1 and Exit 2. The exit predictions become progressively better.

at an exit layer, i.e., the maximum prediction probability over all classes is greater than the threshold t, that pixel is masked for all subsequent layers. Any masked pixel will not be processed again at later layers. The computational cost is reduced due to the induced feature sparsity. While it is possible to let every pixel exit at the same time [67], this approach performs worse at the boundaries of objects, and therefore we focus on ADP-C and pixel-wise early exiting.

A big room for improvement for ADP-C stems from the observation that the same user-specified threshold value t is used for every class. However, it is more plausible that different threshold values should be used for different classes, and the threshold values should reflect the dataset and class properties, rather than just being a user-specified number. The observation from [68] supports our hypothesis: "The distribution of max logits of each predicted class is significantly different from each other." This is because pixels belonging to different classes have different difficulty levels of being predicted correctly. For example, using $t = 0.998$ for *bicycle* class as in ADP-C makes sense because we may want to be really certain about pixels belonging to bicycles. However, pixels belonging to the *sky* class will be often easier to predict than pixels belonging to the *bicycle* class, which means the model will be confident about them much sooner. Therefore, a lower threshold value can be used for the *sky* class without significant penalty in prediction accuracy. Otherwise, more computation will have to be performed for the *sky* pixels.

Given a model trained on a semantic segmentation task with K classes, we propose using different masking threshold values per class, based on the dataset and class properties. Let $T = [T_1 \cdots T_K] \in [0,1]^K$ be the threshold vector that we wish to determine, where the k^{th} element T_k corresponds to class k, and $k \in \{1, 2, \ldots, K\}$. Consider M training inputs, each of which have a height of H pixels, and a width of W pixels. Suppose that we utilize N exit layers in the model. The class prediction probabilities provided by the model at exit n for each (m, h, w) triplet can be represented by the function $\phi_n : \mathbb{R}^{M \times H \times W} \to [0, 1]^K$. Hence, given a pixel at height h, width w of input m, the prediction probabilities for the K classes at exit n are expressed as $\phi_n(m, h, w)$.

Let S_k denote the set of all pixels, or (m, h, w) triplets, whose ground truth is class k. At each exit layer n, for each class k in the training set, we calculate the mean of layer n's prediction probabilities using all training set pixels in S_k. This averaging helps obtaining a broad sense of information about the difficulty of pixels. This yields

$$p_{n,k} \triangleq \frac{1}{|S_k|} \sum_{(m,h,w) \in S_k} \phi_n(m, h, w) \in [0, 1]^K. \tag{3.1}$$

The motivation of averaging in (Equation 3.1) comes from the neural collapse phenomenon, which states feature vectors converge to their average class means as one goes deeper in a network [39]. Indeed, the averages $p_{n,k}$ should empirically be a good estimate for the class probabilities, especially for a deep layer index n. Specifically, the i^{th} element of $p_{n,k}$ denotes the average probability of a pixel belonging to class i when the ground truth for that pixel is class k. Next, we compute

$$P_k = \frac{1}{N} \sum_{n=1}^{N} p_{n,k} \in [0, 1]^K, \tag{3.2}$$

which is the average of $p_{n,k}$ over all layers. Hence, information across layers is shared to obtain a global estimate P_k on the difficulty of classes. The logic for the information sharing across layers is to leverage insights from both shallow and deep layers, and to make the thresholding less complex due to having only one set of thresholds for every exit. Information sharing in CBT can be seen as a naive version of feature reuse in other multi-exit network settings such as [30, 31, 69–71].

We then translate the estimates to classification thresholds as follows. We initialize the threshold T_k to be the difference between the largest and the second largest elements of P_k, because this initialization strategy has been shown to be a reliable confidence score, effectively capturing the importance of the most dominant element relative to the second-largest one [12, 44]. If the confidence score is high, then the masking threshold should be low so that the computation can terminate easily. After all components of T are initialized in this manner, we inversely scale T according to two non-trainable parameters α and β so that the maximum and minimum class confidence scores determined by T will be converted to masking threshold values α and β respectively, where $\alpha < \beta$. The rationale behind this inverse scaling is to guarantee that classes with high confidence scores will have low thresholds and vice versa. Specifically, the scaling is done via

$$T_k \leftarrow \left(1 - \frac{T_k - \min T}{\max T - \min T}\right)(\beta - \alpha) + \alpha. \tag{3.3}$$

The inference is performed as follows: Let $\pi \in [0,1]^K$ be the prediction probabilities for a pixel at an exit layer. Let $j = \arg\max \pi$. If $\pi_j > T_j$, this pixel will be marked as confidently predicted (predicted as class j) and will be incorporated to the mask M as in Figure 9. By doing

TABLE III

RESULTS ON CITYSCAPES.

Method	**Model**	**Exit**							
		1		**2**		**3**		**4**	
		mIoU	**GFLOPs**	**mIoU**	**GFLOPs**	**mIoU**	**GFLOPs**	**mIoU**	**GFLOPs**
MDEQ [72]	S	17.3	521.6	38.7	717.9	65.6	914.2	72.4	1110.5
ADP-C	HRNetV2-W48	44.34	41.92	60.13	93.90	76.82	259.33	68.55	387.80
CBT [0.99, 0.998]		44.34	41.92	59.85	84.02	76.29	206.89	80.69	299.10
CBT-ns [0.99, 0.998]		44.34	41.92	59.82	84.00	76.28	206.88	80.74	299.17
CBT [0.95, 0.998]		44.34	41.92	57.97	71.57	72.86	155.77	76.60	222.65
CBT [0.9, 0.998]		44.34	41.92	56.05	65.91	68.92	132.49	72.29	186.31
ADP-C $\beta = 0.9$		44.34	41.92	54.86	53.27	67.10	118.25	69.31	157.48
ADP-C	HRNetV2-W18	40.83	23.68	48.19	33.27	68.26	45.40	77.02	58.90
CBT [0.99, 0.998]		40.83	23.68	48.07	31.74	67.98	41.40	76.57	51.26
CBT [0.95, 0.998]		40.83	23.68	46.97	29.51	64.88	36.25	71.18	43.35
CBT [0.9, 0.998]		40.83	23.68	45.79	28.39	61.32	33.72	67.45	39.42

so, the outputs of subsequent layers at these locations will not be calculated. Instead, already computed values will be used. Note that once calculated, T is not updated in inference.

3.3 Experiments and Results

We compare CBT against ADP-C [1] and DToP [2]. ADP-C and DToP allow early prediction of pixels, but they use the same thresholds for all classes. We use Cityscapes, ADE20K, COCO-Stuff-10K datasets [62–64], and HRNetV2-W18, HRNetV2-W48, ViT models for evaluation [65, 66]. We use mean intersection over union (mIoU) as our performance metric and number of floating point operations (FLOPs) as our computational cost metric. We attach 3 early exit

TABLE IV

RESULTS ON ADE20K.

Method	Model	Exit							
		1		2		3		4	
		mIoU	**GFLOPs**	**mIoU**	**GFLOPs**	**mIoU**	**GFLOPs**	**mIoU**	**GFLOPs**
ADP-C	HRNetV2-W48	4.12	6.20	5.16	15.42	12.15	52.47	42.82	100.28
CBT [0.9, 0.998]		4.12	6.20	5.15	15.07	12.09	50.48	41.85	94.31
CBT-ns [0.9, 0.998]		4.12	6.20	5.15	15.06	12.08	50.48	41.87	94.34
CBT [0.8, 0.998]		4.12	6.20	5.14	14.80	11.90	48.81	40.17	90.25
CBT [0.7, 0.998]		4.12	6.20	5.12	14.55	11.58	47.27	37.54	86.52
ADP-C	HRNetV2-W18	4.89	5.88	6.83	7.84	8.94	12.73	9.74	19.04
CBT [0.9, 0.998]		4.89	5.88	6.80	7.73	10.07	12.24	11.78	17.89
CBT [0.8, 0.998]		4.89	5.88	6.75	7.67	10.17	11.98	11.95	17.26
CBT [0.7, 0.998]		4.89	5.88	6.70	7.62	10.09	11.75	11.88	16.71

layers to HRNet models as in [1] and 2 to ViT models as in [2] with the same exit structures and positions. The training is done by using the weighted sum of the exit losses. We assign the same weight of 1 to exit losses.

We have evaluated CBT with numerous α-β pairs (denoted as CBT $[\alpha, \beta]$). We kept $\beta = 0.998$ for HRNet models, $\beta = 0.9$ for ViT-Base, and $\beta = 0.95$ for ViT-Large in our experiments for a fair comparison because ADP-C and DToP achieve the best performance with these values. For comparison purposes, we also included $\beta = 0.9$ for ADP-C in Table III. Naturally, it has the lowest mIoU and GFLOPs because all classes use the same threshold of 0.9, the lowest among

TABLE V

COMPARISON OF CBT AGAINST DYNAMIC TOKEN PRUNING (DTOP).

Method	**Dataset**	**Model**	**Exit**					
			1		**2**		**3**	
			mIoU	**GFLOPs**	**mIoU**	**GFLOPs**	**mIoU**	**GFLOPs**
DToP	ADE20K	ViT-Base	41.79	55.70	45.85	66.60	49.21	83.52
CBT [0.85, 0.9]			41.79	55.70	45.52	65.60	49.04	80.80
DToP		ViT-Large	37.86	208.96	47.97	352.32	52.18	452.3
CBT [0.9, 0.95]			37.86	208.96	47.82	336.01	51.69	421.93
DToP	COCOStuff10K		31.89	124.94	41.71	205.14	45.64	266.17
CBT [0.9, 0.95]			31.89	124.94	41.09	197.53	45.29	252.04

the experiment settings. As shown in Table III, Table IV and Table V, lower α facilitates pixels exiting early, and increasing it results in more confident predictions.

By Table III, CBT $[0.99, 0.998]$ decreases the computational cost by 23% while losing only 0.62 mIoU for HRNetV2-W48. For Exits 2 and 3, the computational cost is decreased by 10% and 20% respectively. By using smaller α, the computational cost can be decreased more, but mIoU starts degrading as well. Note that Exit 4 of CBT $[0.95, 0.998]$ can match the performance of Exit 3 of ADP-C while using 14% less computation. For HRNetV2-W18, the results follow the same trend: CBT $[0.99, 0.998]$ decreases the computational cost by 9% and 13% for exits 3 and 4 respectively as seen in Figure 10.

According to Table IV, we observe that CBT can also reduce the computational cost on the ADE20K dataset, which has significantly more classes as compared to the Cityscapes datasets. Specifically, CBT $[0.90, 0.998]$ decreases the computational cost by 6% while losing only 0.97 mIoU for HRNetV2-W48. The reason why the performances at the first three exit is low for both ADP-C and CBT is because the model cannot perform well enough due to large number of classes. It needs significantly more computation (e.g. 94.31 GFLOPs instead of 15.07, also seen in Table V with ViT) to have better performance. Also, this is why CBT cannot reduce the computational cost on ADE20K as much as it does on Cityscapes with HRNet models.

In Figure 11, we illustrate CBT-calculated class thresholds for Cityscapes and ADE20K datasets. Due to the ADE20K dataset's large number of classes, only the 19 classes with the lowest thresholds are displayed for both datasets. For Cityscapes, with a total of 19 classes, we exhibit all class thresholds. Compared to ADE20K dataset, class thresholds are spread out more uniformly between $\alpha = 0.9$ and $\beta = 0.998$) for Cityscapes dataset ($\sigma = 0.033$). For ADE20K dataset on the other hand ($\sigma = 0.009$), the behavior is different: Most class thresholds lie between 0.997 and 0.998. This supports our observation that CBT can reduce the computational cost more when the number of classes is relatively low. We can also observe that for both datasets, simple classes such as "sky" have low thresholds, while more complex classes have typically have higher thresholds values, as expected.

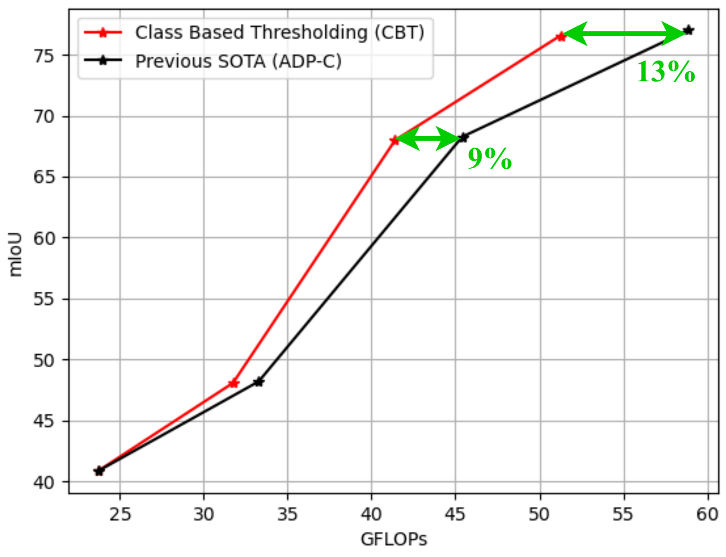


Figure 10. Comparison of CBT with the previous state-of-the-art on the Cityscapes dataset for HRNetV2-W18 model.

Figure 12 shows the relationship between different thresholds and their corresponding class-wise mIoU performances. When a lower α is used, class-wise mIoU performances drop slightly, in line with the results in Table III, Table IV and Table V. The "sidewalk" and "car" classes are affected the most with the change of their thresholds. For easier, high-mIoU classes, the performance drop is not drastic, suggesting the effectiveness of CBT.

3.3.1 Ablation Study

CBT calculates the average of $p_{n,k}$ over all exits as in (Equation 3.2) to obtain one single vector P_k, which is later scaled to obtain the thresholds. This allows the information across the layers to be shared and reduces the number of total thresholds from $N \times K$ to K. Here,

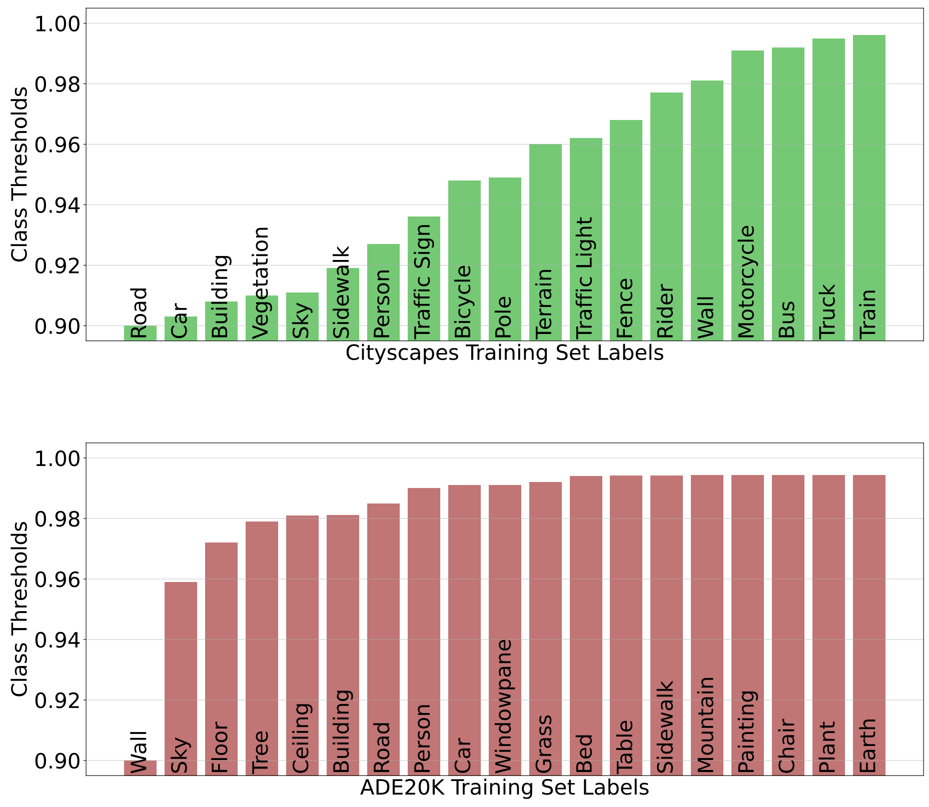


Figure 11. CBT thresholds for Cityscapes and ADE20K ($\alpha = 0.9, \beta = 0.998$).

we disable the information sharing by not averaging and allowing each exit to have its own thresholds based on its $p_{n,k}$. This prevents information flow from deeper exits to shallower exits. Note that this is a more complex method due to having more thresholds. We include the results in Table III and Table IV only for the highest α values and denote by *CBT-ns.* As seen from the numbers, there is no significant difference between CBT and CBT-ns, meaning the more complex CBT-ns is not superior to CBT.

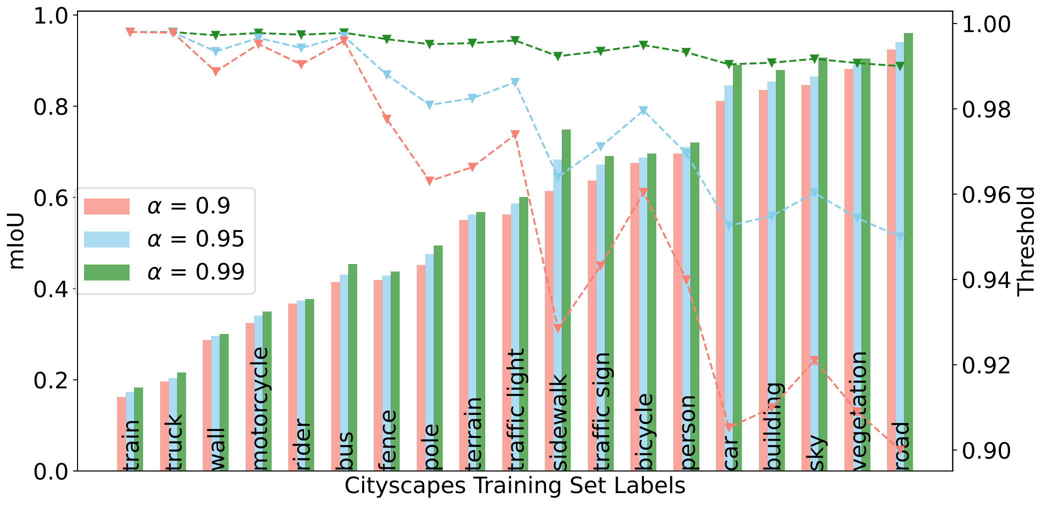


Figure 12. Class-wise mIoU performances for HRNetV2-W48 at Exit 2 with various α values ($\beta = 0.998$). Dashed lines indicate the corresponding class thresholds.

Finally, we note that the ideas developed in this paper can be applied to multi-modal data, which inherently have different requirements for processing complexity [73–76]. In particular, the thresholds used for a neural network classifying text data should be different than the thresholds used for the image data.

CHAPTER 4

DATASET PRUNING USING EARLY EXIT NETWORKS

Overview: We present EEPrune, a novel dataset pruning algorithm that leverages early exit networks during training. EEPrune utilizes the innate ability of early exit networks to assess the difficulty of individual samples and avoid overthinking. It applies multiple criteria to decide whether to prune them. Specifically, for a training sample to be discarded, the confidence level of the model at the early exit should be above a certain threshold, along with a correct classification at both the early exit and final layers. Extensive experiments and ablations on CIFAR-10, CIFAR-100, Tiny Imagenet, KMNIST and ImageNet datasets demonstrate that EEPrune consistently outperforms other dataset pruning methods.

Keywords: Dataset pruning, early exit networks.

4.1 Introduction

Sutton's "bitter lesson" [45, 77] states that "general methods that leverage computation are ultimately the most effective, and by a large margin." Larger models trained on larger datasets with more compute seem to be confirming this lesson due to major outcomes and great performances [6, 46, 78–82]. However, creating larger models and training them on larger datasets for longer, along with the expenses associated with retraining and hyperparameter tuning only add to the already high training costs. These ever-increasing costs may not be sustainable in the long run and are thus a significant problem that need to be addressed.

Dataset pruning methods have emerged as a potential solution to address the high costs of training deep learning models. By discarding redundant samples and keeping, for example, only the difficult samples for training, the model performance on the test set stays the same even when the model is trained on the smaller, pruned dataset from scratch [83–87]. While dataset pruning methods strive to decrease training expenses, they typically train an ensemble of models or train a single model fully to identify which samples can be discarded from the dataset [83–85]. This is particularly undesirable for resource-constrained devices due to the large memory footprint of the ensemble and high computational demands of full training [8, 88].

A different class of techniques to reduce computational cost is conditional computation through early exit [10–12, 30, 31, 44, 89]. Early exit networks allow one to exploit the heterogeneous nature of real-world data, which consists of "easy" samples that need less computation than "hard" samples to be classified correctly [9, 12, 44]. Specifically, an early exit network introduces multiple intermediate classifiers as exit points to an ordinary, base neural network. An easy input that is confidently classified at an intermediate exit point may then exit early without the need for traversing the remaining layers, thereby reducing the overall computation required for inference. This also prevents *overthinking*, as it has been shown that computations in deeper layers can change the classification outcome of the early exit [12]. Early exit networks can also provide reduced training costs by allowing the end user to train only the attached exits while keeping the base model frozen, which is particularly useful for edge devices [90, 91]. Earlier works thus demonstrate that utilizing not only the final outputs but also the intermediate representations can greatly reduce the overall inference or training costs of a neural network. However, to the best of our knowledge, this observation has not yet been exploited for dataset pruning, which becomes the focus of the present work.

In this work, we propose a new dataset pruning method named Early Exit Prune (EEPrune). EEPrune incorporates intermediate features in the data pruning decision process via early exit networks. By utilizing early exit networks to avoid overthinking, EEPrune identifies which samples are easy and can be discarded from the dataset. Specifically, if a data sample **1)** can be classified correctly at the early exit layer, **2)** can be classified correctly at the final exit layer and **3)** the early exit layer is confident enough with its classification, that data sample is deemed redundant and pruned. Through rigorous experiments, we show that EEPrune greatly improves the training performance, measured in terms of the floating point operations (FLOPs) to reach a certain level of accuracy. Our contributions are as follows:

- We use early exit networks, an inference time reduction technique, for the task of dataset pruning, a training time reduction technique. This innovative approach has not been explored in the literature before.
- We introduce a novel dataset pruning algorithm capable of maintaining baseline accuracy and occasionally surpassing it, while consuming significantly less energy compared to other dataset pruning methods.
- We shed light on the impact of exit location on the performance of identifying easy samples.
- We conduct an extensive and rigorous evaluation of dataset pruning methods and assess their ability to maintain a balanced representation across different classes during the pruning process.

4.2 Related Work

Our research integrates two methods that are commonly employed to lower the inference and training costs of deep learning models: early exit networks and dataset pruning.

4.2.1 Early Exit Networks

The first idea of adding early exits to neural networks goes back to the GoogleNet [3]. The primary purpose of adding the early exits was to solve the vanishing gradient problem. To reduce the inference cost, researchers have incorporated an exit criterion based on the entropy or the confidence level of the output of the early exit layer [10–12, 30, 31, 44, 89, 92]. Samples that meet the exit criterion are considered easy and prediction is made at the early exit layer, resulting in a faster computation. More complex samples that do not meet the exit criterion are forwarded to the next layers for further processing. Early exit layers can be trained jointly with the base model or separately [90], but approaches that do not require any training also exist [44]. These make early exit networks suitable for performing training and inference for localized learning [91]. Work on the fundamental limits of general conditional computation architectures include [49].

4.2.2 Dataset Pruning

Dataset pruning methods aim to select the most representative samples from a dataset and remove the rest in order to reduce the training cost [83–87, 93, 94]. This process can be accomplished through the use of scoring techniques, which evaluate the importance or difficulty of each sample in the dataset.

Two other areas related to dataset pruning are active learning and dataset distillation. Active learning involves selecting the most informative data to label from a pool of unlabeled data, with the aim of maximizing the model's accuracy [95, 96]. This approach differs from dataset pruning, which discards some labeled data once and for all to keep the size of the training set small and training cost low. In contrast, active learning repeats the process of selecting unlabeled data and training the model on it. Dataset distillation also aims to reduce the size of the dataset but does so by creating a smaller and often more difficult to interpret dataset from a larger, more human-understandable dataset [97]. In a different direction, the question of how to optimally prune the weights of an early-exit network has been studied in [50] with the objective of reducing the inference costs.

4.3 Problem Statement

We focus on the task of image classification in this work, but our idea is applicable to any supervised learning task. Our goal is to reduce the training cost of a neural network by maximizing the amount of data we prune from the training set, while keeping the test set performance intact.

Let $(x_0^{(i)}, y^{(i)}) \in D_{tr}$ be a training sample-label pair from the training set D_{tr} where there are N_{tr} samples and C classes, $i \in \{1, 2, \ldots, N_{tr}\}$ and $y^{(i)}$ is one-hot-encoded vector of length C. Let D_{te} be the test set in the same manner. Let $D_p \subseteq D_{tr}$ be the subset we want to prune to reduce the size of our dataset. Also, let $\theta_{D_{tr}}$ be the parameters of the network trained on D_{tr} and $A(\theta_{D_{tr}}, D_{te})$ be the accuracy of this network on test set, which we call base performance. We can formulate the optimization problem as

$$\begin{aligned} \max \quad & |D_p| \\ \text{s.t.} \quad & D_p \subseteq D_{tr}, \\ & D_r = D_{tr} \setminus D_p, \\ & |A(\theta_{D_{tr}}, D_{te}) - A(\theta_{D_r}, D_{te})| \leq \epsilon, \\ & \epsilon \geq 0. \end{aligned} \tag{4.1}$$

In the above formulation, ϵ is a small value controlling the deviation from the base performance.

While it holds true that training on a larger set of samples increases the generalization ability of the neural network, we emphasize the following two insights: **1)** Not all samples share the same level of complexity, nor do they contribute equally to the network's learning process [9, 12, 44, 83, 86]. In some cases, certain samples are straightforward and can be accurately classified after minimal computation, subsequently offering limited further learning value. On the other hand, some samples are more *difficult*, and the network can keep learning from them over an extended period. **2)** In dataset pruning we are interested in the amount of information the model can *still* extract from the data after some training. If the model was able to classify the training data with a very *high confidence* in a *short time*, this means the data cannot contribute more to the model. Such data samples yield gradients with negligible magnitudes, and thus their contributions to the model updates are insignificant. Following these insights, we propose a novel method, EEPrune, that uses early exit networks to identify and discard easy samples in order to reduce the size of the training set.

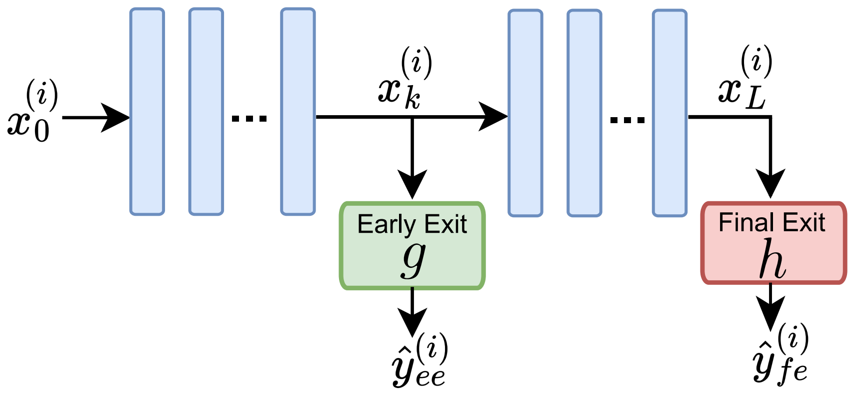


Figure 13. The computation graph of an early exit network used by EEPrune.

4.4 The EEPrune Method

We now introduce the EEPrune method. Let F be an early exit network with one early exit, as shown in Figure 13. The input-output relationships for the network can be expressed as

$$\begin{aligned} x_j^{(i)} &= l_j(x_{j-1}^{(i)}), j = 1, \dots, L, \\ \hat{y}_{ee}^{(i)} &= g(x_k^{(i)}), \quad \hat{y}_{fe}^{(i)} = h(x_L^{(i)}), \end{aligned} \tag{4.2}$$

where l_j are layers of the base model, k is the early exit point, g is the linear classifier at the early exit, $\hat{y}_{ee}^{(i)}$ is the early exit output, h is the linear classifier at the final exit, and $\hat{y}_{fe}^{(i)}$ is the final exit output.

Our ultimate goal is to reduce the training cost while keeping the model's performance on the test set as high as possible. We achieve this by reducing the size of the training set. More specifically, while training our early exit network on the dataset for a short time, we discard training samples from our dataset if **1)** the early exit can predict the correct class correctly, **2)** and the final exit can predict the correct class correctly, **3)** and the maximum prediction probability given by the early exit is greater than a specified threshold. In practice, we flag samples for pruning one by one over multiple epochs of learning. After E epochs, where E is a user-defined parameter, the learning process is stopped and all flagged samples can be removed from the dataset. We can then train a new model over the pruned dataset.

As can be seen, EEPrune verifies three conditions before flagging an input sample for pruning. The first condition ensures that the training sample of interest is easy enough to be correctly classified by using the features of a layer at the middle (or even an earlier point) of the network. This condition uses the easiness notion from the neural collapse phenomenon [39, 44, 52] and the early exit networks [10–12, 44]. The neural collapse phenomenon, and more specifically the cascading collapse phenomenon states that the intermediate representations of the samples at each layer form clusters and the clusters get separated from each other more as we move deeper in the network [39, 52]. The idea of early exit networks is simply some samples are easier to predict and they can be predicted correctly at earlier layers.

The second condition appears to be obvious because when there is no computational budget, the final exit gives the best prediction. However, in some cases *overthinking* can occur, which means that although the early exit can give a correct prediction, the computations after that change the output and the final exit gives an incorrect prediction [12]. With the second condition, we guarantee that the samples which are prone to overthinking are not discarded from the training set. This condition is also important because studies show that adversarial attacks can

Algorithm 2 EEPrune

```
Input: Data $D_{tr}$, early exit network $F$ with parameters $\theta$, threshold $t$, epoch $E$
$D_p \leftarrow \{\}$
for $epoch = 1$ to $E$ do
    for $({x_0}^{(i)}, y^{(i)}) \in D_{tr}$ do
        $\hat{y}_{ee}^{(i)}, \hat{y}_{fe}^{(i)} = F({x_0}^{(i)})$
        $J = -\sum_{c=1}^{C} {y_c}^{(i)} \log(\hat{y}_{ee_c}^{(i)}) - \sum_{c=1}^{C} {y_c}^{(i)} \log(\hat{y}_{fe_c}^{(i)})$
        $\theta \leftarrow \theta - \gamma \frac{\partial J}{\partial \theta}$
        if $\arg\max \hat{y}_{ee}^{(i)} = \arg\max y^{(i)}$ and $\arg\max \hat{y}_{fe}^{(i)} = \arg\max y^{(i)}$ and $\max \hat{y}_{ee}^{(i)} > t$ then
            $D_p \leftarrow D_p \cup \{{x_0}^{(i)}\}$
        end if
    end for
end for
Train the model on $D_{tr} \setminus D_p$ as desired
```

make the network choose the final exit most of the time although the sample is easy, which leads to redundant computation [12]. Hence, this condition also serves as protection against adversarial attacks.

The third condition effectively determines the rate of pruning. Depending on the threshold, if the early exit model is sure enough about its prediction, that sample is pruned. Our full algorithm can be seen in Algorithm 2.

To better understand the effectiveness of EEPrune, we apply it to a toy dataset and analyze which samples are pruned. The dataset is shown in Figure 14. There are 5 classes with 1000 samples each. The data is two dimensional. We use a simple 3 layer feed forward neural network with 10, 10, and 5 neurons at each layer respectively. To make it an early exit network, we add a linear layer with 5 neurons after the first layer. We use $t = 0.7$ as our threshold and run

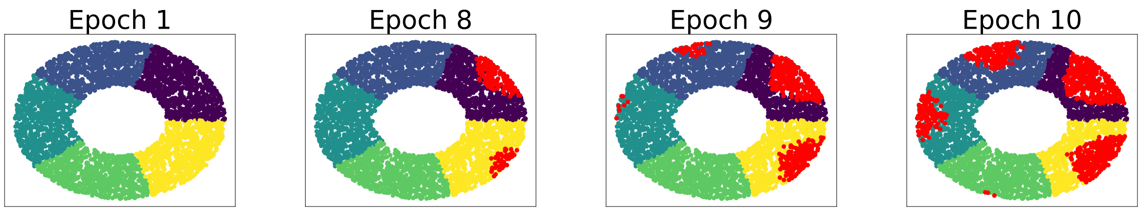


Figure 14. EEPrune discards samples (red) that are furthest away from the decision boundaries.

EEPrune for $E = 10$ epochs. We show the pruned samples at certain epochs in Figure 14 with red. It can be seen that the pruned samples are the ones which are farthest away from any decision boundary. As the model is trained more, more data is pruned but the pruning pattern stays the same. When the model is trained on the remaining samples, it still performs the same without loss in accuracy. The behavior of EEPrune resembles that of support vector machines (SVMs): Note that in the latter, removing the non-support vectors does not change the decision boundaries. At least, for the example in Figure 14, EEPrune correctly identifies the "non-support vector" that are irrelevant for the class decision boundaries and successfully removes them from the training process.

4.5 Experiments and Results

In this section, we describe in detail the experiments we conducted to measure the effectiveness of EEPrune, and we provide the numerical results of the experiments. In our experiments, we aimed to answer the following questions:

1. Is it possible for models trained on pruned datasets to achieve the same accuracy as models trained on unpruned datasets?
2. How does a particular dataset pruning method affect the final model accuracy?
3. How much computational resource do dataset pruning methods use?
4. How does the location of the early exit point affect the pruning threshold t?
5. How does placing the early exit point at a different location affect the final performance?
6. Does any dataset pruning method favor pruning a particular class?
7. How does modifying EEPrune criteria affect the performance?

4.5.1 Datasets and Models

In our experiments, we used the following five publicly available image classification datasets: CIFAR-10, CIFAR-100, Tiny ImageNet, KMNIST and ImageNet [16–19].

CIFAR-10 [16] consists of 60000 32×32 RGB images of 10 common vehicles and animals. Each class has equal number of samples. The training set contains 50000 images and the test set contains 10000 images.

CIFAR-100 [16] is a more challenging dataset than CIFAR-10 as it consists of 60000 32×32 RGB images of 100 classes. Like CIFAR-10, each class has equal number of samples and the training set contains 50000 images while the test set contains 10000 images.

Tiny ImageNet [18] is a dataset even more challenging than CIFAR-100. The training set consists of 100000 RGB images, and there are 200 classes. The images are larger in size (64×64)

and the classes are more diverse. There are 10000 validation and 10000 test images. Each class has equal number of samples.

KMNIST [19] consists of 70000 28×28 gray-scale images of 10 Japanese characters. Each class has equal number of samples. The training set contains 60000 images and the test set contains 10000 images.

ImageNet [17] is one of the most widely used large-scale datasets for the image classification task. The training set consists of 1281167 RGB images coming from a wide variety of sources, and there are 1000 classes. There are 50000 validation and 100000 test images. The images vary greatly in quality, resolution, lightning and each class has a variable number of samples.

We used the following three widely-used and highly-regarded models to benchmark dataset pruning methods in our experiments: EfficientNetV2-M [98], MobileNetV3-large [23] and ResNet-50 [4].

EfficientNetV2-M is a convolutional neural network from the improved EfficientNet model family [24]. It has around 54 million parameters, and it is specifically designed to offer faster training and better parameter efficiency, making it a good subject to measure effectiveness of dataset pruning methods [24, 98].

MobileNetV3-large is an efficient convolutional neural network with around 5.4 million parameters, designed specifically for mobile devices [23]. We selected it for evaluating dataset pruning methods due to its exceptional parameter efficiency and remarkable performance on various visual tasks.

ResNet-50 is a 50 layer convolutional neural network that achieves strong performance on numerous visual tasks [4]. It has around 25 million parameters. Since it is a widely used and well established benchmarking model, we opted to include it in our experiments.

4.5.2 Baseline Methods

We compare EEPrune against the following baselines:

No pruning: The given model is trained on the given dataset fully without any data pruning.

Random pruning: A subset of samples is randomly selected and pruned. The model is then trained on the remaining dataset.

Error-L2-Norm (EL2N) [83]: The EL2N method creates an ensemble of models and trains them for a period of time. The ℓ_2 error vectors (i.e., prediction minus the ground truth label vector) from each model are then averaged, and the samples with the lowest ℓ_2 error norm are discarded. By discarding the samples with the lowest error norm, this method ensures that the most critical samples are retained for further training. In our experiments, we used an ensemble of 10 models with the same architecture as our main model trained on the pruned dataset. Each model in the ensemble was trained for 15 epochs, following the suggestion in [83].

Forgetting [84]: The Forgetting method tracks the number of times each training sample is first predicted correctly but later incorrectly throughout an entire training session. In our experiments, we trained the model on the full dataset for 200 epochs and tracked the amount of forgetting events for each sample. We then sorted the samples based on the amount of forgetting events and pruned the ones that have been forgotten the least.

Selection via proxy (SVP) [85]: The SVP method is similar to the forgetting method, but uses smaller models and shorter training. In our experiments, we followed the same approach as forgetting, but used EfficientNetV2-S, MobileNetV3-small and ResNet-18 models instead of EfficientNetV2-M, MobileNetV3-large and ResNet-50 models respectively. We also trained the model for 50 epochs instead of the full 200 epochs.

Complexity gap: In [86], the authors proposed a novel training-free data scoring method called the *complexity gap score*. This approach involves measuring the difference between the data complexity when a sample is removed from the training set. The data complexity measure is defined as the extent to which a data sample contributes to the movement of network parameters during training [86]. In our experiments we used the source code provided by [86] to calculate the complexity gap score for each dataset and we pruned the lowest scoring samples.

TABLE VI

SUMMARY OF THE EXPERIMENTS.

Experiment axis	**Choices**
Methods	EEPrune, No pruning, Random, EL2N, Forgetting, SVP, Complexity gap
Datasets	CIFAR-10, CIFAR-100, Tiny ImageNet, KMNIST, ImageNet
Models	EfficientNetV2-M, MobileNetV3-large, ResNet-50
Pruning ratios	10%, 20%, 30%, 40%, 50%, 60%
Metrics	Top-1 accuracy, cumulative number of samples seen
Number of repeats	3

4.5.3 Experiment Settings

We conducted six experiments for each method-model-dataset combination to prune 10%, 20%, 30%, 40%, 50% and 60% of the training set. After each dataset pruning phase, the models are reinitialized and trained from scratch on the remaining dataset. To ensure the stability and reliability of our results, we repeated each method-model-dataset evaluation 3 times and reported the mean and standard deviation in our figures and tables. We reported top-1 classification accuracy and the number of floating point operations (FLOPs) (as cumulative number of samples seen during training) to reach that level of accuracy. Since the main goal is reducing the training costs, a dataset pruning method is better than the other dataset pruning methods if it requires fewer samples to train the model to a certain level of accuracy. The summary of our detailed experiments is shown in Table VI.

4.5.4 Training Details

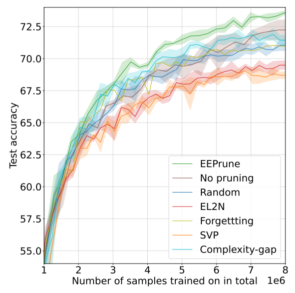
We train our models using stochastic gradient descent with a single NVIDIA RTX A6000 GPU. We train them for 200, 200, 100, 50 and 100 epochs on CIFAR-10, CIFAR-100, Tiny ImageNet, KMNIST and ImageNet respectively. To reduce overfitting and improve generalization, we use label smoothing with $\alpha = 0.1$ [99], dropout with $p = 0.2$ [100] and mixup with $\alpha_{mixup} = 0.2$ [101]. We also employ cosine learning rate decay with a warm-up duration of 5 epochs to gradually reduce the learning rate over the course of training. To optimize GPU memory usage and accelerate training, we set different batch sizes for the EfficientNetV2-M, MobileNetV3-large and ResNet-50 models. We use a batch size of 2048 for EfficientNetV2-M and MobileNetV3-large, and 1024 for ResNet-50.

For EEPrune, we used $E = 10$ for CIFAR-10, CIFAR-100 and KMNIST datasets, and $E = 15$ for the Tiny Imagenet and Imagenet datasets. We found out that as the dataset complexity increased, EEPrune required slightly larger E to decide which samples are redundant, similar to how models need more training to reach a high level of performance on a challenging benchmark. After this point, larger E values did not yield any additional benefits, but increased the training costs (more on this in Section 4.5.6). On the other hand, smaller values resulted in poor performance since a diverse set of samples were pruned, similar to random pruning. We ran experiments for various $t \in [0, 1]$ to prune exactly the 10%, 20%, 30%, 40%, 50% and 60% of the training set. To serve as the early exit point, we added linear layers to the following points: 28th MBConv layer for EfficientNetV2-M, 9th network layer for MobileNetV3-large, and 7th bottleneck block for ResNet-50. All of these points correspond to the middle point of the network.

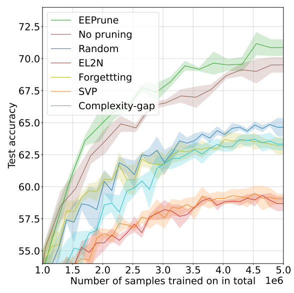
4.5.5 Models Can Reach a Higher Accuracy with EEPrune

In this section, we share our results. Figure 15 illustrates how the dataset pruning methods perform across different model architectures and datasets, in terms of the speed of the models trained on the pruned datasets to reach baseline accuracy. For the no pruning baseline, the curves corresponding to the same model-dataset pair are identical. We only present the portion of these curves that aligns with other dataset pruning method curves in the same subfigure on the x-axis.

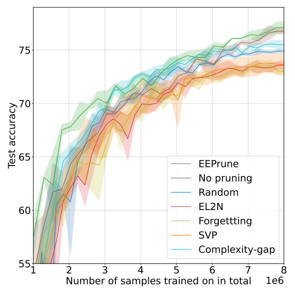
Figures 15a, 15b, 15d and 15j show that models trained on datasets pruned via EEPrune can occasionally surpass the baseline accuracy, despite being trained on less data. As discussed in Section 4.3 and exemplified in Figure 14, removing the simpler samples do not effect the class



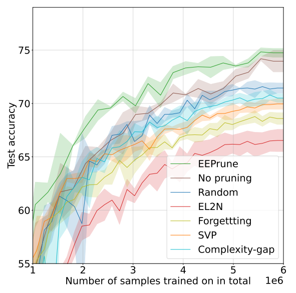
(a) MobileNetV3-large, CIFAR-100, 20%



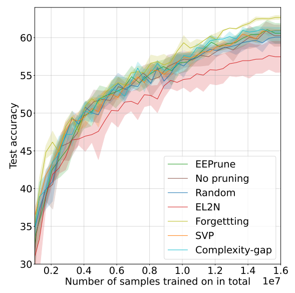
(b) MobileNetV3-large, CIFAR-100, 50%



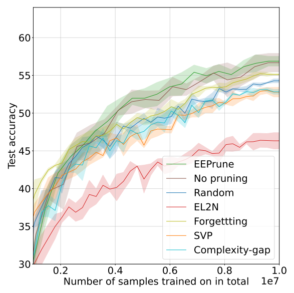
(c) ResNet-50, CIFAR-100, 20%



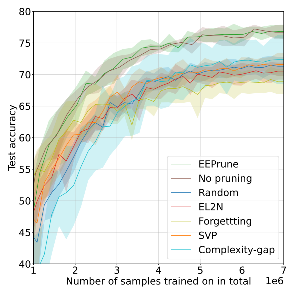
(d) ResNet-50, CIFAR-100, 40%



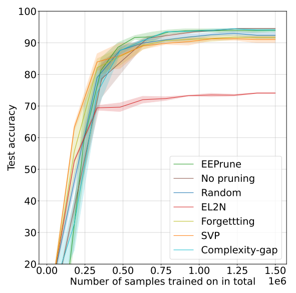
(e) MobileNetV3-large, Tiny ImageNet, 20%



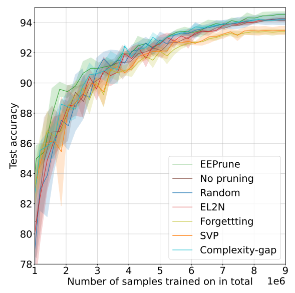
(f) MobileNetV3-large, Tiny ImageNet, 50%



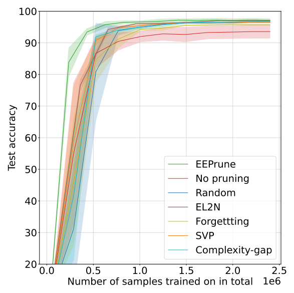
(g) EfficientNetV2-M, CIFAR-10, 30%



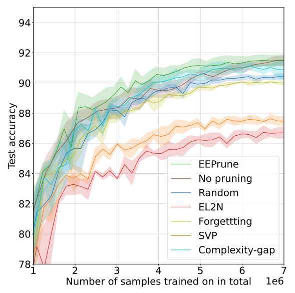
(h) EfficientNetV2-M, KMNIST, 50%



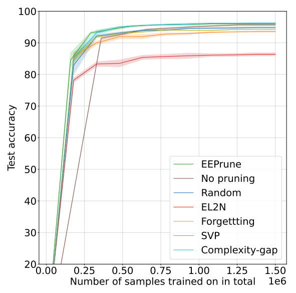
(i) ResNet-50, CIFAR-10, 10%



(j) ResNet-50, KMNIST, 20%



(k) MobileNetV3-large, CIFAR-10, 30%



(l) MobileNetV3-large, KMNIST, 50%

Figure 15. Comparison of dataset pruning methods across different model architectures and datasets, in terms of the speed of the models trained on the pruned datasets to reach the baseline accuracy. The model, dataset and pruning ratios are shown under the subfigures.

decision boundaries. Hence, by removing the simpler samples early in training, EEPrune can achieve the same classification accuracy with lower computational cost of training. In most of the other cases, EEPrune can match the performance of no pruning baseline, and only rarely EEPrune is outperformed slightly by another dataset pruning method (e.g. Figure 15e). Often, complexity gap and random pruning achieves a performance comparable to EEPrune, but EL2N, SVP, and forgetting methods exhibit noticeably worse performance than EEPrune.

One interesting result seen in Figure 15 is that random pruning often performs better than all the other dataset pruning methods except EEPrune. This is particularly surprising as EL2N, Forgetting, SVP and complexity gap require ensembles, long training or sophisticated data scoring but still often place behind random pruning in terms of reached accuracy. One possible reason for this might be ensembles leading to cross-talk and therefore incorrect identification of easy samples. On the other hand, long training and sophisticated data scoring probably *overthink* and fail at identifying easy samples. Although long training and sophisticated data scoring occasionally surpass EEPrune's performance on some model-dataset pairs as seen in Table VII, they are computationally heavy as seen in Figure 16 and contradict the goal of dataset pruning.

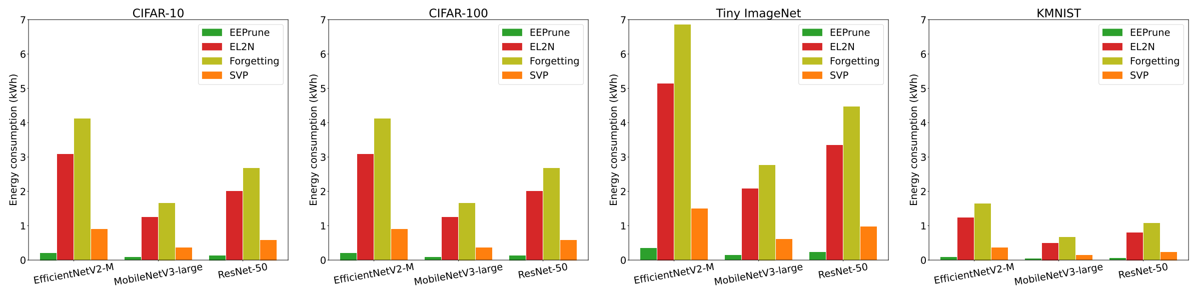
We also share the entire set of results for the MobileNetV3-large model in Table VII. The performances that surpass the no pruning baseline and highest performance in a row are displayed in bold. The accuracies reported for the no pruning baseline are the results obtained after full training sessions on the entire dataset. As seen from the table, EEPrune achieves the highest performance in more than half of the settings. For CIFAR datasets, EEPrune has the majority of

TABLE VII. COMPARISON OF DATASET PRUNING METHODS FOR MOBILENETV3-LARGE.

Dataset, Pruning Ratio	**No pruning**	**EEPrune**	**Random**	**EL2N**	**Forgetting**	**SVP**	**Complexity gap**
CIFAR-10, 10%		$\mathbf{92.05 \pm 0.06}$	91.60 ± 0.22	90.34 ± 0.11	91.26 ± 0.04	90.38 ± 0.11	91.96 ± 0.10
CIFAR-10, 20%		$\mathbf{91.86 \pm 0.05}$	91.07 ± 0.25	88.80 ± 0.29	90.68 ± 0.20	89.00 ± 0.26	91.14 ± 0.36
CIFAR-10, 30%		$\mathbf{91.51 \pm 0.36}$	90.42 ± 0.22	86.69 ± 0.37	90.01 ± 0.14	87.48 ± 0.29	90.88 ± 0.60
CIFAR-10, 40%	89.64 ± 0.55	$\mathbf{90.47 \pm 0.36}$	89.34 ± 0.37	83.63 ± 0.10	88.84 ± 0.07	86.34 ± 0.21	90.03 ± 0.17
CIFAR-10, 50%		88.30 ± 0.74	88.50 ± 0.23	81.03 ± 0.51	88.01 ± 0.23	85.10 ± 0.13	$\mathbf{88.78 \pm 0.16}$
CIFAR-10, 60%		84.48 ± 0.48	86.85 ± 0.17	78.04 ± 1.19	86.10 ± 0.29	82.94 ± 0.32	$\mathbf{86.96 \pm 0.39}$
CIFAR-100, 10%		$\mathbf{73.39 \pm 0.48}$	72.01 ± 0.18	71.34 ± 0.23	71.77 ± 0.03	71.47 ± 0.49	72.30 ± 0.52
CIFAR-100, 20%		$\mathbf{73.57 \pm 0.16}$	71.04 ± 0.42	69.49 ± 0.34	70.98 ± 0.05	68.71 ± 0.27	71.44 ± 0.39
CIFAR-100, 30%		$\mathbf{72.32 \pm 0.41}$	70.07 ± 0.35	66.44 ± 0.61	67.78 ± 0.06	65.88 ± 0.80	70.25 ± 0.25
CIFAR-100, 40%	67.59 ± 0.38	$\mathbf{71.93 \pm 0.70}$	67.02 ± 0.50	62.05 ± 0.25	65.54 ± 0.25	62.87 ± 0.45	67.35 ± 0.48
CIFAR-100, 50%		$\mathbf{70.86 \pm 0.63}$	64.64 ± 0.69	58.67 ± 0.56	63.20 ± 0.68	59.07 ± 0.38	63.33 ± 0.46
CIFAR-100, 60%		$\mathbf{68.52 \pm 0.77}$	60.64 ± 0.48	54.03 ± 0.28	60.33 ± 0.46	55.42 ± 0.14	57.87 ± 0.36
Tiny ImageNet, 10%		61.34 ± 1.31	61.72 ± 1.16	60.93 ± 2.06	$\mathbf{63.43 \pm 0.37}$	62.21 ± 0.67	62.89 ± 0.29
Tiny ImageNet, 20%		60.58 ± 1.32	60.07 ± 0.88	57.46 ± 2.07	$\mathbf{62.67 \pm 0.27}$	60.96 ± 0.11	60.92 ± 0.44
Tiny ImageNet, 30%		59.89 ± 0.95	58.09 ± 1.63	55.73 ± 0.73	$\mathbf{60.79 \pm 0.65}$	58.38 ± 0.50	58.91 ± 0.10
Tiny ImageNet, 40%	56.32 ± 0.95	$\mathbf{58.53 \pm 0.97}$	56.39 ± 0.90	51.70 ± 0.41	58.43 ± 0.55	56.09 ± 0.41	55.89 ± 0.72
Tiny ImageNet, 50%		$\mathbf{56.88 \pm 0.64}$	54.28 ± 0.26	46.33 ± 1.08	55.11 ± 0.11	52.78 ± 0.64	52.82 ± 0.32
Tiny ImageNet, 60%		$\mathbf{55.35 \pm 0.75}$	50.99 ± 0.65	39.59 ± 1.17	51.55 ± 0.05	49.54 ± 0.12	48.50 ± 0.78
KMNIST, 10%		$\mathbf{96.27 \pm 0.16}$	96.11 ± 0.16	93.04 ± 0.03	95.94 ± 0.27	96.00 ± 0.15	96.21 ± 0.15
KMNIST, 20%		96.21 ± 0.04	95.94 ± 0.04	90.99 ± 0.22	95.68 ± 0.10	95.34 ± 0.16	$\mathbf{96.24 \pm 0.03}$
KMNIST, 30%		96.05 ± 0.12	95.69 ± 0.11	89.54 ± 0.53	95.36 ± 0.12	94.83 ± 0.21	$\mathbf{96.13 \pm 0.04}$
KMNIST, 40%	95.10 ± 0.23	96.15 ± 0.11	95.22 ± 0.12	88.49 ± 0.30	94.81 ± 0.12	93.94 ± 0.05	$\mathbf{96.18 \pm 0.11}$
KMNIST, 50%		96.06 ± 0.18	94.84 ± 0.25	86.38 ± 0.50	94.27 ± 0.10	93.58 ± 0.18	$\mathbf{96.23 \pm 0.33}$
KMNIST, 60%		95.72 ± 0.18	94.10 ± 0.01	83.05 ± 1.14	93.62 ± 0.22	92.56 ± 0.24	$\mathbf{96.16 \pm 0.12}$

the high performances, only the complexity gap method can perform better in two high pruning ratio settings. For Tiny ImageNet, forgetting methods achieves strong performance, and is the best for low pruning ratios. On the other hand. EEPrune outperforms all the other methods for high pruning ratios. Finally for KMNIST, complexity gap method achieves five of the six best performances, but all methods are fairly close to each other except the EL2N method.

On a larger and more diverse dataset such as ImageNet, EEPrune is more effective than the other dataset pruning methods as seen from Table VIII. Specifically, EEPrune has the highest early exit and final exit accuracies among all dataset pruning methods, meaning EEPrune reaches the same level of accuracy with less training cost, thus showing the scalability and applicability to larger datasets. On the other hand, the no pruning baseline is unmatched, which suggests that there is still room for improvement to reach the performance of models trained on the full dataset.



(a) (b) (c) (d)

Figure 16. The amount of energy consumed by dataset pruning methods during the pruning phase for different models. (a) CIFAR-10. (b) CIFAR-100. (c) Tiny ImageNet. (d) KMNIST.

TABLE VIII

COMPARISON OF DATASET PRUNING METHODS ON IMAGENET FOR MOBILENETV3-LARGE AND 50% PRUNING.

Method	**Accuracy**	
	Early exit	Final exit
No pruning	48.84 ± 0.07	70.85 ± 0.11
EEPrune	48.22 ± 0.14	68.39 ± 0.11
Random	46.99 ± 0.15	66.23 ± 0.03
EL2N	40.01 ± 0.60	58.52 ± 0.46
Forgetting	46.50 ± 0.23	61.07 ± 0.15
SVP	44.50 ± 0.28	58.24 ± 0.24

4.5.6 EEPrune Uses Less Computation to Prune Data

In this section, we assess how much computation each dataset pruning method uses in order to measure the computational efficiency of methods. We use *carbontracker* Python tool to measure the energy consumption and carbon footprint of the methods [102]. The results are shown in Figure 16.

EL2N uses an ensemble of models and it trains each model for a period of time. Forgetting and SVP methods train the model fully and partially respectively to gather the number of forgetting events. On the other hand, EEPrune uses a single model and trains it for only $E = 10$ epochs, which makes it much more efficient as seen from its energy consumption in Figure 16. In addition, EEPrune outperforms all other methods almost always as seen in the previous section. This makes EEPrune particularly suitable for resource-constrained devices.

Since random pruning and calculating complexity gap scores do not require training of the neural network, and *carbontracker* Python tool only measures the energy consumption during network training, we do not show their performance results in Figure 16. Still, one can argue that random pruning will consume negligible power as it relies simply on randomly selecting the data to prune. However, the resulting inference performance is vastly sub-optimal as we have demonstrated in Figure 15 and Table VII. The complexity gap method relies on creating a Gram matrix $H \in \mathbb{R}^{N_{tr} \times N_{tr}}$, and calculates the complexity gap score for a training sample by inverting H after removing the corresponding row and column from the matrix. Using Schur complement for matrix inversion, the time complexity is $\mathcal{O}({N_{tr}}^3)$, and the space complexity is $\mathcal{O}({N_{tr}}^2)$. Therefore, the time complexity of the complexity gap method grows cubically with the number of training samples but increases linearly with EEPrune. Therefore, EEPrune will consume significantly less power than for datasets with a large number of elements. Not relying on storing and inverting large matrices, using small E (e.g. 10) for a very short amount of training and achieving remarkable results across different models, datasets and pruning ratios make EEPrune an attractive candidate for resource-constrained settings.

4.5.7 EEPrune Retains the Important Samples & Prunes the Easy Samples

In this section, we demonstrate the effectiveness of EEPrune in retaining samples crucial for learning while discarding redundant samples. We set the pruning ratio to 50% and let EEPrune identify the redundant samples, denoted by D_p as in Algorithm 2. Since the pruning ratio is

TABLE IX

MOBILENETV3-LARGE PERFORMANCE WHEN THE MODEL IS TRAINED ON D_P INSTEAD OF $D_{TR} \setminus D_P$ FOR 50% PRUNING WITH EEPRUNE.

Dataset	**Test Accuracy**	
	Trained on D_p	Trained on $D_{tr} \setminus D_p$
CIFAR-10	85.63	**88.30**
CIFAR-100	50.56	**70.86**
Tiny ImageNet	55.43	**56.88**
KMNIST	93.63	**96.06**
ImageNet	68.12	**68.39**

50%, the training set is partitioned into two equal size subsets: D_p and $D_{tr} \setminus D_p$. We then train a MobileNetV3-large on both subsets. We present the results in Table IX. Notably, the model trained on D_p exhibits a subpar performance, indicating the significance of crucial samples present only in $D_{tr} \setminus D_p$.

We also evaluate the model trained on $D_{tr} \setminus D_p$, on D_p instead of the test set. The reason behind is that D_p contains easy and redundant samples, therefore the model should be able to achieve a high accuracy on this set. This is indeed true for low pruning ratios as seen in Table X. As the pruning ratio increases, $D_{tr} \setminus D_p$ shrinks and the model learning worsens. However, the accuracy numbers remain higher compared to the test set accuracies shown in Table VII.

TABLE X

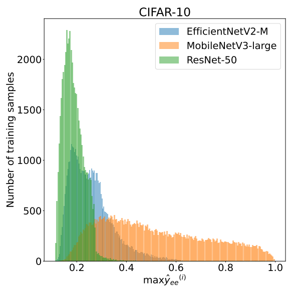
MOBILENETV3-LARGE PERFORMANCE EVALUATED ON D_P AFTER THE MODEL IS TRAINED ON $D_{TR} \setminus D_P$.

Dataset	**Accuracy on D_p**	
	10% pruning	60% pruning
CIFAR-10	98.25	90.94
CIFAR-100	87.95	70.65
Tiny ImageNet	86.85	68.27
KMNIST	99.92	99.81
ImageNet	92.81	70.06

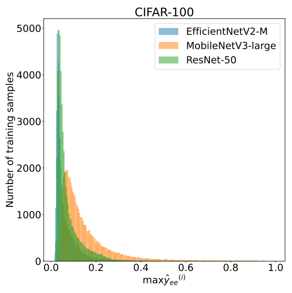
4.5.8 Ablation on Exit Locations

As previously mentioned, we added linear layers after the 28th MBConv layer for EfficientNetV2-M, the 9th network layer for MobileNetV3-large, and the 7th bottleneck block for ResNet-50. All of these points correspond to the middle point of the network. In this section, we investigate the effect of the early exit location on final model accuracy.

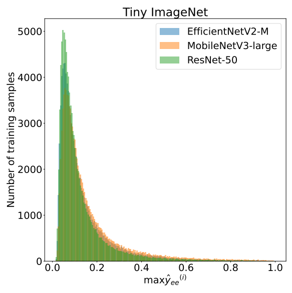
While there are numerous possible locations to place the early exit within a neural network, for the sake of efficiency, we restrict our analysis to two key positions: before and after the midpoint of the network. Specifically, we examine early exit placement after the first and third quartiles of the model, based on the number of layers.



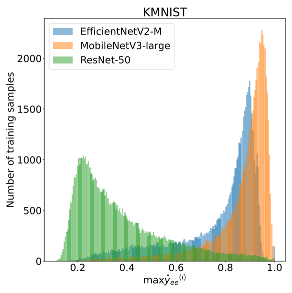
(a)



(b)



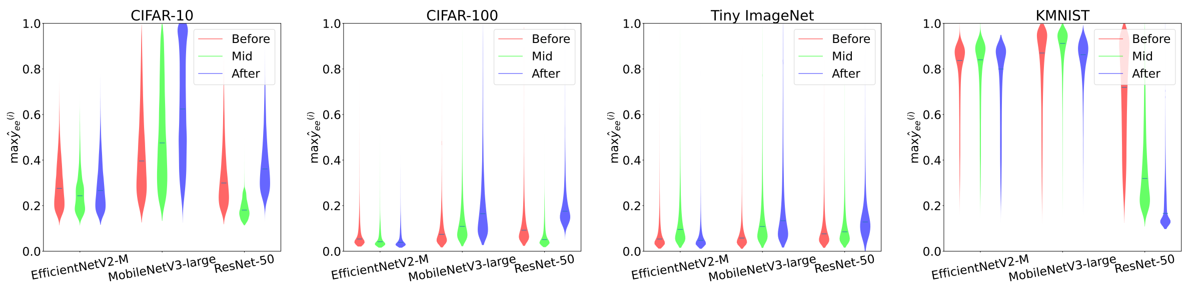
(c)



(d)

Figure 17. Histogram of maximum prediction probability, $\max \hat{y}_{ee}^{(i)}$ for training samples after $E = 10$ epochs. (a) CIFAR-10. (b) CIFAR-100. (c) Tiny ImageNet. (d) KMNIST.

Unlike other dataset pruning methods, EEPrune depends on a prediction threshold t to identify and prune easy samples. This makes it difficult to prune *exactly* $k\%$ of the training samples since it is unknown what value of t will correspond to the easiest $k\%$. For this reason, we ran EEPrune and obtained the histogram of maximum prediction probabilities as shown in Figure 17. As it can be seen from the figure, the threshold t depends on the model and the dataset. For a new model or dataset, the threshold t is determined in the same way after an informative histogram is obtained. Specifically, t is the maximum value that ensures exactly $k\%$ of the samples have $max\hat{y}_{ee}^{(i)} > t$. We note that this phase consumes significantly less energy compared to other dataset pruning algorithms as shown in Figure 16. We also discover that it depends on the exit location. Figure 18 shows the distribution of maximum prediction probability $max\hat{y}_{ee}^{(i)}$ for different models, datasets and exit locations.



(a) (b) (c) (d)

Figure 18. Violin plot demonstrating how maximum prediction probability $max\hat{y}_{ee}^{(i)}$ varies when the early exit is placed before (red) and after (blue) the mid (green) point of the model. Before, mid and after corresponds to first, second and third quartile of the model in terms of number of layers. (a) CIFAR-10. (b) CIFAR-100. (c) Tiny ImageNet. (d) KMNIST.

Figure 18 shows that placing the early exit at the third quartile of the model results in a higher t for pruning the same amount of data except in few cases (e.g. ResNet-50 on KMNIST). This indeed makes sense since a deeper exit uses more computation and *understands* the data better, making it more confident about its predictions. Depending on the inherent dataset difficulty, placing the exit at an earlier location might suffice too (e.g. KMNIST).

We now show the performance results. We consider MobileNetV3-large and ResNet-50 with the CIFAR-100 dataset. As shown earlier in Figure 15 and Table VII, for this combinations, EEPrune outperforms all existing methods at all pruning rates. As seen from Table XI, placing the exit too early or too deep can hurt the final performance. In general, placing it early rather than deep gives a better result, possibly highlighting the fact that earlier layers can detect easier samples. Nevertheless, opting for the mid point turns out to be the optimal choice for

TABLE XI

COMPARISON OF EXIT LOCATIONS FOR EEPRUNE ON CIFAR-100 FOR MOBILENETV3-LARGE AND RESNET-50.

Model	**Exit Location**	**Pruning Ratio**		
		10%	**30%**	**60%**
MobileNetV3-large	Before	67.98 ± 0.37	67.66 ± 0.45	67.59 ± 0.56
	Mid	$\mathbf{73.39 \pm 0.48}$	$\mathbf{72.32 \pm 0.41}$	$\mathbf{68.52 \pm 0.77}$
	After	65.57 ± 0.37	67.01 ± 0.59	67.57 ± 0.48
ResNet-50	Before	75.13 ± 0.41	74.89 ± 0.53	71.26 ± 0.37
	Mid	$\mathbf{77.09 \pm 0.23}$	$\mathbf{75.27 \pm 0.20}$	$\mathbf{71.57 \pm 0.43}$
	After	75.60 ± 0.19	74.14 ± 0.22	70.08 ± 0.30

all pruning rates and network models that we have considered. In fact, the prudent choice of choosing the mid point strikes a balance between computational cost and extracted information, acknowledging that moderation is often the key to success in life. We note that the *specific* exit location (e.g. layer 49 vs. layer 50 vs. layer 51 for a 100-layer network) is a hyper-parameter that should be tuned and evaluated on a hold-out set.

4.5.9 Class Imbalance

While it is feasible to apply a constraint to all dataset pruning methods to maintain an equal pruning of data from each class, these methods do not inherently have that ability. Here, we delve deeper into the samples removed by each dataset pruning method. Figure 19 displays a heat map illustrating the number of samples removed from each class across the four datasets. The most and the least pruned classes are shown in Table XII.

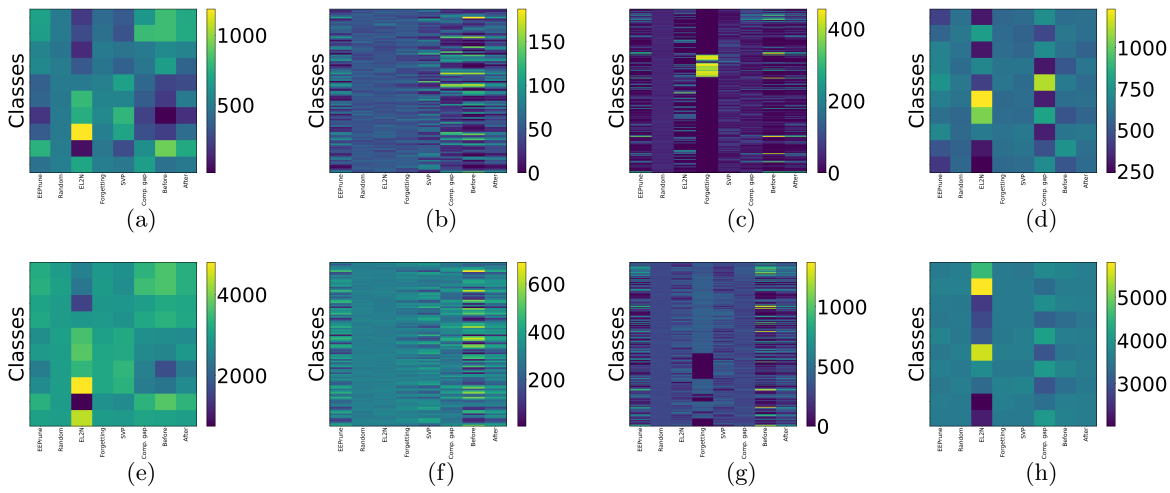


Figure 19. Number of samples each dataset pruning method discards from the training set shown as heat map. The model is MobileNetV3-large. (a,e) CIFAR-10. (b,f) CIFAR-100. (c,g) Tiny ImageNet. (d,h) KMNIST. First row shows 10% total pruning and second row shows 60% total pruning. The 8 columns correspond to EEPrune, Random, EL2N, Forgetting, SVP, Complexity Gap, EEPrune-Before and EEPrune-After.

First, we take a look at low (10%) amount of data pruning. For CIFAR-10, EEPrune seems to prune the *cat* class less than the others. Changing the exit location affects the number of samples pruned from each class, but still the *cat* class is the least pruned class. The most uniform pruning is naturally achieved by random pruning, and forgetting and SVP come next. EL2N achieves a particularly imbalanced pruning as roughly one quarter of samples from the *bird* class is pruned despite 10% total pruning.

Pruning behaviors show similarity between KMNIST and CIFAR-10. We hypothesize that this is due to both having relatively small number of classes. EL2N and complexity gap methods prune two different characters more than the others and they cause the most imbalanced dataset.

TABLE XII. CLASSES THAT UNDERGO THE MOST AND THE LEAST PRUNING WHEN THE DATASET IS SUBJECTED TO EACH METHOD FOR 10% PRUNING. THE MODEL IS MOBILENETV3-LARGE.

Method	**Most Pruned Classes**				**Least Pruned Classes**			
	CIFAR-10	**CIFAR-100**	**Tiny ImageNet**	**KMNIST**	**CIFAR-10**	**CIFAR-100**	**Tiny ImageNet**	**KMNIST**
EEPrune	automobile (765)	wardrobe (105)	monarch (347)	ha (750)	cat (190)	otter (0)	tailed frog (0)	o (401)
Random	airplane (520)	girl (67)	Christmas stocking (69)	re (636)	horse (470)	rocket (33)	convertible (36)	ki (557)
EL2N	bird (1187)	train (74)	desk (376)	na (1235)	automobile (64)	bus (34)	goldfish (0)	o (241)
Forgetting	cat (567)	seal (73)	plunger (455)	ha (632)	truck (427)	sunflower (27)	goldfish (0)	wo (568)
SVP	cat (785)	otter (136)	pop bottle (168)	su (644)	automobile (293)	wardrobe (10)	dugong (8)	re (558)
Complexity gap	ship (814)	orange (161)	orange (223)	ha (1119)	cat (244)	snake (6)	cougar (4)	ma (305)
EEPrune-Before	automobile (928)	wardrobe (187)	monarch (447)	ki (751)	cat (15)	beaver (0)	tailed frog (0)	tsu (501)
EEPrune-After	ship (732)	wardrobe (102)	monarch (321)	su (639)	cat (178)	otter (0)	tailed frog (0)	ya (559)

When the number of classes increases, the imbalance in pruning becomes more apparent as seen in Figure 19b and Figure 19c. EEPrune for CIFAR-100 and forgetting method for Tiny ImageNet seem to favor some classes more than the others for pruning. Also, changing the exit location significantly changes the pruning behavior.

Table XII shows the most and the least pruned classes. Changing the exit location does not change the class that is pruned by EEPrune the most or the least, except for the KMNIST dataset and the "EEPrune-After" method for the CIFAR-10 dataset. Except for a few cases, the other methods focus on different classes each.

When we increase the pruning ratio to 60%, we can see from the second row of Figure 19 that the heat maps become slightly more uniform, except the EL2N method. EEPrune-Before on CIFAR-100 and Tiny ImageNet continues exhibiting imbalanced pruning, which suggests exit location should be carefully selected.

4.5.10 Ablation on EEPrune Conditions

EEPrune is simple and flexible, therefore new data pruning schemes can be easily be proposed based on it. Here, we describe two natural ideas based on accumulating the loss values of the exits and assigning weights to the original conditions of EEPrune.

EEPrune-Loss: According to a previous study conducted by [103], early exits can act as a form of regularization when the exit losses are accumulated and the exit layers are optimized together. In an ideal scenario, easy samples would have both low early exit losses and low final exit losses, but this may not always be the case. We propose bypassing the EEPrune conditions and evaluating the samples based on the sum of the exit losses as in Equation 4.3, similar to how the exit layers are trained jointly by accumulating losses. By sorting the samples based on their aggregate losses, we can prune those with the lowest overall loss. The loss weights λ_1 and λ_2 can be tuned as well, but for simplicity we use equal weights of 1.

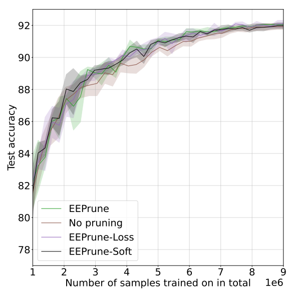
$$J = -\lambda_1 \sum_{c=1}^{C} {y_c}^{(i)} \log(\hat{y}_{ee_c}^{(i)}) - \lambda_2 \sum_{c=1}^{C} {y_c}^{(i)} \log(\hat{y}_{fe_c}^{(i)}) \tag{4.3}$$

EEPrune-Soft: EEPrune is a strict pruning method in the sense that it requires all three conditions to be met for a sample to be pruned. However, it can be made more lenient by assigning weights to the conditions. To this end, we propose assigning weights $\omega_1, \omega_2, \omega_3$ to the three conditions in Algorithm 2, where the sum of the weights is one. This approach allows for a better assessment of the conditions, with each weight indicating the degree to which the condition should be met in order for a sample to be pruned. For example, if the first condition is satisfied, the sample would be assigned a score of ω_1, and so on for the other conditions.

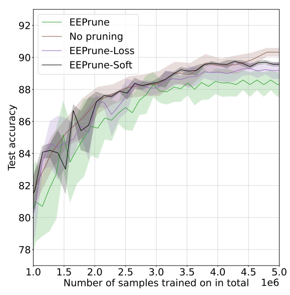
While this weighting system provides a more refined approach to scoring the samples, it may not be sufficient in cases where there is a tie. To address this limitation, we propose the addition of a fourth score, $(\max \hat{y}_{ee}^{(i)} + \max \hat{y}_{fe}^{(i)}) \cdot \omega_4$, which averages the maximum element of the early exit prediction vector and the final exit prediction vector for the sample of interest. Here, ω_4 should be very small to avoid dominating the other scores. By incorporating this fourth score, we can better differentiate between samples with similar scores and make more informed decisions about which samples to prune. Finally, the samples can be sorted according to their total score and the samples with the lowest total score can be pruned.

For EEPrune-Loss, after training for E epochs, we calculated the sum of the early exit loss and the final exit loss for each sample, sorted them, and pruned the lowest scoring $10\%, 20\%, 30\%, 40\%, 50\%, 60\%$ of samples based on their loss. For EEPrune-Soft, we used $\omega_1 = 0.4$, $\omega_2 = 0.2$, $\omega_3 = 0.4$, and $\omega_4 = 0.001$ for weighting the conditions as this set of values allows breaking the ties between sample scores, utilizes both exit predictions and no condition dominates another.

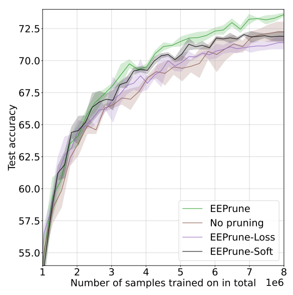
The results are shown in Figure 20. Figures 20a and 20b show that EEPrune-Soft can match the performance of the no pruning baseline despite EEPrune's performance falls of when the pruning rate increases. However, on a more challenging data such as CIFAR-100 and Tiny ImageNet (Figures 20c, 20d, 20g, 20h), neither EEPrune-Soft nor EEPrune-Loss can match EEPrune's performance. In addition, although there are some cases (Figures 20c, 20d, 20e, 20f)



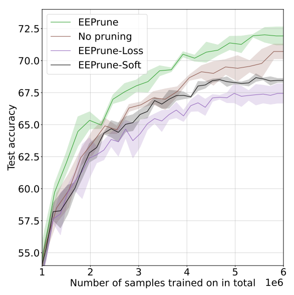
(a) MobileNetV3, CIFAR-10, 10%



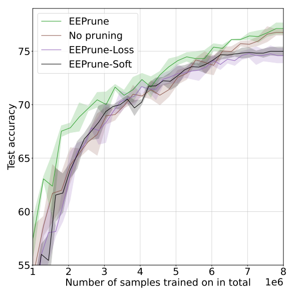
(b) MobileNetV3, CIFAR-10, 50%



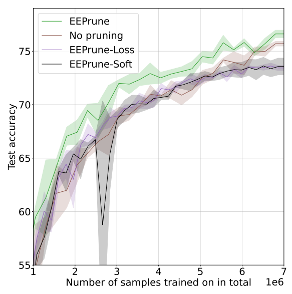
(c) MobileNetV3, CIFAR-100, 20%



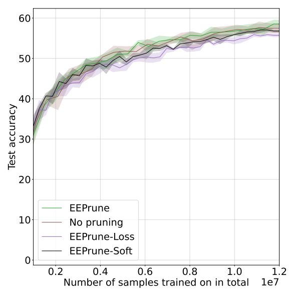
(d) MobileNetV3, CIFAR-100, 40%



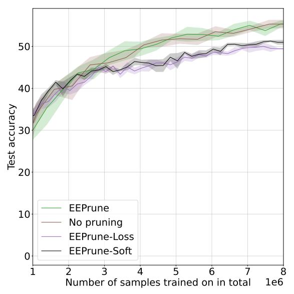
(e) ResNet-50, CIFAR-100, 20%



(f) ResNet-50, CIFAR-100, 30%



(g) MobileNetV3, Tiny ImageNet, 40%



(h) MobileNetV3, Tiny ImageNet, 60%

Figure 20. Comparison of EEPrune-Loss and EEPrune-Soft against EEPrune and no pruning baseline.

where EEPrune can surpass no pruning baseline, we find that EEPrune-Loss and EEPrune-Soft cannot ever surpass the no pruning baseline. This result suggests that EEPrune's reasonable strictness on satisfying all three conditions together is an important property.

CHAPTER 5

CLASS-AWARE INITIALIZATION OF EARLY EXITS FOR PRE-TRAINING LARGE LANGUAGE MODELS

Overview: We propose a novel class-aware weight initialization technique for early exit large language models with the purpose of accelerating pre-training. Our design utilizes the neural collapse phenomenon combined with a Gaussian mixture model for the distribution of feature vectors at a given layer. Specifically, we calculate the average of token representations at the early exit point and use the resulting vectors together with class probabilities for initializing the early exit vectors. The next token prediction accuracy of our class-aware initialization technique is up to five times higher than other baselines at epoch zero and matches or surpasses them in later epochs throughout the pre-training process.

Keywords: early exit, weight initialization, class means, pre-training, LLMs

5.1 Introduction

State-of-the-art large language models (LLMs) have a large number of parameters, and generally, the higher the number of parameters, the better the performance [6, 46, 77, 78, 104–107]. However, their large size and autoregressive design results in high inference latency, which is not desirable for low resource environments and time sensitive settings.

The vast majority of LLMs have a "tunnel-like" architecture: The input to the model is processed by all of the layers in a sequential manner, regardless of the input's inherent difficulty [12, 44]. On the other hand, not all inputs have the same level of difficulty. Early exit networks exploit this heterogeneous difficulty of inputs. One or more intermediate classifiers are attached to the model, allowing token-level conditional computation [10–12, 44, 108–111]. Easy tokens can exit early from the LLM in order to save computation.

While the addition of early exits can reduce the inference latency, initially they do not possess the optimal weights. The early exit layers have to be trained first, before being effective at inference time. Early exit layers are typically trained together with the backbone model, with two primary approaches: Training only the early exit and the final exit layers while freezing the non-exit layers, or training the backbone and the exits together [11, 12]. Generally, the latter performs better since everything is optimized jointly, but the cross-talk between the exits of the network may lead to suboptimal learning and long training times [12]. Ideally, we would like to initialize the weights of the early exit layers in such a way that the cross-talk is minimized, the joint training is facilitated and training time is reduced.

The sizes of both the state-of-the-art LLMs and their training data lead to long training time and high costs. This makes training an early exit LLM even more difficult and costly. In this work, we propose a novel class-aware early exit initialization technique for early exit LLMs to reduce the pre-training costs. We make connections to the optimal detection for the vector additive white Gaussian noise (AWGN) channel from the digital communications domain and utilize the *neural collapse phenomenon* [39]. Specifically, we calculate the average of token representations at the early exit point and use the resulting vectors for the initialization. While calculating the average of vector representations has been shown to work well as a decision mechanism for early exit networks [44, 112], our work is the first to apply it to the initialization of early exit LLMs with the purpose of accelerating pre-training.

We demonstrate the effectiveness of our novel weight initialization technique on WikiText-2 [113] and BookCorpus [114] datasets using OPT [46] and TinyLlama [115] models. Notably, our class-aware initialization technique achieves 5× the performance of other baselines at epoch zero. Moreover, it can match or surpass the other baselines at later epochs throughout the pre-training.

5.2 Related Work

5.2.1 Early Exit LLMs

Numerous attempts have been made in the past to reduce the inference latency of transformer [5] based models in the past. Perhaps the most visited idea has been adding early exits to BERT variants [14, 15, 116–118]. However, these models are primarily designed for classification tasks such sentiment analysis, rather than language modeling and text generation.

Developing early exit LLMs for text generation is more challenging, because token-level early exiting requires careful consideration [119, 120]. Copying hidden states of tokens that exited early to the deeper layers for KV-caching, which confidence measure to use and batch inferencing have been tackled in the past [108–110].

5.2.2 Efficient LLM Training

As the number of parameters in an LLM grows, fine-tuning it on datasets becomes more time-consuming and expensive. To address this challenge, researchers have explored parameter-efficient fine-tuning techniques such as adapter approaches, often coupled with quantization [121–124].

These methods involve training only a subset of the model parameters, effectively reducing the overall training cost. Most recently, parameter-efficient fine-tuning of early exit LLMs has been explored via data, tensor and pipeline parallelism [125, 126].

5.3 Preliminaries and Problem Formulation

In this section, we establish our notation and lay the foundation for our method by describing the pre-training process of a decoder-only LLM. We then provide background on the problem of optimal detection for the vector AWGN channel from the digital communications domain, which we will make critical connections to later on.

5.3.1 Pre-training

We focus on the pre-training phase of models belonging to the family of decoder-only LLMs. The model consists of an embedding layer, L decoder blocks and a language modeling (LM) head. Let V, D, C denote the vocabulary size, the embedding dimensionality, and the context length, respectively.

During the pre-training process, the tokenizer breaks down a text from the training set into C tokens, denoted as $T_i \in \{1, \dots, V\}$, where $i \in \{1, \dots, C\}$. Let S_v denote the set of all training tokens T_i at any position i such that $T_i = v$.

The tokens are subsequently passed through the embedding layer in parallel, resulting in corresponding vectors $R_{i,0} \in \mathbb{R}^D$. These vectors are then fed into the first decoder block, generating output vectors $R_{i,1} \in \mathbb{R}^D$. This iterative process continues sequentially, with the output $R_{i,l}$ of decoder block l being passed to decoder block $l+1$ as the input, where l ranges from 1 to $L-1$.

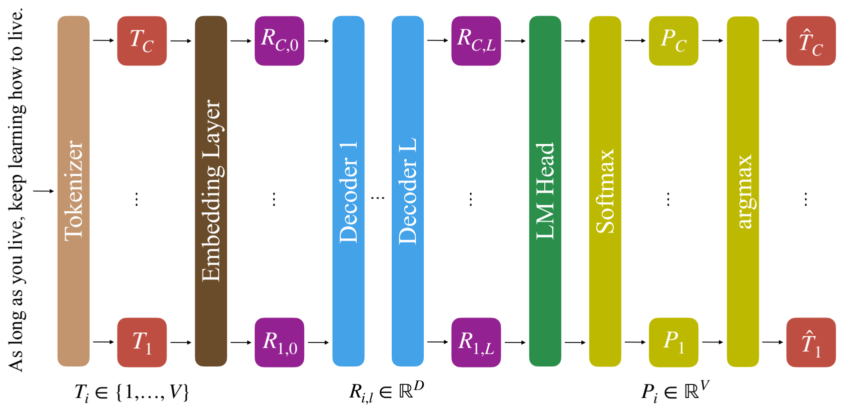


Figure 21. Feed-forward phase of pre-training a decoder-only language model.

In the final stage of the feed-forward process, the output $R_{i,L}$ of the last decoder block is fed to the LM head, which is a linear layer with weight matrix $W \in \mathbb{R}^{D \times V}$. The output of the LM head is converted to probability vectors $P_i \in \mathbb{R}^V$ via the softmax operation. Suppose the index of the maximum probability in P_i is $\hat{T}_i$. Since the primary objective of the pre-training phase is next-token prediction, the model is optimized with the cross-entropy loss to ensure $\hat{T}_i = T_{i+1}$. This process is shown in Figure 21.

In order to accelerate inference, one or more early exit LM heads can be integrated to the already pre-trained decoder-only language model. However, the integration of the additional layer(s) necessitates a separate pre-training, which may incur substantial costs as discussed in Section 5.1. Here, we assume that only one early exit LM head is added. Suppose that this LM head appears after decoder block K where $K < L$, and its weight matrix is $\overline{W} \in \mathbb{R}^{D \times V}$, sharing the same dimensions with the backbone LM head. Our main goal is to find a smart way of initializing $\overline{W}$ such that the pre-training phase for the early exit LM head will be accelerated, and therefore training costs associated with it will be decreased. Our proposed solution relies on the problem of optimal detection for the vector AWGN channel.

5.3.2 Optimal Detection for the Vector AWGN Channel

The vector AWGN channel can be modeled as

$$r = s_m + n, \quad m \in \{1, \dots, M\}, \tag{5.1}$$

where r, s_m and n are N-dimensional vectors. A message s_m is sent to the receiver through the AWGN channel, which adds a noise n to the message. The components of the noise vector are independent and identically distributed Gaussian random variables with zero mean and $\frac{N_0}{2}$ variance. The receiver observes r, and decides which message was sent among $\{s_1, \dots, s_M\}$. The goal is to minimize the probability of error. Using the Bayes rule, the optimal detection rule can be written as

$$\begin{aligned} \hat{m} &= \underset{1 \le m \le M}{\arg\max} \left[P(s_m \mid r)\right] \\ &= \underset{1 \le m \le M}{\arg\max} \left[\frac{P(s_m)P(r \mid s_m)}{P(r)}\right] \\ &= \underset{1 \le m \le M}{\arg\max} \left[P(s_m)P(r \mid s_m)\right]. \end{aligned} \tag{5.2}$$

As in Equation 4.2-15 from [127], the equation above can be simplified further as follows:

$$\begin{aligned}
\hat{m} &= \underset{1\leq m\leq M}{\arg\max}\ [P(s_m)P(r \mid s_m)] \\
&= \underset{1\leq m\leq M}{\arg\max}\ [P(s_m)P_n(r - s_m)] \\
&= \underset{1\leq m\leq M}{\arg\max}\ \left[P(s_m)\left(\frac{1}{\sqrt{\pi N_0}}\right)^N e^{-\frac{\|r-s_m\|^2}{N_0}}\right] \\
&= \underset{1\leq m\leq M}{\arg\max}\ \left[P(s_m)e^{-\frac{\|r-s_m\|^2}{N_0}}\right] \\
&= \underset{1\leq m\leq M}{\arg\max}\ \left[\ln P(s_m) - \frac{\|r - s_m\|^2}{N_0}\right] \\
&= \underset{1\leq m\leq M}{\arg\max}\ \left[\frac{N_0}{2}\ln P(s_m) - \frac{1}{2}\|r - s_m\|^2\right] \\
&= \underset{1\leq m\leq M}{\arg\max}\ \left[\frac{N_0}{2}\ln P(s_m) - \frac{1}{2}\|s_m\|^2 + r \cdot s_m\right] \\
&= \underset{1\leq m\leq M}{\arg\max}\ [\eta_m + r \cdot s_m],
\end{aligned} \tag{5.3}$$

where $\eta_m = \frac{N_0}{2}\ln P(s_m) - \frac{1}{2}\|s_m\|^2$. The careful reader will notice the striking similarity between the last line of Equation 5.3 and the operational logic of a linear layer serving as a classification head. Given an input x, the linear layer with weights w and biases b classifies the input according to the maximum element of $b + x \cdot w$.

5.4 Method

Our aim is to initialize the early exit LM head in such a way that it starts from a reasonably good point and achieve a certain level of next-token prediction accuracy before any pre-training, rather than starting from a random point achieving a low next-token prediction accuracy.

We calculate the average of all output vectors after decoder K that correspond to the tokens in S_v:

$$M_v = \frac{1}{|S_v|} \sum_{T_i \in S_v} R_{i,K}. \tag{5.4}$$

Note that the backbone model is already pre-trained at this point, therefore the intermediate representations are not bad representations. The underlying idea behind Equation 5.4 is the *neural collapse phenomenon* [39]: The intermediate representation of an input belonging to a certain class converges to its corresponding class mean in the final layer of the network. Here, we carry this idea one step further and postulate that, if the input token T_i satisfies $T_i \in S_v$ for some class/word v, then the corresponding feature $R_{i,K}$ at layer K is a Gaussian random vector with mean M_v (i.e. the class mean in (Equation 5.4)), and covariance $\frac{N_0}{2}\mathbf{I}$, where N_0 is a hyperparameter to be tuned experimentally. Now suppose that the early exit LM head is the receiver we mentioned in Section 5.3.2. In this context, we can write

$$R_{i,K} = M_v + \epsilon, \quad v \in \{1, \dots, V\}, \tag{5.5}$$

where $R_{i,K}$, M_v, and ϵ are all the D-dimensional vector. Also, ϵ, is a zero-mean noise vector with covariance $\frac{N_0}{2}\mathbf{I}$. The mean vector M_v is sent to the early exit LM head as the message, and noise ϵ has been added during transmission. The early exit LM head observes $R_{i,K}$, and decides which mean vector was sent among $\{M_1, \dots, M_V\}$.

Similar to Equation 5.3, the optimal decision equation for the early exit LM head can be written as

$$\begin{aligned}\overline{T}_i &= \underset{1\leq v\leq V}{\arg\max}\left[\eta_v + R_{i,K}\cdot M_v\right], \, i = 1,\ldots,C \\ \eta_v &= \frac{N_0}{2}\ln P(M_v) - \frac{1}{2}\left\|M_v\right\|^2,\end{aligned} \tag{5.6}$$

where N_0 is a hyper-parameter and $P(M_v)$ is the prior probability for each token in the training set, determined using the empirical frequencies in the training set.

As a result, the early exit LM head is initialized as

$$\overline{W} = [M_1, \ldots, M_V] \in \mathbb{R}^{D\times V}, \tag{5.7}$$

with a separate bias vector $\eta = [\eta_1, \ldots, \eta_V]$. Our initialization method is shown in Figure 22.

5.5 Experiments and Results

In this section we describe our experiments in detail and present the numerical results.

5.5.1 Models

In our experiments, we used OPT-125M, OPT-350M, OPT-1.3B models from the OPT model family [46]; as well as TinyLlama-1.1B [115].

The OPT model family is a series of open-sourced decoder-only language models ranging from 125M to 175B parameters. The largest OPT model performs similarly to GPT-3 [6] with approximately $1/7^{th}$ of the training cost [46].

The TinyLlama1.1B model is developed with the goal of pre-training such a compact model on 3 trillion tokens [115]. The model shares the same architecture with Llama2 model family [104].

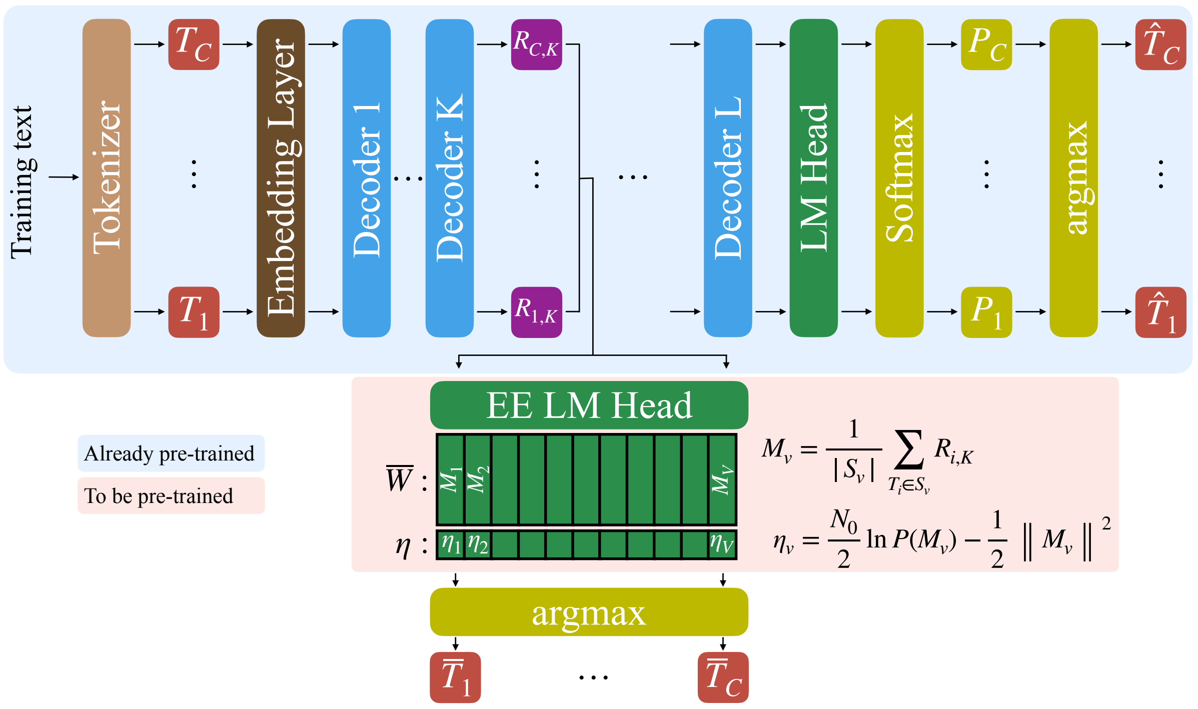


Figure 22. Our proposed method for initializing the early exit LM head using the mean representation vectors for each token in the vocabulary set.

The number of decoder layers (L), embedding dimensionality (D) and vocabulary size (V) of the models we used in our experiments are shown in Table XIII.

5.5.2 Datasets

In our experiments, we used the WikiText-2 [113] and the BookCorpus [114] datasets for pre-training the models.

TABLE XIII

SUMMARY OF THE MODELS USED IN OUR EXPERIMENTS.

Model	L	D	V
OPT-125M	12	768	50272
OPT-350M	24	1024	50272
OPT-1.3B	24	2048	50272
TinyLlama-1.1B	22	2048	32000

WikiText-2 is a collection of tokens extracted from the verified "Good" and "Featured" articles from Wikipedia [113]. For pre-training, we used the "wikitext-2-v1" subset from HuggingFace, which contains 44.8K rows.

BookCorpus dataset is a large collection containing more than 11K books and 74M rows [114]. Due to its size, we used 1% of the dataset. We allocated 80% of the 1% for pre-training, and the remaining 20% for evaluation.

5.5.3 Experiment Settings

For a backbone model, we begin by downloading the most recent checkpoint from HuggingFace. This checkpoint is the result of training the model on a large and diverse collection of datasets. However, since we are going to add an early exit layer and pre-train it on only one dataset, we fine-tune the backbone model on our dataset for 3 epochs so the effect of other datasets on the model is minimal. We found out that fine-tuning for more than 3 epochs led to overfitting.

After the initial fine-tuning of the backbone model, we add the early exit LM head after decoder $K = L/2$. Specifically, K is 6, 12, 12 and 11 for OPT-125M, OPT-350M, OPT-1.3B and TinyLlama-1.1B models respectively. The early exit LM heads share the same architecture and number of parameters with the backbone LM head, the only difference is that we allow the early exit LM head to have a bias vector.

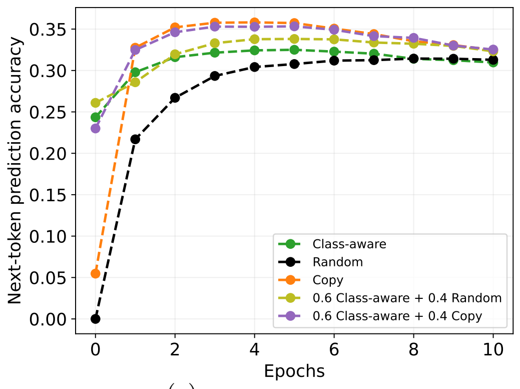
We train the resulting early exit LLM on two different settings. In the first setting, called "no freezing," all parameters are trainable. In the second setting, called "freezing," we freeze the parameters of the model except the two LM heads. These two settings are how early exit neural networks are trained in the literature [90, 128]. The training is done on a single NVIDIA A6000 with a batch size of 32. Due to limited hardware memory, we used a context length of $C = 128$. We used PyTorch [129] in our experiments.

5.5.4 Results

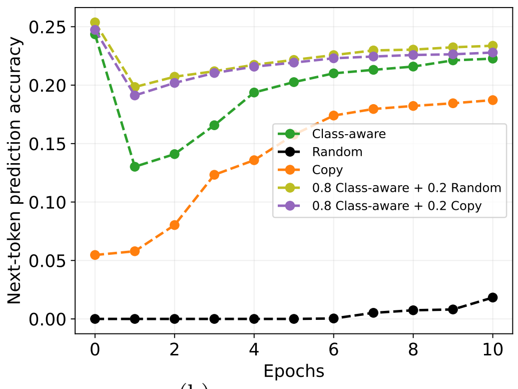
We now present the results of our class-aware early exit initialization method and compare it against two other initialization techniques:

1 **Random initialization:** This is the default weight initialization technique for linear layers in PyTorch [129]. The weights $\overline{W} \in \mathbb{R}^{D \times V}$ are initialized as $\overline{W} \sim U\left[-\frac{1}{\sqrt{D}}, \frac{1}{\sqrt{D}}\right]$, where U is the random uniform distribution.

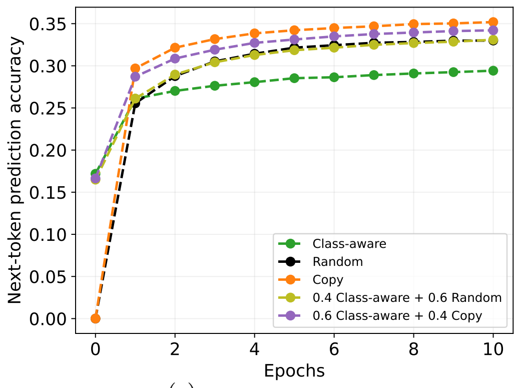
2 **Copy-from-backbone:** Since the weights W of the backbone LM head is already pre-trained and have the same dimensions as $\overline{W}$, copying W into $\overline{W}$ can serve as a good starting point [126].



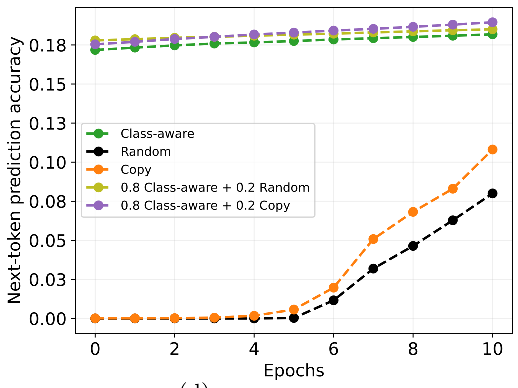
(a) OPT-125M, No freezing



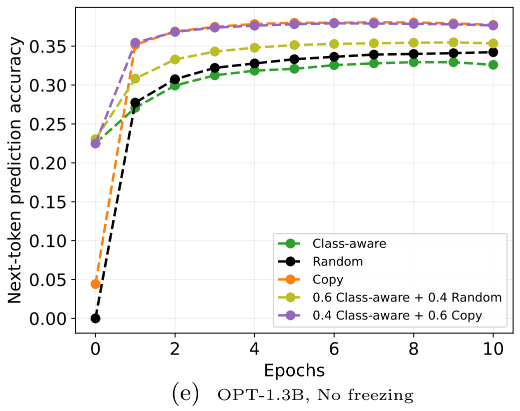
(b) OPT-125M, Freezing



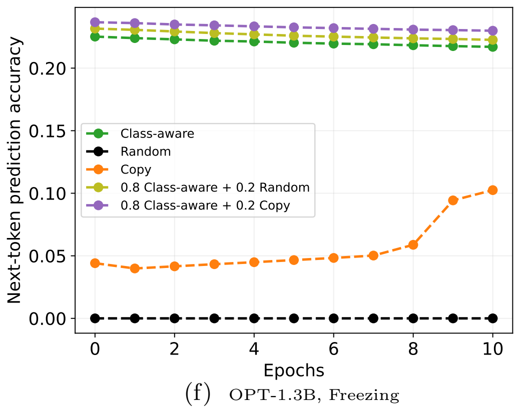
(c) OPT-350M, No freezing



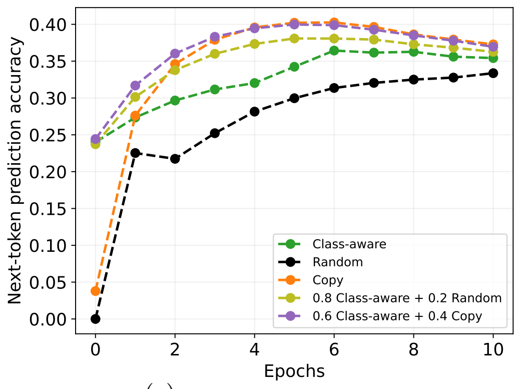
(d) OPT-350M, Freezing



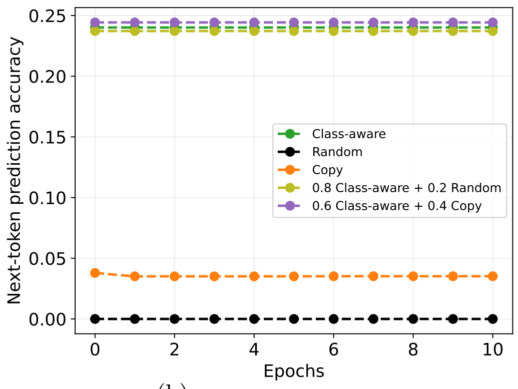
(e) OPT-1.3B, No freezing



(f) OPT-1.3B, Freezing



(g) TinyLlama-1.1B, No freezing



(h) TinyLlama-1.1B, Freezing

Figure 23. Next-token prediction accuracies on WikiText-2 for the early exit LM head initialization techniques.

For our class-aware initialization, we use $N_0 = 0.25$ and we use the empirical frequencies of tokens in the training set for $P(M_v)$, i.e., the number of occurrences of the token divided by the total number of tokens in the training set.

We report the next-token prediction accuracy throughout the pre-training epochs for all initialization techniques. We pre-trained the models for 10 epochs as performance started to drop due to overfitting. The results on the WikiText-2 datasets are shown in Figure 23.

The most important takeaway from Figure 23 is that, our class aware initialization technique achieves 25% next-token prediction accuracy at epoch zero, without any training. On the other hand, random initialization and copying from backbone can achieve at most 5%. This shows that class-aware initialization of early exits is a promising technique for resource constrained devices and settings.

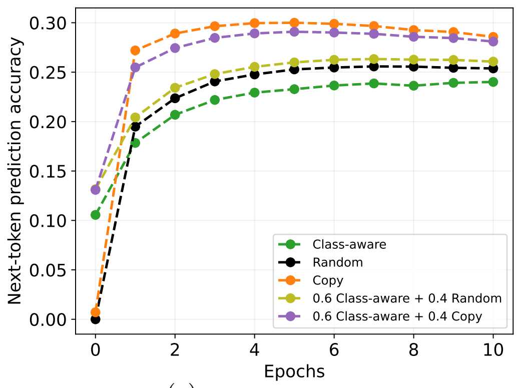
For the "no freezing" setting, although class-aware initialization starts pretty well, it is surpassed by the copy-from-backbone method easily. There are also some scenarios where random initialization surpasses the class-aware initialization as in Figures 23c and 23e. Here, we can easily match the baselines via a convex combination:

$$\overline{W} = \alpha W_{CA} + (1 - \alpha) W_B, \tag{5.8}$$

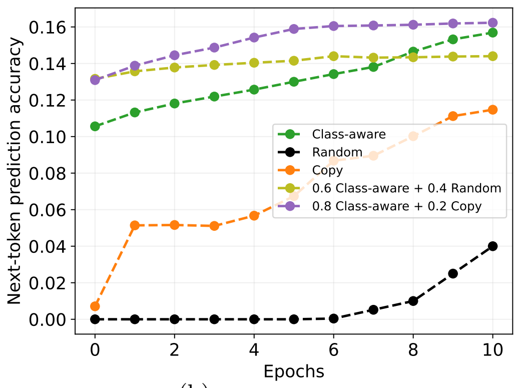
where W_{CA} is the weights initialized in a class-aware manner, and W_B is either random initialized weights or the copied weights from the backbone LM head. This convex combination gets the best of both worlds: It helps preserve the performance of class-aware initialization at epoch zero, and it matches the copy-from-backbone performance at later epochs. In our experiments we evaluated $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$, and we show the best performing α-curves in Figure 23.

In the "freezing" setting, only the LM heads are trainable, therefore learning is more difficult. As it can be seen from Figures 23b, 23d, 23f, 23h; the random and copy-from-backbone methods struggle heavily and cannot achieve a good next-token prediction accuracy. On the other hand, our class-aware initialization starts from a pretty good point and keeps performing at the same level throughout the pre-training. Only for the OPT-125M model, there is a sharp drop at the first epoch of the pre-training as seen in Figure 23b. This drop can be somewhat treated by the convex combination equation given in Equation 5.8.

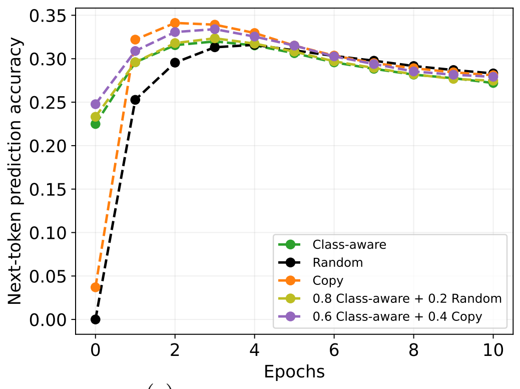
The same trends are observed for the BookCorpus dataset as seen in Figure 24. Specifically, without any training, the class-aware initialization starts from a high next-token prediction accuracy and the convex combination allows preserving the high performance throughout the pre-training. Notably, in the "freezing" setting, class-aware initialization performs the best



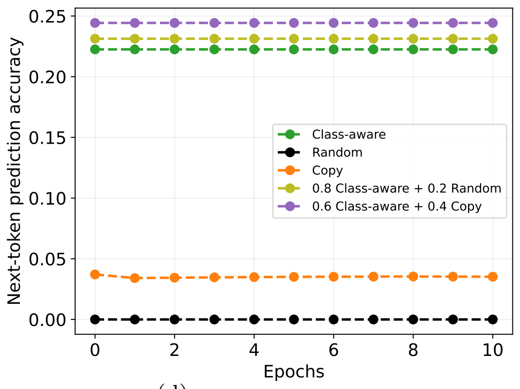
(a) OPT-125M, No freezing



(b) OPT-125M, Freezing



(c) TinyLlama-1.1B, No freezing



(d) TinyLlama-1.1B, Freezing

Figure 24. Next-token prediction accuracies on BookCorpus for the early exit LM head initialization techniques.

CHAPTER 6

FUTURE WORK

We outline some promising research directions that could further reduce inference and training cost of deep learning models.

Given the overhead of E^2CM, it becomes prohibitive for the settings where the number of classes is large. Therefore, it can be extended such that the overhead is reduced, possibly with pooling layers and/or weight/activation quantization techniques. Doing so would lead to a further decrease in inference costs.

Similar to combination of early exit networks with pruning, early exit networks can also be combined with other inference acceleration techniques such as weight/activation quantization and knowledge distillation. The correct combination recipe is not obvious and requires extensive experimentation on a variety of tasks.

CBT can be extended to multimodal data, which inherently have different requirements for processing complexity. In particular, the thresholds used for a multimodal neural network classifying text data should be different than the thresholds used for the image data.

In order to reduce the training costs, EEPrune may be extended to unsupervised learning tasks as the amount of unlabeled data is typically much greater than the amount of labeled data. It is possible to modify the conditions of EEPrune to make it applicable to unsupervised learning tasks, opening up new avenues for cost-efficient training. One other potential modification draws inspiration from contrastive learning and data augmentation techniques [131]. Similar to how contrastive learning generates positive pairs from augmented samples, EEPrune could be adapted to discard a sample from the dataset if a specific set of augmented versions of the sample also satisfy the pruning conditions.

Class-aware early exit LLM initialization can be improved further to accelerate not only pre-training, but also supervised fine-tuning and reinforcement learning from human feedback stages too. Moreover, it can be made quantization-friendly so that large LLMs initialized in a class-aware manner can be deployed to consumer GPUs with limited memories.

CHAPTER 7

CONCLUSION

We proposed E^{2}CM, a novel early exit mechanism based on the class means. Unlike existing early exit mechanisms, our method does not modify the base model and does not require gradient-based training, which makes it useful for network training on low-power devices. Under fixed training time budget, our method outperforms existing early exit schemes. In addition, combining our method with existing early exit techniques achieve better trade-off between the computational cost and the network accuracy. Moreover, we showed that our method is not only useful in supervised learning tasks, but also in unsupervised learning tasks.

We considered the problem of pruning early exit architectures. We evaluated the performance of two strategies in particular. First, intermediate classifiers are pruned jointly with the base network. Second, the base network is pruned first, followed by the intermediate classifiers. Although the former strategy outperforms the latter in general, the performance of the two strategies are close at high accuracy rates. Therefore, the processes of pruning and early exit can potentially be separated without significant penalty in performance.

We proposed CBT, which utilizes the naturally occurring neural collapse phenomenon to reduce the computational cost of early exit semantic segmentation models. Experiment results on different datasets and models suggest our method is effective in reducing the computational cost without significant penalty in model performance.

We presented EEPrune, a dataset pruning algorithm that utilizes the inherent ability of early exit networks to distinguish easy samples from the difficult ones. We showed that EEPrune achieves superior performance across various pruning rates as compared to existing approaches. We noted that EEPrune and variants do not need a full training session to identify the easy samples, unlike competing algorithms. This makes them valuable for resource constrained settings.

We developed a novel class-aware weight initialization technique for early exit LLMs based on mean representation of tokens. We made connections to the optimal detection problem for the vector AWGN channel from the digital communications domain. Our method performs better than baselines in both “no freezing” and “freezing” settings. We showed the applicability of our method to various model families and datasets, and its effectiveness on accelerating the pre-training phase.

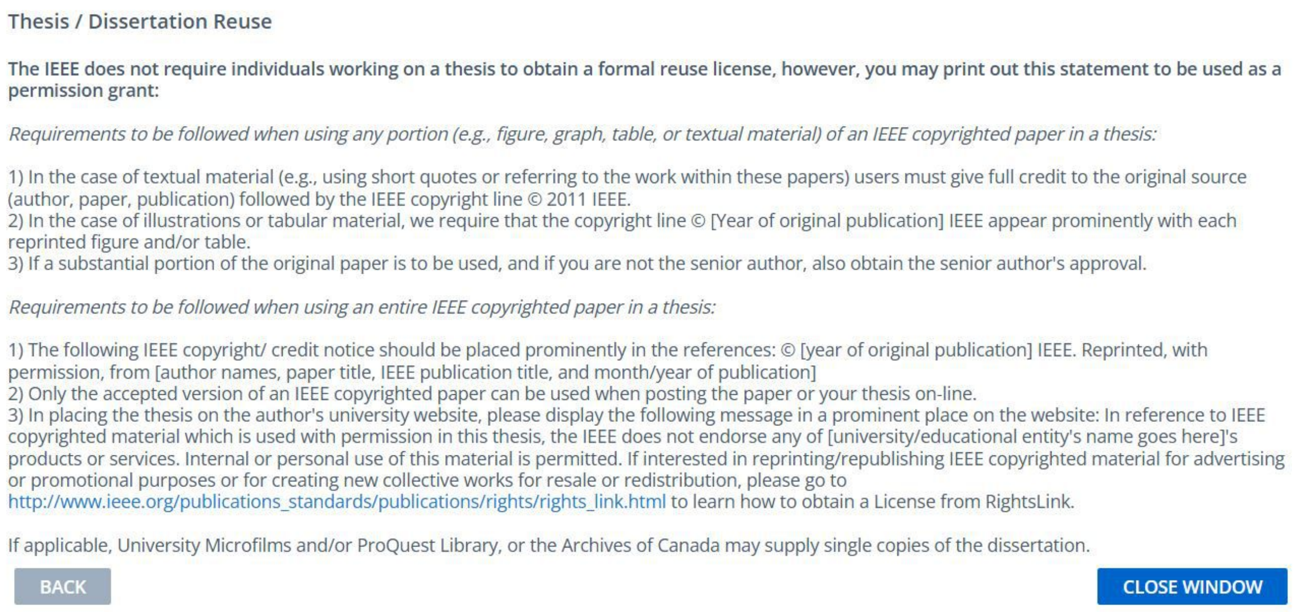
APPENDICES

Appendix A

COPYRIGHT PERMISSIONS

In this appendix, we present the copyright permissions for the articles, whose contents were used in this thesis.

Appendix A (Continued)



Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

CITED LITERATURE

1. Liu, Z., Xu, Z., Wang, H.-J., Darrell, T., and Shelhamer, E.: Anytime dense prediction with confidence adaptivity. International Conference on Learning Representations (ICLR), 2022.

2. Tang, Q., Zhang, B., Liu, J., Liu, F., and Liu, Y.: Dynamic token pruning in plain vision transformers for semantic segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 777–786, 2023.

3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A.: Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

4. He, K., Zhang, X., Ren, S., and Sun, J.: Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

6. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.

7. Li, P., Koyuncu, E., and Seferoglu, H.: ResPipe: Resilient model-distributed DNN training at edge networks. In IEEE Intl. Conf. Acoustics Speech Signal Process. (ICASSP), June 2021.

8. Li, P., Seferoglu, H., Dasari, V. R., and Koyuncu, E.: Model-distributed dnn training for memory-constrained edge computing devices. In 2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 1–6. IEEE, 2021.

9. Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V.: Adaptive neural networks for efficient inference. In International Conference on Machine Learning, pages 527–536. PMLR, 2017.

10. Panda, P., Sengupta, A., and Roy, K.: Conditional deep learning for energy-efficient and enhanced pattern recognition. In 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 475–480. IEEE, 2016.

11. Teerapittayanon, S., McDanel, B., and Kung, H.-T.: Branchynet: Fast inference via early exiting from deep neural networks. In 2016 23rd International Conference on Pattern Recognition (ICPR), pages 2464–2469. IEEE, 2016.

12. Kaya, Y., Hong, S., and Dumitras, T.: Shallow-deep networks: Understanding and mitigating network overthinking. In International Conference on Machine Learning, pages 3301–3310. PMLR, 2019.

13. Biasielli, M., Bolchini, C., Cassano, L., Koyuncu, E., and Miele, A.: A neural network based fault management scheme for reliable image processing. IEEE Transactions on Computers, 69(5):764–776, 2020.

14. Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F.: Bert loses patience: Fast and robust inference with early exit. arXiv preprint arXiv:2006.04152, 2020.

15. Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J.: Deebert: Dynamic early exiting for accelerating bert inference. arXiv preprint arXiv:2004.12993, 2020.

16. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. 2009.

17. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255, 2009.

18. Le, Y. and Yang, X.: Tiny imagenet visual recognition challenge. CS 231N, 7(7):3, 2015.

19. Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D.: Deep learning for classical japanese literature, 2018.

20. Xiao, H., Rasul, K., and Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

21. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

22. Zagoruyko, S. and Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.

23. Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1314–1324, 2019.

24. Tan, M. and Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning, pages 6105–6114. PMLR, 2019.

25. Xie, J., Girshick, R., and Farhadi, A.: Unsupervised deep embedding for clustering analysis. In International conference on machine learning, pages 478–487. PMLR, 2016.

26. Bengio, Y., Léonard, N., and Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432, 2013.

27. Bengio, E., Bacon, P.-L., Pineau, J., and Precup, D.: Conditional computation in neural networks for faster models. arXiv preprint arXiv:1511.06297, 2015.

28. Veit, A. and Belongie, S.: Convolutional networks with adaptive inference graphs. In Proceedings of the European Conference on Computer Vision (ECCV), pages 3–18, 2018.

29. Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., and Gonzalez, J. E.: Skipnet: Learning dynamic routing in convolutional networks. In Proceedings of the European Conference on Computer Vision (ECCV), pages 409–424, 2018.

30. Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q.: Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844, 2017.

31. Yang, L., Han, Y., Chen, X., Song, S., Dai, J., and Huang, G.: Resolution adaptive networks for efficient inference. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2369–2378, 2020.

32. McGill, M. and Perona, P.: Deciding how to decide: Dynamic routing in artificial neural networks. In International Conference on Machine Learning, pages 2363–2372. PMLR, 2017.

33. Koch, G., Zemel, R., and Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In ICML deep learning workshop, volume 2. Lille, 2015.

34. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D.: Matching networks for one shot learning. arXiv preprint arXiv:1606.04080, 2016.

35. Snell, J., Swersky, K., and Zemel, R. S.: Prototypical networks for few-shot learning. arXiv preprint arXiv:1703.05175, 2017.

36. Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M.: Learning to compare: Relation network for few-shot learning. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1199–1208, 2018.

37. Pan, Y., Yao, T., Li, Y., Wang, Y., Ngo, C.-W., and Mei, T.: Transferrable prototypical networks for unsupervised domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2239–2247, 2019.

38. Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B.: A closer look at few-shot classification, 2020.

39. Papyan, V., Han, X., and Donoho, D. L.: Prevalence of neural collapse during the terminal phase of deep learning training. Proceedings of the National Academy of Sciences, 117(40):24652–24663, 2020.

40. Cohen, U., Chung, S., Lee, D. D., and Sompolinsky, H.: Separability and geometry of object manifolds in deep neural networks. Nature communications, 11(1):1–13, 2020.

41. Zarka, J., Guth, F., and Mallat, S.: Separation and concentration in deep networks. arXiv preprint arXiv:2012.10424, 2020.

42. Wu, J., Leng, C., Wang, Y., Hu, Q., and Cheng, J.: Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4820–4828, 2016.

43. Han, S., Mao, H., and Dally, W. J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

44. Görmez, A., Dasari, V. R., and Koyuncu, E.: E2cm: Early exit via class means for efficient supervised and unsupervised learning. In 2022 International Joint Conference

on Neural Networks (IJCNN), pages 1–8, 2022. 2022 © IEEE. Reprinted with permission from the authors.

45. Sevilla, J., Heim, L., Ho, A., Besiroglu, T., Hobbhahn, M., and Villalobos, P.: Compute trends across three eras of machine learning. In 2022 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2022.

46. Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al.: Opt: Open pre-trained transformer language models. arXiv preprint arXiv:2205.01068, 2022.

47. Li, P., Seferoglu, H., and Koyuncu, E.: Model distributed inference in multi-source edge networks. In IEEE ICASSP Workshop on Timely and Private Machine Learning over Networks, June 2023.

48. Li, P., Koyuncu, E., and Seferoglu, H.: Adaptive and resilient model-distributed inference in edge computing systems. IEEE Open Journal of the Communications Society, 4:1263–1273, 2023.

49. Koyuncu, E.: Memorization capacity of neural networks with conditional computation. In International Conference on Learning Representations, 2023.

50. Görmez, A. and Koyuncu, E.: Pruning early exit networks. In Sparsity in Neural Networks, 2022.

51. Görmez, A. and Koyuncu, E.: Dataset pruning using early exit networks. In ICML Localized Learning Workshop, July 2023.

52. Hui, L., Belkin, M., and Nakkiran, P.: Limitations of neural collapse for understanding generalization in deep learning. arXiv preprint arXiv:2202.08384, 2022.

53. Li, G., Li, L., and Zhang, J.: Hierarchical semantic broadcasting network for real-time semantic segmentation. IEEE Signal Processing Letters, 29:309–313, 2021.

54. Zhang, G., Xue, J.-H., Xie, P., Yang, S., and Wang, G.: Non-local aggregation for rgb-d semantic segmentation. IEEE Signal Processing Letters, 28:658–662, 2021.

55. Huang, Y., Tang, Z., Chen, D., Su, K., and Chen, C.: Batching soft iou for training semantic segmentation networks. IEEE Signal Processing Letters, 27:66–70, 2019.

56. Li, Y., Li, X., Xiao, C., Li, H., and Zhang, W.: Eacnet: Enhanced asymmetric convolution for real-time semantic segmentation. IEEE signal processing letters, 28:234–238, 2021.

57. Li, G., Li, L., and Zhang, J.: Biattnnet: bilateral attention for improving real-time semantic segmentation. IEEE Signal Processing Letters, 29:46–50, 2021.

58. Yue, Y., Zhou, W., Lei, J., and Yu, L.: Two-stage cascaded decoder for semantic segmentation of rgb-d images. IEEE Signal Processing Letters, 28:1115–1119, 2021.

59. Yang, E., Zhou, W., Qian, X., and Yu, L.: Mgcnet: Multilevel gated collaborative network for rgb-d semantic segmentation of indoor scene. IEEE Signal Processing Letters, 29:2567–2571, 2022.

60. Krešo, I., Krapac, J., and Šegvić, S.: Efficient ladder-style densenets for semantic segmentation of large images. IEEE Transactions on Intelligent Transportation Systems, 22(8):4951–4961, 2020.

61. Cheng, B., Misra, I., Schwing, A. G., Kirillov, A., and Girdhar, R.: Masked-attention mask transformer for universal image segmentation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 1290–1299, 2022.

62. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In IEEE CVPR, June 2016.

63. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A.: Scene parsing through ade20k dataset. In IEEE CVPR, 2017.

64. Caesar, H., Uijlings, J., and Ferrari, V.: Coco-stuff: Thing and stuff classes in context. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1209–1218, 2018.

65. Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al.: Deep high-resolution representation learning for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 43(10):3349–3364, 2020.

66. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is

worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

67. Kouris, A., Venieris, S. I., Laskaridis, S., and Lane, N.: Multi-exit semantic segmentation networks. In European Conference on Computer Vision, pages 330–349. Springer, 2022.

68. Jung, S., Lee, J., Gwak, D., Choi, S., and Choo, J.: Standardized max logits: A simple yet effective approach for identifying unexpected road obstacles in urban-scene segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 15425–15434, 2021.

69. Yang, L., Jiang, H., Cai, R., Wang, Y., Song, S., Huang, G., and Tian, Q.: Condensenet v2: Sparse feature reactivation for deep networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3569–3578, 2021.

70. Phuong, M. and Lampert, C. H.: Distillation-based training for multi-exit architectures. In Proceedings of the IEEE/CVF international conference on computer vision, pages 1355–1364, 2019.

71. Zhang, D., Zhang, H., Tang, J., Hua, X.-S., and Sun, Q.: Self-regulation for semantic segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 6953–6963, 2021.

72. Bai, S., Koltun, V., and Kolter, J. Z.: Multiscale deep equilibrium models. Advances in Neural Information Processing Systems, 33:5238–5250, 2020.

73. Liu, K., Li, Y., Xu, N., and Natarajan, P.: Learn to combine modalities in multimodal deep learning. arXiv preprint arXiv:1805.11730, 2018.

74. Shervedani, A. M., Li, S., Monaikul, N., Abbasi, B., Di Eugenio, B., and Zefran, M.: An end-to-end human simulator for task-oriented multimodal human-robot collaboration. arXiv preprint arXiv:2304.00584, 2023.

75. Chen, X., Wang, X., Beyer, L., Kolesnikov, A., Wu, J., Voigtlaender, P., Mustafa, B., Goodman, S., Alabdulmohsin, I., Padlewski, P., et al.: Pali-3 vision language models: Smaller, faster, stronger. arXiv preprint arXiv:2310.09199, 2023.

76. Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y.: Multimodal deep learning. In Proceedings of the 28th international conference on machine learning (ICML-11), pages 689–696, 2011.

77. Sutton, R.: The bitter lesson. Incomplete Ideas (blog), 13(1), 2019.

78. OpenAI: Gpt-4 technical report, 2023.

79. Thoppilan, R., Freitas, D. D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H., Jin, A., Bos, T., Baker, L., Du, Y., Li, Y., Lee, H., Zheng, H. S., Ghafouri, A., Menegali, M., Huang, Y., Krikun, M., Lepikhin, D., Qin, J., Chen, D., Xu, Y., Chen, Z., Roberts, A., Bosma, M., Zhou, Y., Chang, C., Krivokon, I., Rusch, W., Pickett, M., Meier-Hellstern, K. S., Morris, M. R., Doshi, T., Santos, R. D., Duke, T., Soraker, J., Zevenbergen, B., Prabhakaran, V., Diaz, M., Hutchinson, B., Olson, K., Molina, A., Hoffman-John, E., Lee, J., Aroyo, L., Rajakumar, R., Butryna, A., Lamm, M., Kuzmina, V., Fenton, J., Cohen, A., Bernstein, R., Kurzweil, R., Aguera-Arcas, B., Cui, C., Croak, M., Chi, E. H., and Le, Q.: Lamda: Language models for dialog applications. CoRR, abs/2201.08239, 2022.

80. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N.: Palm: Scaling language modeling with pathways, 2022.

81. Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A., Caron, M., Geirhos, R., Alabdulmohsin, I., Jenatton, R., Beyer, L., Tschannen, M., Arnab, A., Wang, X., Riquelme, C., Minderer, M., Puigcerver, J., Evci, U., Kumar, M., van Steenkiste, S., Elsayed, G. F., Mahendran, A., Yu, F., Oliver, A., Huot, F., Bastings, J., Collier, M. P., Gritsenko, A., Birodkar, V., Vasconcelos, C., Tay, Y., Mensink, T., Kolesnikov, A., Pavetić, F., Tran, D., Kipf, T., Lučić, M., Zhai, X., Keysers, D., Harmsen, J., and Houlsby, N.: Scaling vision transformers to 22 billion parameters, 2023.

82. Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., Schramowski, P., Kundurthy, S., Crowson, K., Schmidt, L., Kaczmarczyk, R., and Jitsev, J.: Laion-5b: An open large-scale dataset for training next generation image-text models, 2022.

83. Paul, M., Ganguli, S., and Dziugaite, G. K.: Deep learning on a data diet: Finding important examples early in training. Advances in Neural Information Processing Systems, 34:20596–20607, 2021.

84. Toneva, M., Sordoni, A., Combes, R. T. d., Trischler, A., Bengio, Y., and Gordon, G. J.: An empirical study of example forgetting during deep neural network learning. arXiv preprint arXiv:1812.05159, 2018.

85. Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M.: Selection via proxy: Efficient data selection for deep learning. arXiv preprint arXiv:1906.11829, 2019.

86. Nohyun, K., Choi, H., and Chung, H. W.: Data valuation without training of a model. In The Eleventh International Conference on Learning Representations, 2023.

87. Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., and Morcos, A.: Beyond neural scaling laws: beating power law scaling via data pruning. Advances in Neural Information Processing Systems, 35:19523–19536, 2022.

88. Miao, R. and Koyuncu, E.: Resource-efficient federated clustering with past negatives pool. In 2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW), pages 1–5, 2023.

89. Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y.: Dynamic neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(11):7436–7456, 2021.

90. Scardapane, S., Scarpiniti, M., Baccarelli, E., and Uncini, A.: Why should we add early exits to neural networks? Cognitive Computation, 12(5):954–966, 2020.

91. Baccarelli, E., Scardapane, S., Scarpiniti, M., Momenzadeh, A., and Uncini, A.: Optimized training and scalable implementation of conditional deep neural networks with early exits for fog-supported iot applications. Information Sciences, 521:107–143, 2020.

92. Li, Y., Geller, T., Kim, Y., and Panda, P.: Seenn: Towards temporal spiking early exit neural networks. In Advances in Neural Information Processing Systems, 2023.

93. Feldman, D. and Langberg, M.: A unified framework for approximating and clustering data. In Proceedings of the forty-third annual ACM symposium on Theory of computing, pages 569–578, 2011.

94. Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R.: Geometric approximation via coresets. Combinatorial and computational geometry, 52(1):1–30, 2005.

95. Settles, B.: Active learning literature survey. 2009.

96. Sener, O. and Savarese, S.: Active learning for convolutional neural networks: A core-set approach. arXiv preprint arXiv:1708.00489, 2017.

97. Wang, T., Zhu, J.-Y., Torralba, A., and Efros, A. A.: Dataset distillation. arXiv preprint arXiv:1811.10959, 2018.

98. Tan, M. and Le, Q.: Efficientnetv2: Smaller models and faster training. In International conference on machine learning, pages 10096–10106. PMLR, 2021.

99. Müller, R., Kornblith, S., and Hinton, G. E.: When does label smoothing help? Advances in neural information processing systems, 32, 2019.

100. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

101. Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D.: mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412, 2017.

102. Anthony, L. F. W., Kanding, B., and Selvan, R.: Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems, July 2020. arXiv:2007.03051.

103. Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., and Tu, Z.: Deeply-supervised nets. In Artificial intelligence and statistics, pages 562–570. PMLR, 2015.

104. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen,

M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T.: Llama 2: Open foundation and fine-tuned chat models, 2023.

105. Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E.: Mixtral of experts, 2024.

106. Gemini Team: Gemini: A family of highly capable multimodal models, 2024.

107. Meta: Introducing meta llama 3: The most capable openly available llm to date, 2024.

108. Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D.: Confident adaptive language modeling. Advances in Neural Information Processing Systems, 35:17456–17472, 2022.

109. Del Corro, L., Del Giorno, A., Agarwal, S., Yu, B., Awadallah, A., and Mukherjee, S.: Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. arXiv preprint arXiv:2307.02628, 2023.

110. Bae, S., Ko, J., Song, H., and Yun, S.-Y.: Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. arXiv preprint arXiv:2310.05424, 2023.

111. Zhu, Y., Yang, X., Wu, Y., and Zhang, W.: Hierarchical skip decoding for efficient autoregressive text generation. arXiv preprint arXiv:2403.14919, 2024.

112. Görmez, A. and Koyuncu, E.: Class based thresholding in early exit semantic segmentation networks. IEEE Signal Processing Letters, 31:1184–1188, 2024. 2024 © IEEE. Reprinted with permission from the authors.

113. Merity, S., Xiong, C., Bradbury, J., and Socher, R.: Pointer sentinel mixture models, 2016.

114. Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S.: Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In The IEEE International Conference on Computer Vision (ICCV), December 2015.

115. Zhang, P., Zeng, G., Wang, T., and Lu, W.: Tinyllama: An open-source small language model, 2024.

116. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

117. Xin, J., Tang, R., Yu, Y., and Lin, J.: Berxit: Early exiting for bert with better fine-tuning and extension to regression. In Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume, pages 91–104, 2021.

118. Zhu, W.: Leebert: Learned early exit for bert with cross-level optimization. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 2968–2980, 2021.

119. Elbayad, M., Gu, J., Grave, E., and Auli, M.: Depth-adaptive transformer. arXiv preprint arXiv:1910.10073, 2019.

120. Liu, Y., Meng, F., Zhou, J., Chen, Y., and Xu, J.: Faster depth-adaptive transformers. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 13424–13432, 2021.

121. Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W.: Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

122. Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. A.: Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. Advances in Neural Information Processing Systems, 35:1950–1965, 2022.

123. Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T.: Adaptive budget allocation for parameter-efficient fine-tuning. In The Eleventh International Conference on Learning Representations, 2023.

124. Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L.: Qlora: Efficient finetuning of quantized llms. Advances in Neural Information Processing Systems, 36, 2024.

125. Chen, Y., Pan, X., Li, Y., Ding, B., and Zhou, J.: Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. arXiv preprint arXiv:2312.04916, 2023.

126. Pan, X., Chen, Y., Li, Y., Ding, B., and Zhou, J.: Ee-tuning: An economical yet scalable solution for tuning early-exit large language models, 2024.

127. Proakis, J. G. and Salehi, M.: Digital communications. McGraw-hill, 2008.

128. Laskaridis, S., Kouris, A., and Lane, N. D.: Adaptive inference through early-exit networks: Design, challenges and directions. In Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning, pages 1–6, 2021.

129. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: Pytorch: An imperative style, high-performance deep learning library, 2019.

130. Gormez, A. and Koyuncu, E.: Class-aware initialization of early exits for pre-training large language models. In 2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024).

131. Chen, T., Kornblith, S., Norouzi, M., and Hinton, G.: A simple framework for contrastive learning of visual representations. In International conference on machine learning, pages 1597–1607. PMLR, 2020.

VITA

NAME	Alperen Görmez
EDUCATION	**Ph.D., Electrical and Computer Engineering,** University of Illinois Chicago, Chicago, IL. 2019 - 2024
	B.S., Electrical and Electronics Engineering, Bilkent University, Ankara, TURKEY. 2015 - 2019
EXPERIENCE	Research & Teaching Assistant at University of Illinois Chicago, 2019 - 2024.
	Research Intern at Google, May 2024 - Aug 2024.
	AIML Intern at Apple, May 2023 - Aug 2023.
	Machine Learning Intern at Roku, May 2021 - Aug 2021.
	Candidate Engineer at ASELSAN, Feb 2019 - Jun 2019.
PUBLICATIONS	
	A. Görmez and E. Koyuncu, "Class-aware Initialization of Early Exits for Pre-training Large Language Models," in *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization*, 2024.
	A. Görmez and E. Koyuncu, "Class Based Thresholding in Early Exit Semantic Segmentation Networks," in *IEEE Signal Processing Letters*, vol. 31, pp. 1184-1188, 2024.
	A. Görmez and E. Koyuncu, “Dataset Pruning Using Early Exit Networks,” *ICML Workshop on Localized Learning (LLW)*, 2023.

A. Görmez and E. Koyuncu, “Pruning Early Exit Networks,” *2022 Sparsity in Neural Networks*, 2022.

A. Görmez, V. R. Dasari and E. Koyuncu, “E2CM: Early Exit via Class Means for Efficient Supervised and Unsupervised Learning,” *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1-8.

PRESENTATIONS

Invited Talks

2023 Cohere For AI ML Efficiency

Conference Presentations

2024 ICML Workshop on Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization

2024 IEEE International Workshop on Machine Learning for Signal Processing

2023 ICML Workshop on Localized Learning

2022 International Joint Conference on Neural Networks

Poster Presentations

2023 Mediterranean Machine Learning Summer School

2022 Sparsity in Neural Networks Workshop

2022 Eastern European Machine Learning Summer School

AWARDS

IEEE Computational Intelligence Society Travel Grant: Received a travel grant to attend IEEE WCCI 2022.

Eastern European Machine Learning Summer School 2022: Received the top-voted poster award for E^2CM.

Bilkent University Comprehensive Scholarship: Full tuition waiver and stipend during the B.S. program, 2015 - 2019.

SERVICES

IEEE Signal Processing Letters, 2024.

International Conference on Learning Representations (ICLR), 2024.

IEEE Transactions on Signal and Information Processing over Networks, 2024.

IEEE Global Communications Conference (GLOBECOM), 2023.

Lacmus Foundation, 2022-2023.

Deep Learning Indaba, 2021-2023.

IEEE Transactions on Computational Imaging, 2022.

IEEE International Conference on Network Protocols (ICNP), 2019.