EARLY-EXIT CONVOLUTIONAL NEURAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EDANUR DEMIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2019

Approval of the thesis:

EARLY-EXIT CONVOLUTIONAL NEURAL NETWORKS

submitted by **EDANUR DEMIR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Emre Akbaş
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assist. Prof. Dr. Gökberk Cinbiş
Computer Engineering, METU _____

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Assist. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Edanur Demir

Signature :

ABSTRACT

EARLY-EXIT CONVOLUTIONAL NEURAL NETWORKS

Demir, Edanur
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Emre Akbaş

January 2019, 53 pages

This thesis is aimed at developing a method that reduces the computational cost of convolutional neural networks (CNN) during inference. Conventionally, the input data pass through a fixed neural network architecture. However, easy examples can be classified at early stages of processing and conventional networks do not take this into account. In this thesis, we introduce "Early-exit CNNs", *EENets* for short, which adapt their computational cost based on the input by stopping the inference process at certain exit locations. In EENets, there are a number of exit blocks each of which consists of a confidence branch and a softmax branch. The confidence branch computes the confidence score of exiting (i.e. stopping the inference process) at that location; while the softmax branch outputs a classification probability vector. Both branches are learnable and they are independent of each other. During training of EENets, in addition to the classical classification loss, the computational cost of inference is taken into account as well. As a result, the network adapts its many confidence branches to the inputs so that less computation is spent for easy examples. Inference works as in conventional feed-forward networks, however, when the output of a confidence branch is larger than a certain threshold, the inference stops for that specific

example. Through comprehensive experiments, we show that EENets significantly reduce the computational cost upto 2% of the original without degrading the testing accuracy. The idea of EENets is applicable to available CNN architectures such as ResNets. On MNIST, SVHN and CIFAR10 datasets, early-exit (EE) ResNets achieve similar accuracy with their non-EE versions while reducing the computational cost to 20% of the original.

Keywords: Deep Learning, Object classification, Adaptive Computation, Early Termination, Confidence for Classification

ÖZ

ERKEN-SONLANDIRMALI EVRİŞİMSEL SİNİR AĞLARI

Demir, Edanur
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Emre Akbaş

Ocak 2019 , 53 sayfa

Bu tez, evrişimsel yapay sinir ağlarının (CNN) kestirim sırasında harcadığı hesaplama maliyetini azaltan bir yöntem geliştirmeyi amaçlamaktadır. Geleneksel evrişimsel yapay sinir ağlarında, veriler sabit bir sinir ağı mimarisinden geçer. Aslında, tüm mimariden geçen kolay örnekler işlemenin erken aşamalarında sınıflandırılabilir ve geleneksel ağlar bu durumu dikkate almaz. Bu çalışmada, belirli çıkış noktalarında kestirim sürecini durdurup, hesaplama maliyetini girdiye dayalı olarak uyarlayan "Erken-sonlandırmalı Evrişimsel Yapay Sinir Ağlarını", kısaca *EENetleri* tanıtacağız. EENetlerde, her biri güvenirlik ve softmax çıkışlarından oluşan bir çok erken sonlandırma bloğu vardır. Güvenirlik çıkışı, herhangi bir sonlandırma noktasında modelden çıkmanın (yani kestirim sürecini durdurmanın) güven skorunu hesaplar; softmax çıkışı ise sınıflandırmanın olasılık vektörünü üretir. Her iki çıkış dalı da öğrenilebilir ve birbirinden bağımsızdır. EENetlerin eğitimi sırasında, klasik sınıflandırma kaybına ek olarak, sonuç çıkarmanın hesaplama maliyeti de dikkate alınır. Sonuç olarak, önerdiğimiz sinir ağı güvenirlik çıkışlarında girdileri işler ve sonlandırabilir, böylece kolay örnekler için daha az hesaplama yapılır. Kestirim, geleneksel ağlardaki ileri besleme gibi çalışır, ancak bir güvenirlik çıkış dalındaki güven skoru belirli bir eşikten daha

büyük olduğunda, kestirim bu örnek için tamamlanır ve hesaplama sonlandırılır. Kapsamlı deneyler sayesinde, EENetlerin test doğruluğuna zarar vermeden hesaplama maliyetini orjinal modelin %2'sine kadar azalttığını gösteriyoruz. EENet fikri, ResNetler gibi mevcut CNN mimarilerine uygulanabilir. MNIST, SVHN ve CIFAR10 veri setlerinde, erken çıkışlı (EE) ResNetler, EE olmayan sürümleriyle benzer bir doğruluk elde ederken hesaplama maliyetini orjinal modelin %20'sine kadar azalttığı görülmüştür.

Anahtar Kelimeler: Derin Öğrenme, Nesnelerin Sınıflandırılması, Uyumlu Hesaplama, Erken Sonlandırma, Sınıflandırma Güvenirliği

To my lovely family and friends Tuba Nur and Büşra,

ACKNOWLEDGMENTS

I would first like to thank my advisor Dr. Emre Akbaş with my deepest appreciation and gratitude. He was always there and steered me in the right direction whenever I ran into a trouble spot or had a question. This study would not have been possible without his continuous support, guidance and motivation.

I would like to thank my thesis committee members Dr. Gökberk Cinbiş and Dr. Mehmet Tan for their valuable feedback.

I would like to thank Muhammed Kocabaş and Ufuk Ertenli for their help.

I would like to thank my team leader Dr. Arif Yılmaz and deep learning project leader Dr. Ersin Esen for their valuable comments on this thesis.

I would like to thank my team members and colleagues, especially Büşra Güvenen, Nurel Polat, Şeyma Arslan and Tuba Nur Oğuztürk for all their help and contribution to this work.

I would like to thank to my lovely family and friends for their continuous supports.

I would like to express my deepest gratitude to all of my teachers and specially the first teacher, my mother, for bringing me up and steering me in the right direction.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ANT	Adaptive Neural Tree
CDL	Conditional Deep Learning
CIFAR10	Labeled Subsets of the 80 Million Tiny Images Dataset
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DT	Decision Tree
EB	Exit Block
EE-block	Early-exit Block of Early-exit Convolutional Neural Network
FC	Fully-connected Layer of Neural Networks
FLOP	Floating Point Operation
MNIST	The MNIST Database of Handwritten Digits
RNN	Recurrent Neural Networks
SACT	Spatially Adaptive Computation Time for Residual Networks
SGD	Stochastic Gradient Descent
SVHN	The Street View House Numbers Dataset

CHAPTER 1

INTRODUCTION

1.1 Problem Definition and Motivation

Deep neural networks are power-hungry. They typically need powerful processing units (i.e. GPU cards) in order to run in a reasonable amount of time. Reducing their computational cost with zero or minimal degradation in accuracy is an important goal that has been approached from several different directions. One promising way to this end is to make the network adapt its computational cost to the input during inference. This idea has recently been explored in many ways. Researches have proposed early termination networks [1] [2] [3] [4] [5], layer skipping networks [6] [7] [8], specialized branches with wide networks [9], adaptive neural trees [10], cascaded networks [11] and pruning methods such as channel gating networks [12].

Conventionally, the input data pass through a fixed neural network architecture. However, easy examples can be classified at early stages of processing and conventional networks do not take this account. In order to reduce the computational cost, the methods mentioned above aim to adapt the computation graph of the network to the characteristics of the input instead of running the fixed model that is agnostic to the input. Our work in this thesis can be categorized under the "early termination networks" category.

1.2 Proposed Method

In this thesis, we introduce "Early-exit CNNs", *EENets* for short, which adapt their computational cost based on the input itself by stopping the inference process at cer-

tain exit locations. Therefore, an input does not have to flow through all the layers of the fixed network; on the contrary, the computational cost can be significantly decreased based on the characteristics of inputs. Figure 1.1 shows the architectural overview of the Early-exit Convolutional Neural Networks.

In EENets, there are a number of exit blocks which consist of a confidence branch and a softmax classification branch. The confidence branch computes the confidence score of exiting (i.e. stopping the inference process) at that location; while the softmax branch outputs a classification probability vector. Both branches are learnable and they are independent of each other. These exit blocks can be built just by adding a small number of parameters ($\sim$0.0002% of the total parameters of EENet-110 for each early-exit block with 10-classes datasets). Thus, the additional parameters coming from early-exit blocks do not increase the computational cost and they can be ignored.

During training of EENets, in addition to the classical classification loss, the computational cost of inference is taken into account as well. As a result, the network adapts its confidence branches to the inputs so that less computation is spent for easy examples. Inference works as in conventional feed-forward networks, however, when the output of a confidence branch is larger than a certain threshold, the inference stops for that specific example.

Since the early termination is performed according to the trained confidence scores, it maintains the accuracy with a proper loss trade-off that will be explained in chapter 3. Instead, early-exit blocks can provide some kind of network regularization and the mitigation of vanishing gradients in back-propagation.

1.3 Challenges

Defining a proper way to train confidence scores is important because the model could be biased towards wrong decisions such as early or late termination. These wrong decisions either decrease accuracy or increase the computational cost unnecessarily. Deciding at which point an input can be classified and the execution can be terminated is the key challenge of the problem.

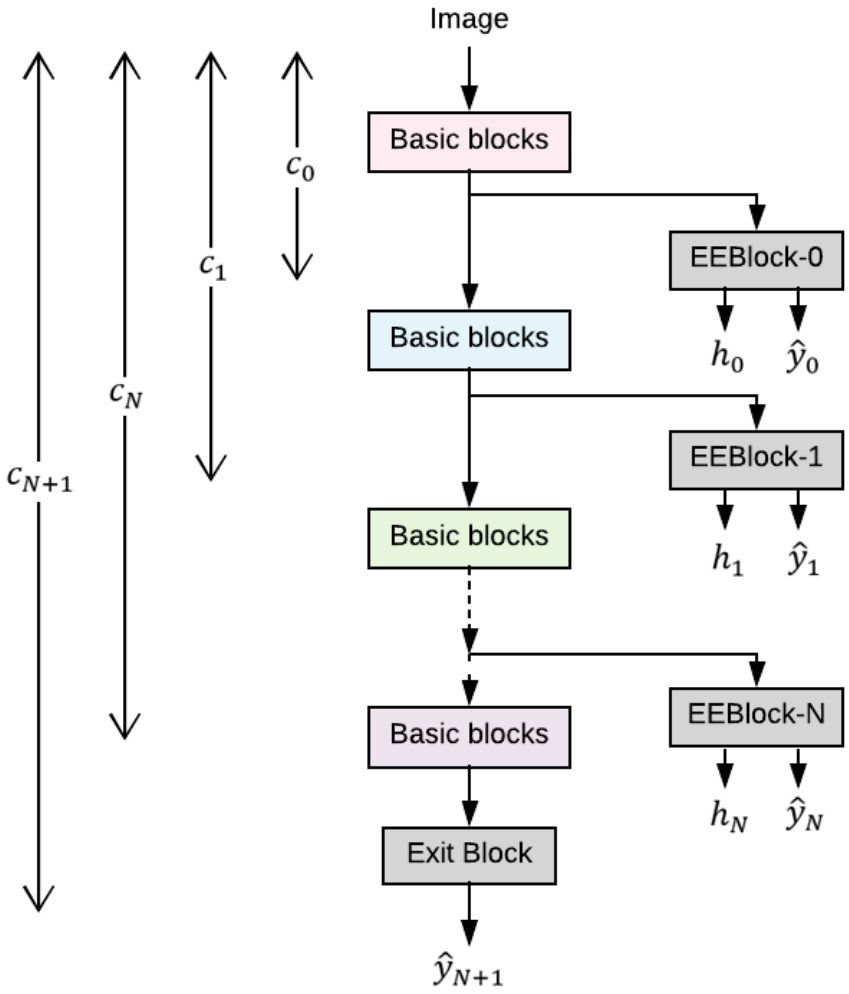

Figure 1.1: Architectural overview of EENets. An early-exit block (shown with gray color) can be added at any layer. If, at a certain early-exit block, say the i^{th} one, the network is sufficiently confident (i.e. $h_i > 0.5$), then the execution is terminated at that point and the network's output is set to $\hat{y}_i$. c_i denotes the computational cost (in terms of the total number of floating-point operations) upto the i^{th} early-exit block. Basic blocks are classical computation blocks that may be composed of one or more convolutional layers and non-linear activation functions. Their contents depend on the base network that is intended to be transformed into early-exit version. EENets aim to strike a balance between minimizing the computational cost and maximizing the accuracy.

We tackle this problem by making confidence scores learnable within our novel loss function which balances computational cost and classification loss in a single expression.

1.4 Contributions and Novelties

In comparison to the existing early termination networks, our contributions in this work are as follows:

- All exit blocks of an EENet are fed by all inputs even if some of them are classified in early stages of the model. This avoids a possible dead unit problem where some layers are not trained at all.
- The confidence scores of EENets are learnable and they do not depend on heuristic calculations. As a consequence, their initialization is not an issue and they can be initialized just like other parameters of the network.
- Our loss function considers both accuracy and cost simultaneously and provides a trade-off between them via an hyperparameter.
- EENet has a single stage training as opposed similar previous work which are trained in multiple stages.
- EENets are compact models not requiring additional hyper parameters such as non-termination penalty or confidence threshold variables.

By maintaining the same accuracy, EENets classify most of the examples early just by spending around 5-fold less computational cost in terms of floating-point operations (FLOPs) compared to their counterpart ResNets [13] on MNIST [14], SVHN [15] and CIFAR10 [16] datasets. In the models which have a large capacity such as EENet-152, the cost savings can be upto 50x. In such deep models, EENets not only maintain but also improve the accuracy a little probably due to the regularizing effect of using less parameters.

1.5 Outline of the Thesis

Chapter 2 discusses the related work on the area of adaptive computational networks. It emphasizes the differences between our novel model and them.

Chapter 3 describes the architecture of EENets. It includes the types of the early-exit blocks, how these blocks can be distributed to a network, feed-forward and backward phases of the model and our novel loss function. Other details about training and inference phases are also given in that chapter.

Chapter 4 exhibits the performances of EENets. The test environment, the architectures that we experimented with and the results on benchmark datasets with ResNets are given and demonstrated in that chapter.

Chapter 5 provides a brief summary and discussion.

CHAPTER 2

RELATED WORK

2.1 Introduction

In this chapter, we review related work in the area of adaptive computational networks, and discuss their differences and similarities with our model. Adaptive computational networks based on the input's characteristic can be examined in the following main categories: early termination networks, layer skipping networks, specialized branches with wide networks, neural trees, cascaded networks and pruning methods such as channel gating networks.

2.2 Early Termination Networks

Early termination network are based on the idea that it might not be necessary to run the whole network for some inputs. Similar to EENets, early termination networks [1], [2], [3], [4], [5] have multiple exit blocks that allow early termination based on an input's characteristics. All of these studies have some kind of confidence scores to decide early termination.

One of the early termination networks, BranchyNets [2] have multiple exit blocks each of which consists of a few convolutional layers followed by a classifier with softmax. In other words, BranchyNets have one head just for classification at their exit blocks. The exit blocks of Beretizshevsky and Even [3] are composed of pooling, two fully-connected (FCs) and batch normalization layers. Like BranchyNets, one conventional head at an exit block is trained for classification. The confidence scores are derived via some heuristics. In the training procedure of the model of Beretiz-

shevsky and Even, the weights of only convolutional and the last FC layers are firstly optimized. Later, the remaining FC layers are optimized, one by one. On the other hand, MSDNets [1] have multi-scaled features with dense connectivity among them. Exits of MSDNets consist of two convolutional layers followed by a pooling and a linear layer. However, similar to BranchyNets, MSDNets do not have confidence branches at their exit blocks.

In these studies, the confidence scores are derived from the predicted classification results (i.e. the maximum over the softmax). Because such confidence scores are not learnable, termination criteria or threshold of an exit branch is an important issue. The exit threshold providing the maximum accuracy should be empirically discovered in these models. Unlike EENets, the loss functions of these studies do not encourage an early-exit by considering the computational cost. In addition, they have a dead layer problem coming from improper initialization of the confidence scores in the training. The scores may be biased to exit always early and deeper layers may not take learning signals properly. To avoid that situation, all of BranchyNets, MSDNets and the model of Beretizshevsky and Even use a multi-stage training.

Spatially Adaptive Computation Time for Residual Networks, shortly SACTs [4], is another study in the area. Exit blocks of the model consist of a pooling and a fully-connected layer followed by a sigmoid function like our model. However, the final confidence score of early termination (namely halting score in the paper) is calculated by the cumulative learnable scores of the previous exit blocks. As soon as the cumulative halting score reaches a constant threshold (i.e. $T \geq 1.0$), the computation is terminated. Unlike EENets, the classification output vector of SACTs (i.e. the output of the softmax branch) is derived from weighted summation of the inputs of the confidence branches so far. While EENets directly train the confidence scores by taking them into account in the loss function, SACTs employ the number of executed layers as non-termination penalty in the loss function. Another work, Conditional Deep Learning (CDL) [5] has multiple exit blocks each of which consists of just a linear classifier. Starting from the first layer, linear classifiers are added to the end of each convolutional layer iteratively as long as this addition process does not decrease the accuracy. In CDL, user defined threshold is used to decide if the CDL is confident enough to exit. The training procedures of SACTs and CDLs are multi-stage.

2.3 Layer Skipping Networks

Layer skipping networks [6], [7], [8] adapt the computation to the input by selectively skipping layers. In these networks, a gating mechanism determines whether the execution of the layer is required for an example or whether it can be skipped. In this way, the execution can skip some set of the layers and jump on the effective ones. However, their main challenge is learning the discrete decisions of the gates. AdaNets [6] use Gumbel Sampling [17] while SkipNets [7] and BlockDrop [8] apply reinforcement to this end. None of the studies has a separate confidence branch at the gate blocks.

Similar to the early-exit blocks of early termination nets, the gates of the layer skipping networks may die and lose their functionality if they incline to be too much turned off during training. Thus, the actual capacity usage decreases. On the other hand, if the gates tend to be turned on, the networks can lose their cost diminishing effects on computation time. As a result, the networks can not only perform as counterpart static models but also spend additional computational cost for the gate functions (i.e. the same capacity with more cost). In order to avoid such cases, the gate blocks require to be initialized carefully and trained properly. Thus, the layer skipping networks have a complicated multi-stage training.

2.4 Specialized Branches with Wide Networks

As wide networks, HydraNets [9] are another approach in the area. HydraNets contain distinct branches specialized in visually similar classes. HydraNets possess a single gate and a combiner. The gate decides which branches to be executed at inference. And the combiner aggregates feature from multiple branches to make a final prediction. In training, given a subtask partitioning (i.e. dividing dataset into visually similar classes), the gate and the combiner of the HydraNets are trained jointly. The branches are indirectly supervised by the classification predictions after combining the features computed by the top-k branches.

2.5 Neural Trees

Adaptive Neural Trees, ANTs [10], can be considered as a combination of decision trees (DTs) with deep neural networks (DNNs). In other words, it includes the features of the conditional computation of DTs with the hierarchical representation learning and gradient descent optimization of DNNs. ANTs learn routing functions of a decision tree thanks to the training feature of DNNs. While doing this, instead of a classical entropy, ANTs use stochastic routing, where the binary decision is sampled from Bernoulli distribution with mean $r^{\theta}(x)$ for input x. As an example, r^{θ} can denote a small convolutional neural networks (CNNs). However, ANTs are trained in two stages: *growth phase* during which the model is trained based on local optimization and *refinement phase* which further tunes the parameters of the model based on global optimization. The training process of ANTs is complicated because of the refinement phase.

2.6 Cascaded Networks

Some other approaches focus on cascaded systems. The model of Bolukbasi *et al.* [11] adaptively chooses a deep network among the-state-of-arts such as AlexNet [18], GoogleNet [19], and ResNet [13] to be executed per example. Each convolutional layer is followed by the decision function to choose a network. But it is hard to decide if termination should be performed just by considering a convolutional layer without employing any classifier. It has a multi-stage training procedure where the gates are trained independently from the rest of the model.

2.7 Pruning Methods

Channel Gating Neural Networks [12] dynamically prune computation on subset of input channels. Based on the first p channels, a gate decides whether to mask the rest of the channels. Similar to SACTs [4], when the classification confidence score reaches some threshold, the remaining channels are not computed.

2.8 Contributions and Novelties

In the inference phase of adaptive computational networks, inputs pass onto the layers of model until the model reaches some confidence level for classification and early termination. However, in the training phase of most of these networks, feeding the model is also terminated when getting enough confidence for early termination. Consequently, the models may not be trained properly if the execution is frequently terminated at the intermediate exit blocks. On the other hand, all exit blocks of EENets are contributed by all inputs, even if some of them are classified in the early stages of the model. This compact training procedure also avoids the possible dead unit problem happened in the previous works.

Another contribution of EENets is the separate confidence branches at their exit blocks. Unlike most of the previous adaptive computational approaches, the confidence scores of EENets are trainable and do not depend on some heuristic calculations. Having separate learnable parameters allows the confidence branches to be not biased towards classification results. On the other hand, taking learning signals through our composite loss function (in back-propagation phase) trains them indirectly based on the classification accuracy as well. Since the confidence branches are not calculated by heuristics, their initialization is not an issue and they can be initialized just like other parameters of the network. This separate confidence branches approach allows EENets to be more flexible structures than the previous networks in the area.

Another novelty of our study is the loss function that takes both accuracy and the cost spending into account simultaneously and provides a trade-off between them through the confidence scores. Rather than most of the previous studies, our cost values employed in the loss function are not hyper-parameters and calculated as rates of actual floating-point operations. Unlike most of the previous studies, EENets have a single stage training in spite of multiple outputs of their exit blocks. It is a compact model not requiring additional hyper parameter such as non-termination penalty or confidence threshold variables.

Table 2.1 summarizes the differences between EENets and the related work.

Table 2.1: Differences with related work. The features of the related work are compared in terms of whether they have a single stage training (SST), a non-specialized initialization process (NSI) and learnable confidence scores (LCS) in the table below. Check mark represents whether the model has the feature or not. The last column shows what their loss functions include (e.g. the classical classification loss as the accuracy or the number of executed layers (# exec. layers) as the computational cost). The term of "accuracy and cost" just shows that the loss function takes both of them into account but note that the accuracy and cost values of different models can be obtained in different ways. Some features may not be applicable for some models. In such cases, we use "-" symbol.

Differences with related work

Model	**SST**	**NSI**	**LCS**	**Loss func. includes**
AdaNet [6]	✓	✗	✓	accuracy and # exec. layers
ANT [10]	✗	✗	-	accuracy
Beretizshevsky *et al.* [3]	✗	✗	✗	accuracy
BlockDrop [8]	✗	✗	✓	accuracy and cost
Bolukbasi *et al.* [11]	✗	✓	✓	accuracy and cost
BranchyNet [2]	✓	✗	✗	accuracy
CDL [5]	✗	✗	✗	accuracy and cost
Channel gating [12]	✗	✗	✓	accuracy and cost
HydraNet[9]	✓	✗	-	accuracy of top-k branches
MSDNet [1]	✓	✓	✗	accuracy of top-k classifier
SkipNet [7]	✗	✗	✗	accuracy and cost
SACT [4]	✗	✗	✓	accuracy and # exec. layers
EENet	✓	✓	✓	accuracy and cost

CHAPTER 3

THE MODEL

3.1 Introduction

In this chapter, the architecture of EENets is described. The chapter includes the details about types of exit blocks, how to distribute early-exit blocks to a network, feed-forward and backward phases of the model and the loss function.

3.2 Architecture

Any given convolutional neural network (CNN) can be converted to an early-exit network by adding early-exit blocks at desired locations. In this thesis, we explain EENets through the ResNet architectures [13] since they are widely used and they yield state-of-the-art results in many problems. ResNets have identity skip connections that bypass each layer, meaning the input to each layer is also added to its output. Therefore, gradients can propagate directly through the skip-connection, early layers still receive sufficient learning signal even in very deep networks. By this method, ultra-deep networks with over a thousand layers can be built easily [20], [21]. Since ResNets are deep networks, it is convenient to add multiple early-exit blocks to them.

Because we explain EENets through ResNets, the main flow of EENets can be considered being composed of the identity skip-connections and the layer functions such as convolution, pooling etc. Formally, consider $F_l(\cdot)$ and $\mathbf{x}_l$ as the function of the l^{th}

layer and the output of this function, respectively. The main flow can be defined as:

$$\mathbf{x}_l = \mathbf{x}_{l-1} + F_l(\mathbf{x}_{l-1}) \tag{1}$$

Beside the main network, the architecture of EENets has multiple early-exit (EE) blocks. Each EE-block consists of two fully-connected (FC) heads, namely confidence branch and conventional classification softmax branch. Both take channel-based feature maps as an input.

In order to employ in EENets, we define three types of early-exit blocks, namely *plain*, *pool* and *bnpool*. The *plain*-type exits are composed of just separate fully-connected (FC) layers and input of that block are directly used in the FC branches. The *pool* exits have a global average pooling layer before FC branches. Lastly, the *bnpool*-type exit blocks consist of a batch normalization layer followed by a ReLU activation and a global average pooling layer. After passing these layers, the data flows to the separate confidence and classification branches. Figure 3.1 shows the architecture types of the early-exit blocks.

In the *pool* and *bnpool* early-exit blocks, the size of feature map is reduced by global average pooling that is denoted by $z(\mathbf{x})$. The purpose of this is to reduce the computational cost at early-exit blocks. Consequently, they can decide to terminate the execution, in a shorter amount of time. The early-exit blocks that have a global average pooling layer provided more accurate results in experiments (will be given in Chapter 4). The following equations are constructed on the idea of *pool* early-exit blocks. The average pooling function can be described as:

$$z_{n,c}(\mathbf{x}) = \frac{1}{H * W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{x}_{n,c,i,j} \tag{2}$$

In Equation (2), n denotes the batch size and c denotes the number of channels. H and W denote height and width of the feature maps, respectively.

The pooled data passes onto two separate branches, namely classification branch and confidence branch. The number of outputs of the classification softmax branch is same as the number of classes in the dataset. This branch has a softmax activation

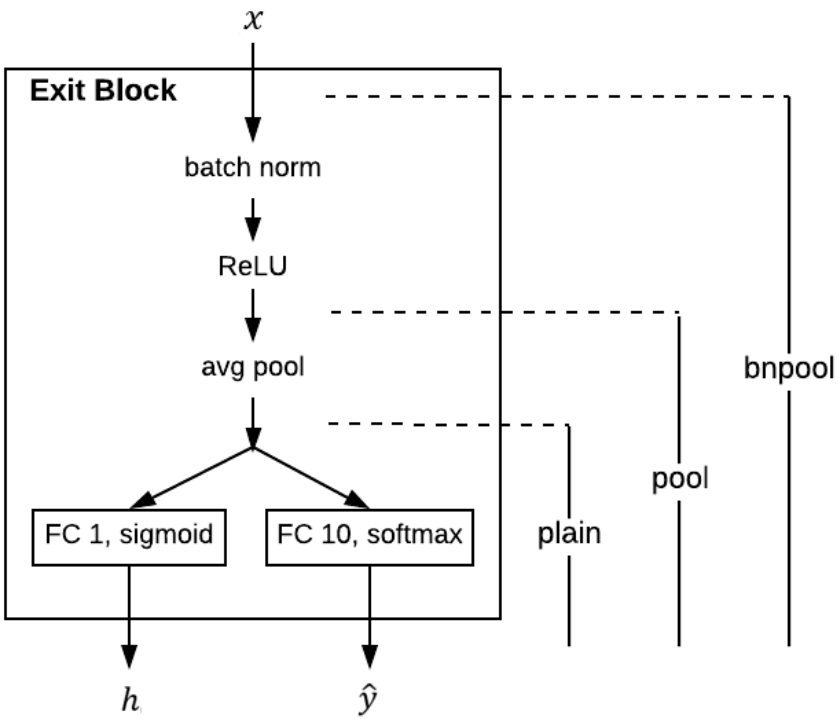

Figure 3.1: Architecture of *plain*, *pool*, and *bnpool* early-exit blocks. The *plain*-type exits are composed of just separate fully-connected (FC) layers and input of that block is directly processed in the FC branches. The *pool* exits have a global average pooling layer before FC branches. Lastly, the *bnpool*-type exit blocks consist of a batch normalization layer followed by a ReLU activation and a global average pooling layer. The input of the early-exit block, x, passes onto these layers before entering the separate FC branches. h and $\hat{y}_n$ denote the confidence score and the predicted classification label. "$\mathrm{fc}\ X, \mathrm{activation}$" denotes the fully-connected heads which have X number of outputs. The $\mathrm{activation}$ is the last activation function of branches.

in the end. The other branch, namely confidence, uses a sigmoid function as an activation function and it has a scalar output representing the confidence of the work at that specific exit block. Both branches (or output heads) feed the network jointly and are back-propagated in the training.

Formally, let $\mathbf{x}$ be the input to the n^{th} early-exit block. $\mathbf{x}$ is actually the output of the basic block (see Figure 1.1) immediately preceding the n^{th} early-exit (EE) block. In the EE-block, two things are computed: (i) $\hat{\mathbf{y}}_n$, the class prediction vector, and (ii) h_n, the confidence level of the network for the prediction $\hat{\mathbf{y}}_n$. They are given in Equation (3) where $\mathbf{w}_1$ and $\mathbf{w}_2$ are the parameters of separate fully-connected layers of the softmax and confidence branches, respectively.

$$\begin{aligned} \hat{\mathbf{y}}_n &= \mathrm{softmax}(\mathbf{w}_1^T z(\mathbf{x})) \\ h_n &= \sigma(\mathbf{w}_2^T z(\mathbf{x})) \end{aligned} \tag{3}$$

Note that the classification and the confidence branches have softmax and sigmoid activation functions, respectively. As a traditional multi-classification head, the classification branch of the early-exit blocks employs the softmax activation function. On the other hand, the confidence branch of EE-blocks uses the sigmoid activation function because it is expected to be estimate the probability of prediction correctness. The sigmoid function maps the output of the branch onto the interval $[0, 1]$. The confidence score of the last exit layer (note that it is not an early-exit block) is set to 1 in order to guarantee to terminate the execution at the end of the model.

3.3 Inference

The Early-exit Convolutional Neural Networks have a certain threshold in order to decide early termination in inference procedure. If the confidence score of an early-exit block is above the threshold, the classification results of the current stage will be the final prediction. Each input is classified based on their individual confidence scores predicted by the early-exit blocks. Thus, one input can be classified and terminated early while others continue the execution on the model.

Early termination threshold is $T = 0.5$. It is the midpoint of the confidence score since the result of the sigmoid function of confidence branches is a probability. The threshold is employed just in the inference phase. Because all early-exit blocks contribute the loss function for all examples (even if some of them can be classified early), the examples are executed through the whole model. Therefore, the threshold does not affect the training process. The pseudo-code of the inference procedure of EENets are given in Algorithm 1.

Algorithm 1 Inference of Early-exit Convolutional Neural Networks

$i \leftarrow 0$
while $i < N$ **do**
    $x \leftarrow \text{BasicBlocks}_i(x)$
    $h_i, \hat{y}_i \leftarrow \text{EEBlock}_i(x)$
    if $h_i \geq T$ **then**
        return $\hat{y}_i$
    end if
    $i \leftarrow i + 1$
end while
$x \leftarrow \text{BasicBlocks}_i(x)$
$\hat{y} \leftarrow \text{ExitBlock}(x)$
return $\hat{y}$

In Algorithm 1, EEBlock_i represents the i^{th} early-exit (EE) block of the model and BasicBlocks_i denotes the sequence of intermediate blocks between $(i-1)^{th}$ EE-block and i^{th} EE-block. Obviously, BasicBlocks_0 is the initial basic blocks of the model before entering any EE-block. N denotes the total number of early-exit blocks. h_i and $\hat{y}_i$ shows the confidence score and classification output vector of i^{th} EE-block.

3.4 Training

The trainable heads of our model are the confidence and the classification softmax branches of the early-exit blocks and the final conventional classification softmax head at the end of the model as mentioned in the previous sections. All these output

heads should be trained carefully to maintain the balance between accuracy and cost. Especially, the confidence scores are the key point of that trade-off.

Assume that the model obtains some accuracy with the current classification output vector ($\hat{\mathbf{y}}_n$) by spending the computational cost (c_n) at the n^{th} early-exit block. Our training motivation is that the confidence score of the n^{th} early-exit block (h_n) should be promoted if the prediction of the following early-exit block ($\hat{\mathbf{y}}_{n+1}$) is not worth executing the following layers with the additional computational cost between these consecutive early-exit blocks (i.e. $c_{n+1} - c_n$). In this way, the confidence scores can maintain the accuracy by taking the cost spending into account.

To catch such a proper balance between the accuracy and the computational cost, EENets are trained by separate loss parts, namely $\mathcal{L}_{MC}$ and $\mathcal{L}_{Cost}$. The $\mathcal{L}_{MC}$ denotes the multi-classification loss and the $\mathcal{L}_{Cost}$ denotes the loss coming from the computational cost. The general loss, $\mathcal{L}$, is the combination of both these losses with a trade-off parameter λ like in Equation (4).

$$\mathcal{L} = \mathcal{L}_{MC} + \lambda \mathcal{L}_{Cost} \tag{4}$$

In order to define the $\mathcal{L}_{MC}$ and $\mathcal{L}_{Cost}$, our inference approach can be employed for the training as well. As seen from Equation (5), during the training procedure, the final classification output vector, $\hat{\mathbf{y}}$, can be obtained from a single equation which consists of the outputs of all exit blocks.

$$\begin{aligned} \hat{\mathbf{y}} = & \mathrm{I}_{\{h_0 \geq T\}} * \hat{\mathbf{y}}_0 + \mathrm{I}_{\{h_0 < T\}} * \{ \\ & \mathrm{I}_{\{h_1 \geq T\}} * \hat{\mathbf{y}}_1 + \mathrm{I}_{\{h_1 < T\}} * \{ \dots \\ & \mathrm{I}_{\{h_N \geq T\}} * \hat{\mathbf{y}}_N + \mathrm{I}_{\{h_N < T\}} * \hat{\mathbf{y}}_{N+1} \} \dots \} \end{aligned} \tag{5}$$

In Equation (5), $\hat{\mathbf{y}}$ denotes the final classification output vector and N shows the number of early-exit blocks. $\hat{\mathbf{y}}_i$ and h_i denote the classification output vector and confidence score of the n^{th} early-exit block, respectively. Obviously, $\hat{\mathbf{y}}_{N+1}$ symbolizes the predicted output vector of the last exit block of the model. In addition, $\mathrm{I}_{\{h_0 \geq T\}}$ denotes the indicator function mapping a confidence score onto 0 or 1 that mean the

continue or exit decision, respectively. For example, if the first early-exit block is confident of classification that means $\mathrm{I}_{\{h_0 \geq T\}} = 1$, the predicted class label will be the classification output of the first early-exit block, namely $\hat{\mathbf{y}}_0$.

However, Equation (5) is not differentiable. As a consequence, it is not convenient for back-propagation. We modified it by approximating the indicator function with sigmoid in Equation (6) where h_n denotes the confidence score of the n^{th} early-exit block. The $\hat{\mathbf{Y}}_0$ symbolizes the classification output vector derived by all exit blocks. Similarly, $\hat{\mathbf{Y}}_{N+1}$ is the output of conventional NNs that do not consider any early-exit blocks. The derived cumulative prediction, $\hat{\mathbf{Y}}_n$, can be employed later in feeding the confidence branches of the early-exit blocks to encourage maintaining the accuracy.

$$
\begin{aligned}
\hat{\mathbf{Y}}_0 =& h_0 * \hat{\mathbf{y}}_0 + (1 - h_0) * \{ \\
& h_1 * \hat{\mathbf{y}}_1 + (1 - h_1) * \{ \dots \\
& h_N * \hat{\mathbf{y}}_N + (1 - h_N) * \hat{\mathbf{y}}_{N+1} \} \dots \} \\
\hat{\mathbf{Y}}_1 =& h_1 * \hat{\mathbf{y}}_1 + (1 - h_1) * \{ \dots \\
& h_N * \hat{\mathbf{y}}_N + (1 - h_N) * \hat{\mathbf{y}}_{N+1} \} \dots \} \\
\hat{\mathbf{Y}}_N =& h_N * \hat{\mathbf{y}}_N + (1 - h_N) * \hat{\mathbf{y}}_{N+1}
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
\hat{\mathbf{Y}}_n =& h_n * \hat{\mathbf{y}}_n + (1 - h_n) * \hat{\mathbf{Y}}_{n+1} \\
\hat{\mathbf{Y}}_{N+1} =& \hat{\mathbf{y}}_{N+1}
\end{aligned}
$$

On the other hand, the computational cost should be defined as a contrasting force in the loss function to maintain accuracy-cost balance. It should encourage the model to classify easy examples early. For this purpose, the computational cost until the n^{th} exit block, c_n, are calculated by the number of floating-point operations (FLOPs) from the beginning of the model to the last operation of that exit block. Since FLOPs values are too large to be included in the loss function (when comparing to cross-entropy loss values), they should be scaled down to some convenient number.

In order to construct a general form of this scaling down operation, the computational costs are normalized to the total number of FLOPs from the beginning to the end of the model (i.e. $c_n \in [0, 1]$). Similar to the classification label $\hat{\mathbf{Y}}_n$, the general cost,

C_n can be derived from the same approach in Equation (7).

$$
\begin{aligned}
C_0 =& h_0 * c_0 + (1 - h_0) * \{ \\
& h_1 * c_1 + (1 - h_1) * \{ \dots \\
& h_N * c_N + (1 - h_N) * c_{N+1} \} \dots \} \\
C_1 =& h_1 * c_1 + (1 - h_1) * \{ \dots \\
& h_N * c_N + (1 - h_N) * c_{N+1} \} \dots \} \\
C_N =& h_N * c_N + (1 - h_N) * c_{N+1}
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
C_n =& h_n * c_n + (1 - h_n) * C_{n+1} \\
C_{N+1} =& c_{N+1}
\end{aligned}
$$

After defining the training prediction of class label and the computational cost, the $\mathcal{L}_{MC}$ and $\mathcal{L}_{Cost}$ can be described in Equation (8) where $\mathbf{y}$ is the ground truth value and $\mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}})$ is the cross-entropy function as a conventional classification loss. In Equation (8), K denotes the number of classes in the dataset.

$$
\begin{aligned}
\mathcal{L}_{Cost} &= C \\
\mathcal{L}_{MC} &= \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}) = -\sum_{k=1}^{K} \mathbf{y}_k * \log(\hat{\mathbf{Y}})
\end{aligned}
\tag{8}
$$

Note that the C and $\hat{\mathbf{Y}}$ are not specified with exit-id, n^{th}, in Equation (8). As a combination of $\mathcal{L}_{MC}$ and $\mathcal{L}_{Cost}$, one of our proposed loss functions, $\mathcal{L}_{v1}$, is derived in Equation (9). It is the first version of general loss function where λ trades-off between the different loss terms (in experiments, we choose $\lambda = 1$).

$$
\begin{aligned}
\mathcal{L}_{v1} &= \mathcal{L}_{MC} + \lambda \mathcal{L}_{Cost} \\
&= \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_0) + \lambda C_0
\end{aligned}
\tag{9}
$$

The zero degree of $\hat{\mathbf{Y}}$ and C are used in the $\mathcal{L}_{v1}$ since $\hat{\mathbf{Y}}_0$ and C_0 are the cumulative form of the prediction and computational cost of all exit blocks, respectively. For

example, the loss function of the model which has three early-exit blocks are given in Equation (10).

$$
\begin{aligned}
\mathcal{L}_{v1} =& \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_0) + \lambda C_0 \\
\mathcal{L}_{v1} =& \mathrm{CE}(\mathbf{y}, h_0 * \hat{\mathbf{y}}_0 + (1-h_0) * (h_1 * \hat{\mathbf{y}}_1 + (1-h_1) * \hat{\mathbf{y}}_2)) \\
&+ \lambda * (h_0 * c_0 + (1-h_0) * (h_1 * c_1 + (1-h_1) * c_2))
\end{aligned}
\tag{10}
$$

As seen from Equation (9), the multi-classification cross entropy employs the cumulative prediction, $\hat{\mathbf{Y}}_0$. However, if the confidence score of an early-exit block is high (i.e. $h_n \simeq 1$), the predictions of the following exit blocks (i.e. $\hat{\mathbf{y}}_{n+1}$, $\hat{\mathbf{y}}_{n+2}$, $\ldots$, $\hat{\mathbf{y}}_{N+1}$) may not feed the model since they do not contribute the prediction $\hat{\mathbf{Y}}_0$. As a consequence, the following exit blocks are not trained properly without enough back-propagation signals. We need to be sure that the predictions coming from all exit blocks are trained fairly.

By this motivation, we propose the second version of general loss functions, namely $\mathcal{L}_{v2}$ that is the weighted summation of the losses promoting the all exit blocks. $\mathcal{L}_{v2}$ can be defined in Equation (11).

$$
\begin{aligned}
\mathcal{L}_{MC}^{(n)} &= \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_n) \\
\mathcal{L}_{Cost}^{(n)} &= C_n \\
\mathcal{L}_{v2} &= \sum_{n=0}^{N+1} (\mathcal{L}_{MC}^{(n)} + \lambda \mathcal{L}_{Cost}^{(n)}) \\
&= \sum_{n=0}^{N+1} (\mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_n) + \lambda C_n)
\end{aligned}
\tag{11}
$$

The second version of loss function of the model which has three early-exit blocks

are given in Equation (12) as an example.

$$\mathcal{L}_{v2} = \mathcal{L}^0_{MC} + \lambda\mathcal{L}^0_{Cost} + \mathcal{L}^1_{MC} + \lambda\mathcal{L}^1_{Cost} + \mathcal{L}^2_{MC} + \lambda\mathcal{L}^2_{Cost}$$

$$\begin{aligned}\mathcal{L}_{v2} &= \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_0) + \lambda C_0 \\ &\quad + \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_1) + \lambda C_1 \\ &\quad + \mathrm{CE}(\mathbf{y}, \hat{\mathbf{Y}}_2) + \lambda C_2\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{v2} &= \mathrm{CE}(\mathbf{y}, h_0 * \hat{\mathbf{y}}_0 + (1-h_0) * (h_1 * \hat{\mathbf{y}}_1 + (1-h_1) * \hat{\mathbf{y}}_2)) \\ &\quad + \lambda(h_0 * c_0 + (1-h_0) * (h_1 * c_1 + (1-h_1) * c_2)) \\ &\quad + \mathrm{CE}(\mathbf{y}, h_1 * \hat{\mathbf{y}}_1 + (1-h_1) * \hat{\mathbf{y}}_2) \\ &\quad + \lambda(h_1 * c_1 + (1-h_1) * c_2) \\ &\quad + \mathrm{CE}(\mathbf{y}, \hat{\mathbf{y}}_2) \\ &\quad + \lambda c_2\end{aligned} \tag{12}$$

In this version of general loss, each early-exit block has a change to contribute the prediction $\hat{\mathbf{Y}}_n$ (relatively the cross-entropy loss) since the equation of the prediction starts with its confidence score h_n that cannot be dominated by previous confidence scores. Even if some of the exit blocks are dominant with high confidence scores, the others can promote the loss function and can be properly back-propagated as well.

Overall, all exit blocks contribute the loss function for all examples, even if easy examples can be classified at the shallower early-exit blocks. In other words, multiple outputs coming from all exit blocks are trained jointly with the general custom loss. Therefore, EENets do not have any dead layer problem happened in previous works [1], [2], [3]. As a result, EENets do not require a complicated multi-stage training process as well.

Finally, note that the cost loss $\mathcal{L}_{Cost}$ is an important factor to avoid entire execution of network for all examples. As seen from Equation (6), because the most possible correct predictions come from the last exit block that exploits all capacity of the model,

it is expected that the confidence scores tend to be higher value at the deeper exit points, gradually. However, the cost penalty $\mathcal{L}_{Cost}$ forces the model to terminate early since the shallower early-exit blocks have less computational cost. Therefore, the confidence scores of the shallower early-exit blocks are promoted. In the big picture, the confidence scores actually learn if the predictions of the following exit blocks are worth executing more layers with the additional cost coming from these layers.

3.5 Distributing Early-exit Blocks to a Network

The number of early-exit (EE) blocks and their distribution technique is another important factor in the architecture of the model. The number of EE-blocks depends on the depth of the network. Because the additional parameters coming from EE-blocks are so small and negligible (i.e. $\sim$0.0002% of the total parameters of EENet-110 for each EE-block with 10-classes datasets), the early-exit blocks can be added as much as desired if the model holds.

The EE-blocks can be distributed based on the dataset and the capacity of the network. EENets are suitable for many distribution methods such as; *Pareto, Golden Ratio, Fine, Linear, Quadratic*, etc. According to the Pareto principle, 80% of the results have been done by 20% of works. The *Pareto* distribution is inspired by that principle (i.e. 80% of examples may be classified just by spending 20% of the total computational cost of the model). By this motivation, the first EE-block splits the network according to the Pareto principle where 20% of the total computational cost is calculated in terms of the number of floating-point operations (FLOPs). Similarly, the *Fine* distribution method divides the network and places the EE-blocks based on 5% of the total FLOPs. On the other hand, the *Golden ratio* distribution employs the golden ratio, 0.6180, to apportion computational cost and to place the EE-blocks.

The *Linear* and *Quadratic* distributions split the network where the computational cost of the layers between two consecutive EE-blocks increases in linear or quadratic form, respectively. Figure 3.2 shows some of the distribution methods. Note that there is not a best distribution method for all EENets. Empirically, the effects of the distribution methods can be examined based on the dataset and the model capacity.

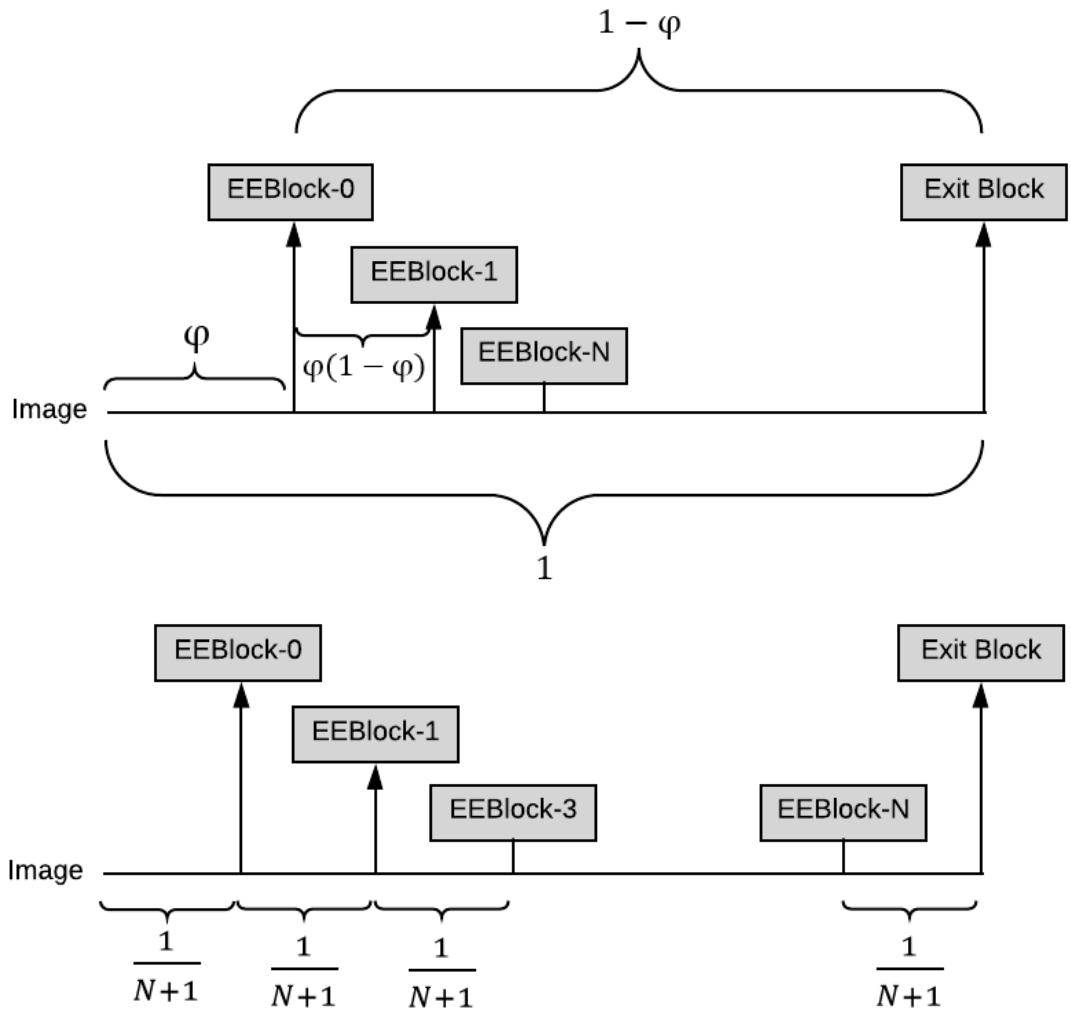

Figure 3.2: Distributing early-exit blocks to a network. *Pareto*, *Golden Ratio* and *Fine* can be represented in the **upper** figure. The φ denotes the ratio used in the methods. For example, φ will be 0.2, 0.6180 and 0.05 for *Pareto*, *Golden Ratio* and *Fine* distributions, respectively. N shows the number of early-exit blocks. The **below** figure shows the *Linear* distribution where the computational costs between consecutive early-exit blocks are same and this cost can be calculated by the desired number of the early-exit blocks. Notice that the total cost is represented by 1 since our cost terms are rates (i.e. $c \in [0, 1]$).

CHAPTER 4

EXPERIMENTS

4.1 Introduction

In this chapter, we describe our experimental validation of EENets. The chapter provides and demonstrates the test environment, the overview of experimented architectures, and the results of benchmark with ResNets.

Although our early-exit blocks can be applied to any CNN architecture, we have chosen ResNets [13] for their widespread use. In the experiments, early-exit (EE) ResNets are compared with their non-EE versions on MNIST [14], CIFAR10 [16] and SVHN [15] datasets. In addition to ResNets, we also experiment with a very low-capacity, custom CNN on the MNIST dataset.

The experiments are diversified in order to observe the effects of EENets in different aspects and certain conditions. In this section, we try to answer the following questions through comprehensive experiments:

- Do EENets really work? That is, is the inference processes terminated for individual examples at different exit locations? Is there a variety in the exit locations chosen by the network?
- Are EENets successful when compared to their counterparts in terms of the computational cost and the accuracy?
- Which type of early-exit block should be chosen in certain conditions?
- How does the distribution of early-exit blocks affect the accuracy and the computational cost?

Our experiment environment included a i7-6700HQ CPU processor with 16GB RAM and 2x NVIDIA Tesla PICe P100 16GB. The models are implemented in Keras with the Tensorflow backend but we also implemented a toy model in PyTorch. Both source codes of Keras and PyTorch implementations are published in GitHub[1]. The models were optimized by Adam with the *learning rate* $= 0,001$ and we used early stopping with *min delta* $= 0,001$ and *patience* $= 20$. The mini-batch size in the experiments was 32. The most of the models were trained up to 200 epochs unless otherwise stated.

4.2 Experimented Architectures

In the experiments, EENets built by adding the early-exit blocks to the ResNet models [13] (both basic and bottleneck architectures) are evaluated. The early-exit (EE) ResNets based on bottleneck architectures consist of 50, 101 and 152 layers. By modifying the 6n+2 layers ResNet models [13], we have constructed 20, 32, 44 and 110 layers EENets. Various number of early-exit blocks are distributed according to the capacity of the models. The models which have a large capacity are trained on CIFAR10 [16] and SVHN [15].

On the other hand, the models having a smaller capacity are evaluated on MNIST [14] to observe how EENets perform in the situation of a dataset forcing the capacity of the model. These small capacity networks are composed of 6, 8 and 18 layers with a small number of filters.

Some of the ResNet based architectures that are evaluated in our experiments are shown in Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6. Our own design EENet-8 which has identity connections like ResNet is shown in Figure 4.7. This is a very small CNN having 2-8 filters in its layers. We ran this low capacity model on the MNIST dataset.

[1] Keras implementation: https://github.com/eksuas/EENet and
PyTorch implementation: https://github.com/eksuas/EENets_pytorch

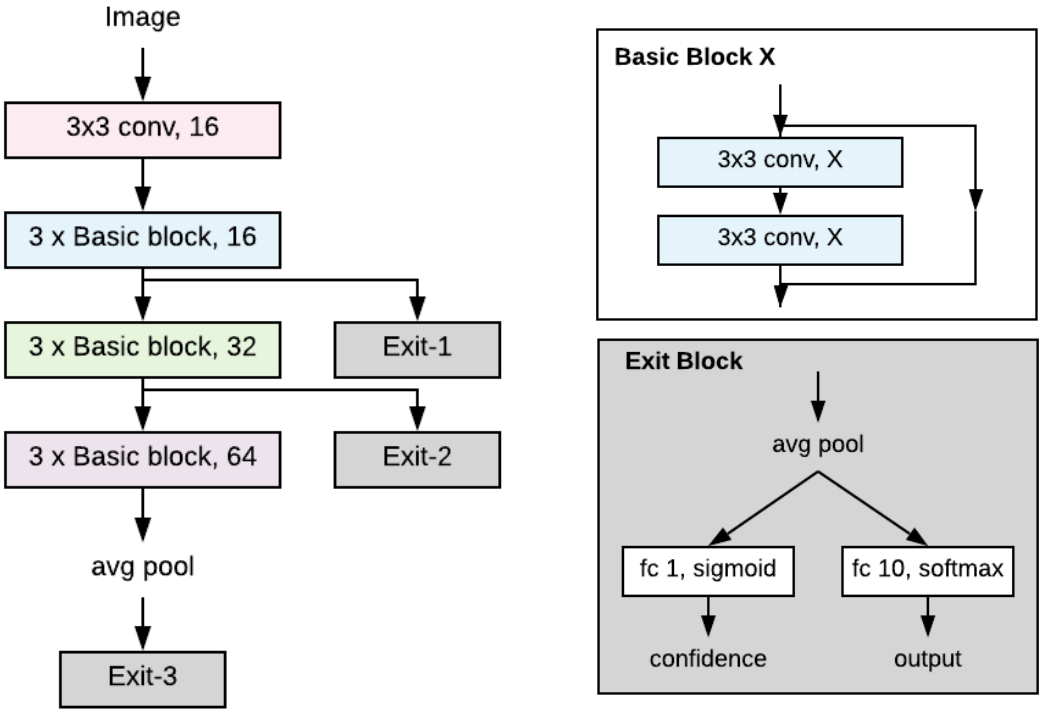

Figure 4.1: EENet-20 model with two early-exit blocks. The *pool*-type of early-exit blocks and the basic blocks of ResNets are used. The X denotes the number of filters in the basic blocks. The number of floating-point operations (FLOPs) of the exit blocks are $74K$, $334K$ and $1368K$, respectively. Thus, the costs of the exit blocks are 0.05, 0.24 and 1.00. The architecture of the model is the form of 6n+2 ResNet models.

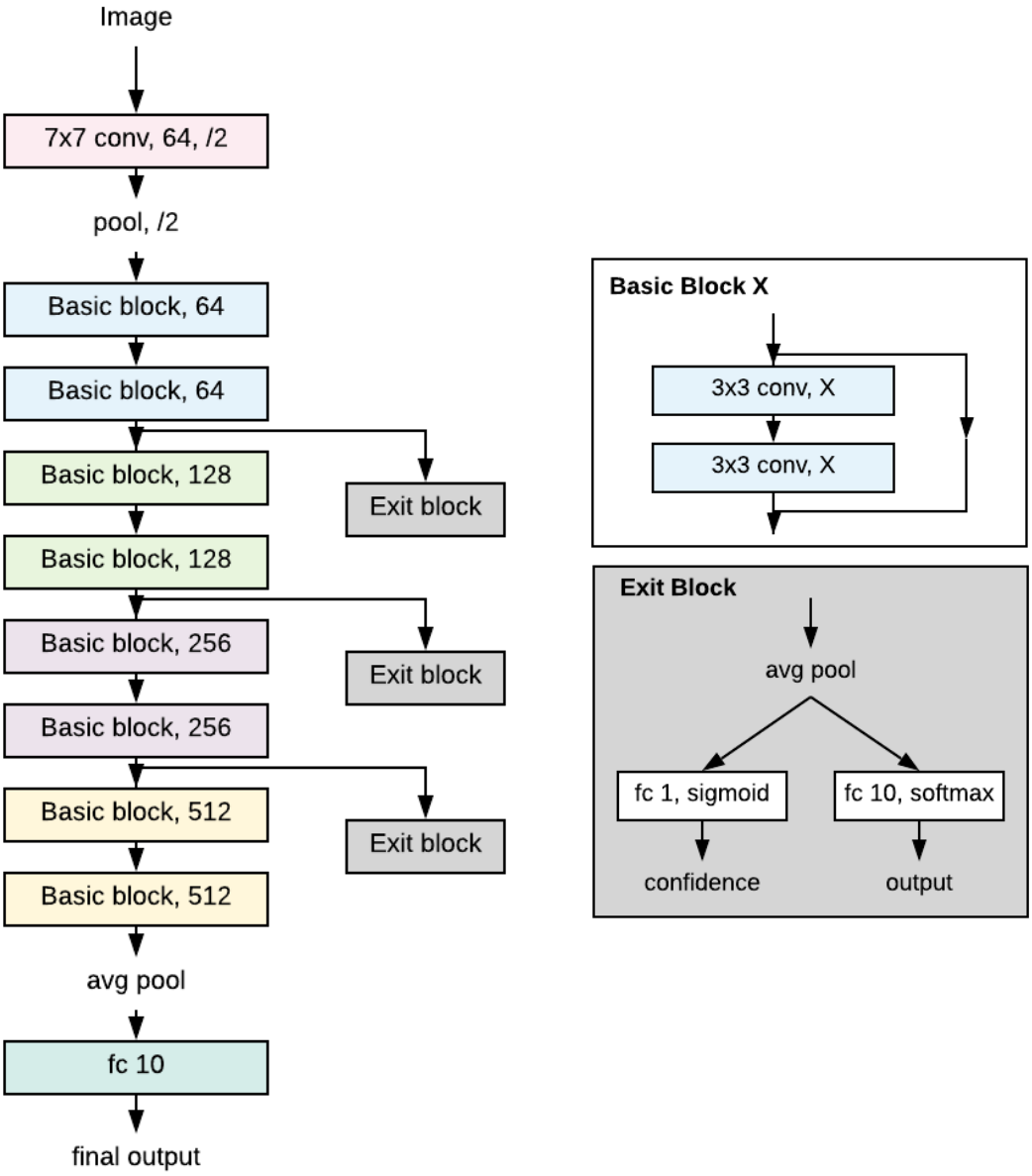

Figure 4.2: EENet-18 model with three early-exit blocks whose type is the *pool*-type. The basic blocks of ResNets are used as well. The X denotes the number of filters in the basic blocks. The FLOPs of the exit blocks are $0.79M$, $3.42M$, $13.93M$ and $55.92M$, respectively. Thus, the costs of the exit blocks are 0.01, 0.06, 0.25 and 1.00. The architecture of the model is the form of Naive ResNet models.

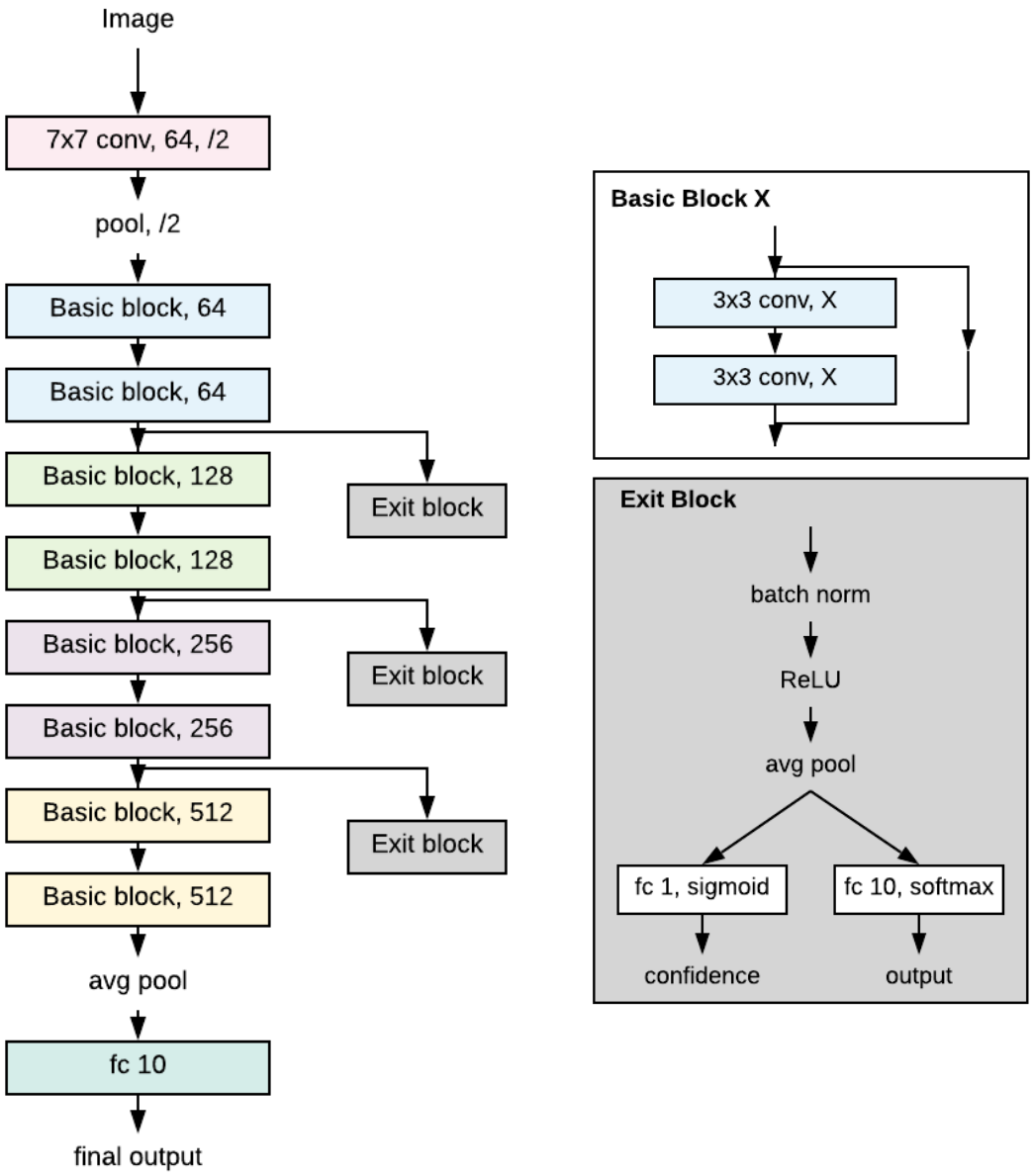

Figure 4.3: EENet-18 model with three early-exit blocks. The type of the early-exit blocks is the *bnpool*. The basic blocks of ResNets are used as well. The X denotes the number of filters in the basic blocks. The FLOPs of the exit blocks are $0.79M$, $3.42M$, $13.94M$ and $55.94M$, respectively. As a consequence, the costs of the exit blocks are 0.01, 0.06, 0.25 and 1.00. It is another example of the architectures in the Naive ResNet form.

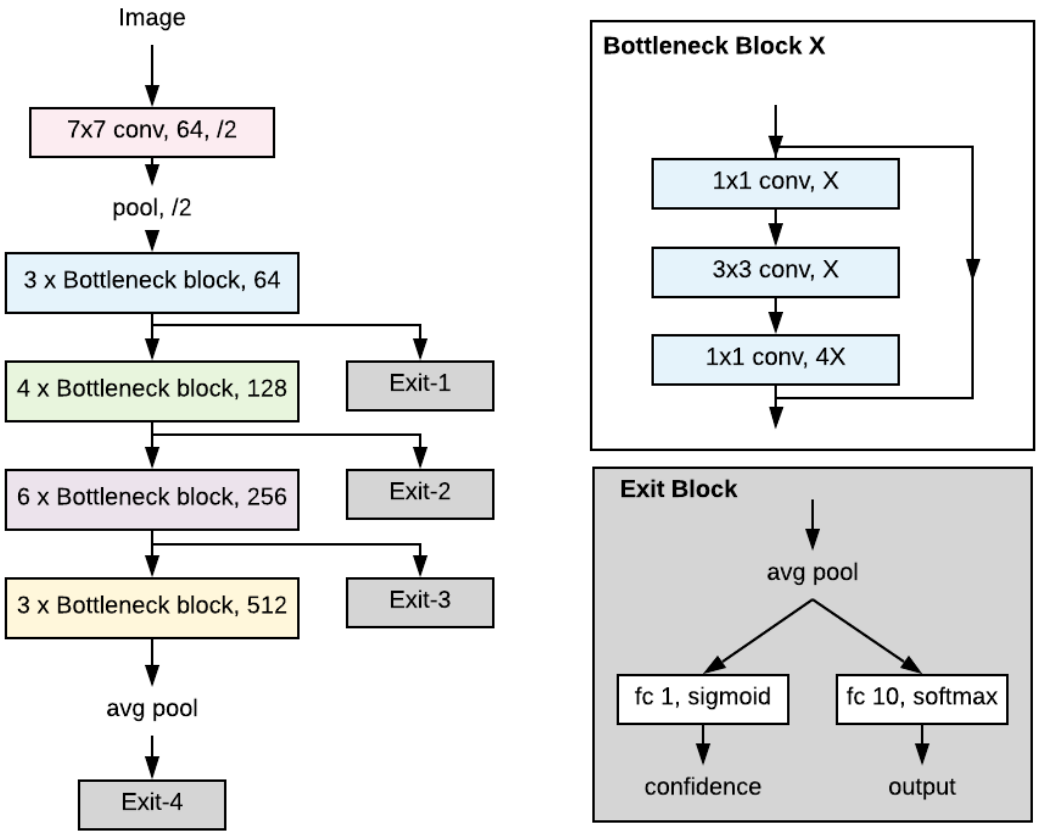

Figure 4.4: EENet-50 model with three early-exit blocks. The type of the early-exit blocks is the *pool*. The bottleneck blocks of ResNets are used in this architecture. The X denotes the number of filters in the bottleneck blocks. The FLOPs of the exit blocks are $1.14M$, $7.27M$, 42.86 and $117.83M$, respectively. As a result, the costs of the exit blocks are 0.01, 0.03, 0.36 and 1.00. The architecture of the model is the form of Naive ResNet models.

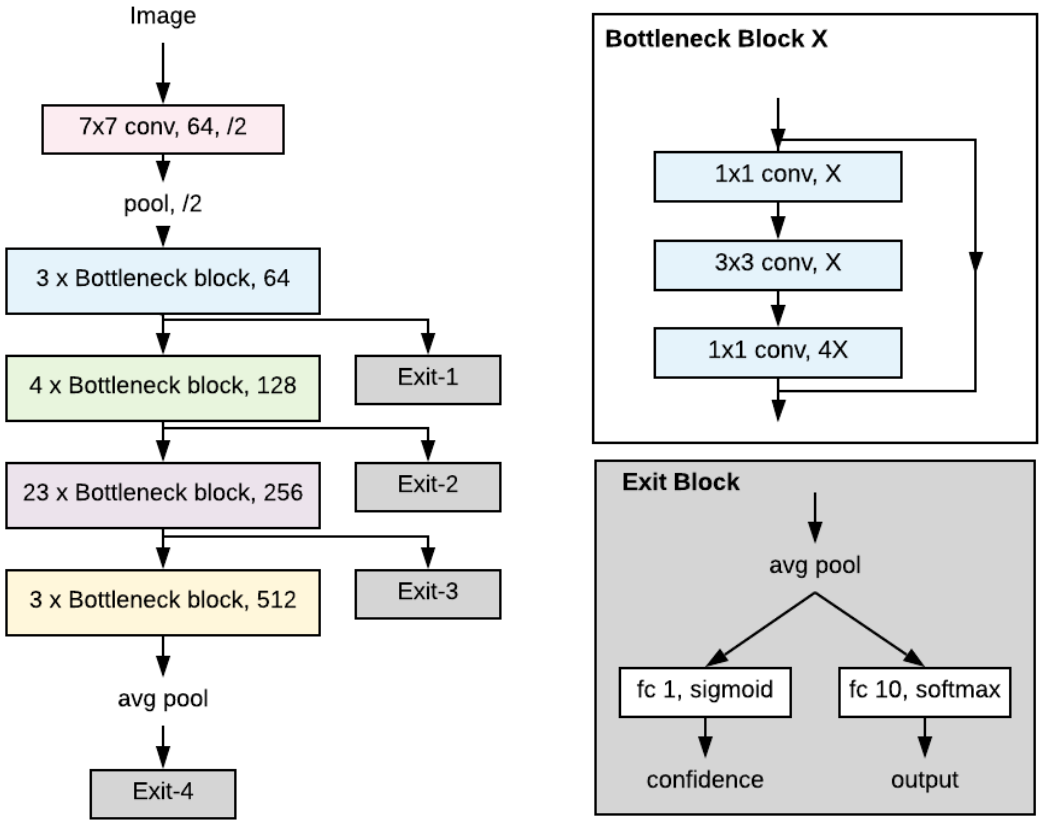

Figure 4.5: EENet-101 model with three early-exit blocks. The *pool*-type of early-exit blocks and the bottleneck blocks are employed. The FLOPs of the exit blocks are $1.14M$, $7.27M$, 138.05 and $213.02M$, respectively. Thus, the costs of the exit blocks are 0.01, 0.06, 0.65 and 1.00. The architecture of the model is the form of Naive ResNet models.

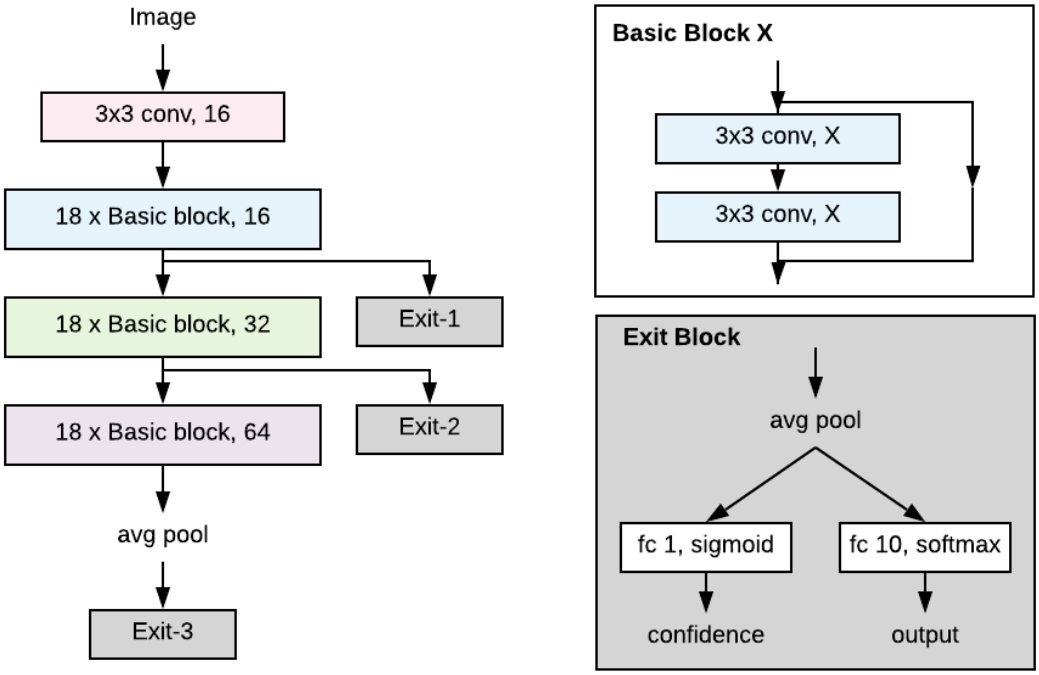

Figure 4.6: EENet-110 model with two early-exit blocks. The early-exit blocks are in the *pool*-type. The FLOPs of the exit blocks are $0.42M$, $2.09M0$ and $8.69M$, respectively. Consequently, the costs of the exit blocks are 0.05, 0.240 and 1.00. It is another example of the architectures in the 6n+2 ResNet form.

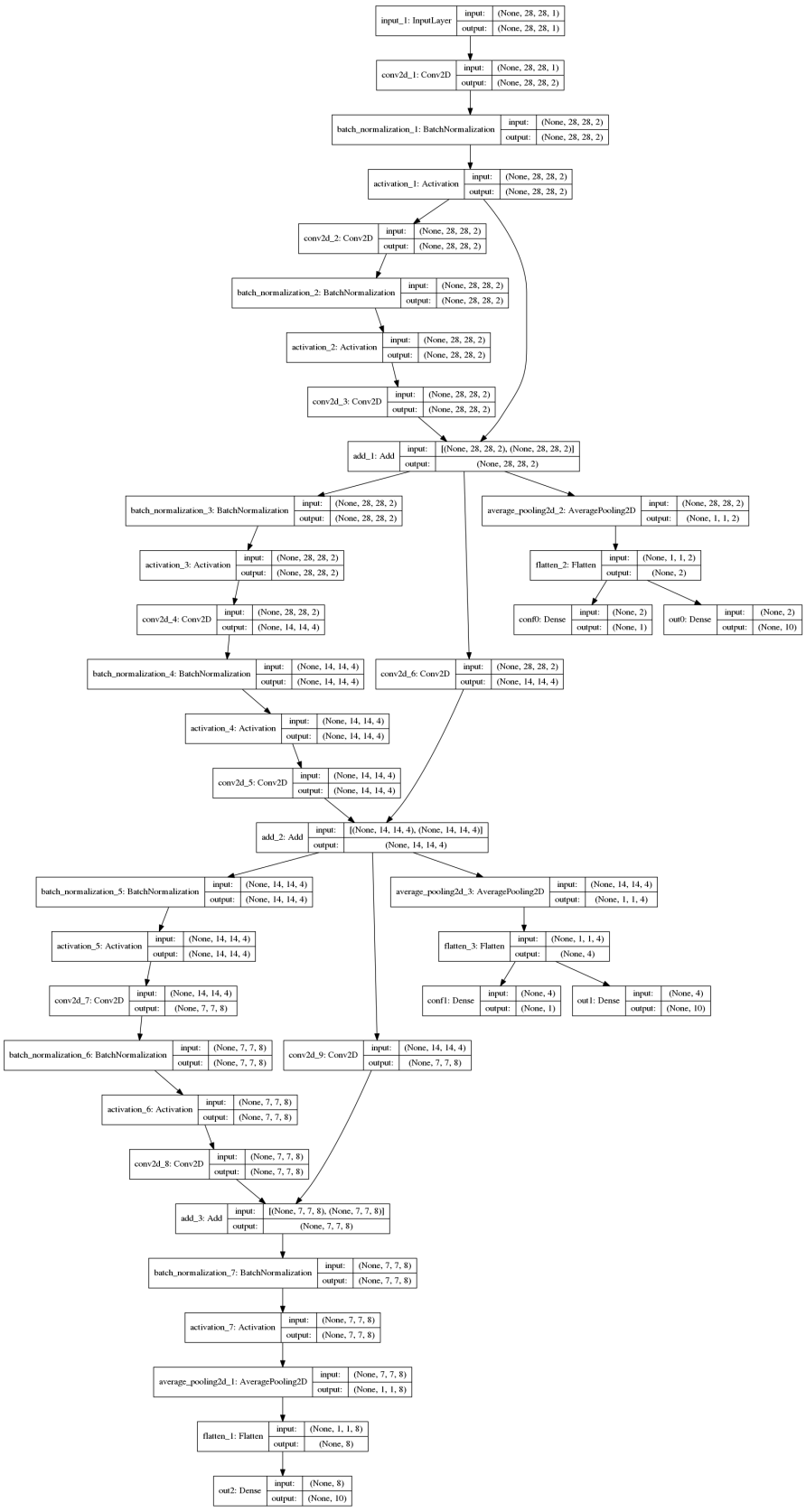

Figure 4.7: EENet-8 model with two early-exit blocks.

4.3 Metrics

The testing accuracy of EENets is measured by considering the predicted labels at exit locations. To measure computational cost, we use the number of floating-point operations (FLOPs).

4.4 Results on MNIST

Firstly, we perform a set of experiments on MNIST dataset [14] to validate whether the confidence scores of the early-exit blocks are meaningful (i.e. related with the accuracy of the predictions of these early-exit blocks) and have a variety in inputs. In these basic tests, the EENet-8 model is employed with quadraticly distributed two *pool*-type early-exit blocks whose number of floating-point operations (FLOPs) and costs are given in Table 4.1. The exit distribution of the MNIST examples on the EENet-8 models trained with different loss functions are given in Table 4.1 as well. The λ trade-off of our loss functions is chosen as 1.0 in these experiments.

As expected, the model EENet-8-$\mathcal{L}_{Cost}$ terminates the executions at the first early-exit block by considering only the computational cost while EENet-8-$\mathcal{L}_{MC}$ classifies all examples at the last exit block to get the highest accuracy. On the other hand, the EENet-8 model trained with $\mathcal{L}_{v2}$ takes both the cost and accuracy into account, as a consequence, it maintains the accuracy by spending less computational cost. Moreover, our models classify inputs and terminate executions in a proportional distribution. For example, in Table 4.1, 969, 2251 and 6780 numbers of test examples of MNIST are classified at the early-exit (EE) block-0, EE-block-1 and the last exit layer of the EENet-8 model, respectively. This experiment shows that our loss function performs as expected and maintains the balance between the accuracy and computational cost by meaningful confidence scores. Because of that, the EENet models trained with the $\mathcal{L}_{v2}$ loss function are evaluated in the remaining experiments.

Note that the success of our model (in Table 4.1) in terms of the computational cost is not as good as the results of experiments performed on other models and datasets (the experiments will be given in the following sections). The reason behind that, the

Table 4.1: Exit distribution of the MNIST examples with different loss functions. This table shows the results of MNIST examples evaluated on the EENet-8 model (Figure 4.7) with 20 epochs in PyTorch. In the **upper** table, the computational cost rates and the number of FLOPs from the beginning to the early-exit blocks are given in the ***Relative Cost*** and ***FLOP*** columns, respectively. ***# examples that exit*** shows the number of examples that are classified at that exit block. The exit distribution of the MNIST test examples (10000 examples) on the EENet-8 models are shown in the **below** table where EENet-8-$\mathcal{L}_{v1}$, EENet-8-$\mathcal{L}_{v2}$, EENet-8-$\mathcal{L}_{Cost}$ and EENet-8-$\mathcal{L}_{MC}$ are the EENet-8 models trained with only the $\mathcal{L}_{v1}$, $\mathcal{L}_{v2}$, $\mathcal{L}_{Cost}$ and $\mathcal{L}_{MC}$ loss functions, respectively. ***Last Exit*** represents the last exit block. Testing accuracy is given in the ***Accuracy*** column. Note that the cost of ResNet-8 is always 1 since it computes the whole model. Since early-exit blocks are not available for ResNets, "-" is placed in the early-exit columns.

A. Exit distribution on EENet-8-$\mathcal{L}_{v2}$

Exit Blocks	**FLOP**	**Relative Cost**	**# examples that exit**
EE-block0	546	0.08	969
EE-block1	1844	0.26	2251
Last Exit	6982	1.00	6780

B. Benchmark of different loss functions

Model	**# examples that exit from**			**Accuracy**	**Relative Cost**
	EE-block0	**EE-block1**	**Last Exit**		
ResNet-8	-	-	10000	97.38	1.00
EENet-8-$\mathcal{L}_{MC}$	0	0	10000	97.42	1.00
EENet-8-$\mathcal{L}_{Cost}$	10000	0	0	10.32	0.08
EENet-8-$\mathcal{L}_{v1}$	6614	3386	0	54.05	0.14
EENet-8-$\mathcal{L}_{v2}$	47	2247	7706	96.55	0.82

capacity of the model EENet-8 (in Figure 4.7) is very low than the traditional ResNets in the original paper [13]. As seen from Table 4.1, the FLOPs of EENet-8 is 6982 while a traditional ResNet110 has 8.6M FLOPs. We deliberately choose this model to force it to early classify the examples of MNIST as an easy dataset. Otherwise, with a large capacity model, MNIST examples are classified in the early stages of the model (mostly at the first early-exit block) because the dataset consists of simple examples. As a consequence, the proportional distribution may not be examined on MNIST if the model capacity is large like a traditional ResNet.

The computational cost, accuracy and loss values per epoch are shown in Figure 4.9. We evaluate the model with different optimizers and learning rates. Adam optimizer with learning rate 0.001 gives the best results. As seen in the figure, the accuracy increases while the computational cost decreases gradually. As a consequence, we continue training the models with Adam in the remaining experiments.

Another set of experiments are performed on MNIST to observe the effects of λ trade-off on the loss function $\mathcal{L}_{v2}$. The results can be seen in Table 4.2 and Figure 4.8. The best balance between the accuracy and the computational cost is observed in condition $\lambda \approx 0.95$. However, the effects of $\mathcal{L}_{MC}$ or $\mathcal{L}_{Cost}$ can be changed through the λ trade-off if more accurate results or less computational cost consumption are desired (e.g. λ can be decreased to obtain more accurate results if the computational cost is not an issue).

Random MNIST examples classified with EENet-8 which consists of two early-exit (EE) blocks are shown in Figures 4.10, 4.11 and 4.12 as classified at the EE-block-0, EE-block-1 and the last exit blocks of the model, respectively . It is observed that the early-exit blocks are specialized in visually similar examples of the same class or in a few visually similar classes. For example, the EE-block-0 is only specialized in the class of the number eight in this model (i.e. visually similar examples of the same class). On the other hand, the EE-block-1 classifies the class of the number one, four and seven. Note that these classes are visually similar as well.

Table 4.2: Effects of $\boldsymbol{\lambda}$ trade-off on the loss functions $\boldsymbol{\mathcal{L}_{v2}}$. In the experiment, EENet-8 model starting with 4 filters is evaluated on MNIST. The model is trained with different $\boldsymbol{\lambda}$ trade-off by using ADAM optimizer on 20 epochs. The exit distribution of the MNIST test examples (10000 examples) is shown in the table where the values are the results of the last epoch. Testing accuracy is given in the ***Accuracy*** column. ***Time*** column shows the average wall clock time of inference procedure in microseconds (µs). The computational cost rates are given in the ***Relative Cost*** column. ***# examples that exit from*** shows the number of examples that are classified at that exit block.

$\boldsymbol{\lambda}$	**Accuracy**	**Time (µs)**	**Relative Cost**	**# examples that exit from**		
				EE-block0	**EE-block1**	**Last Exit**
0.50	98.84	638.0	1.00	0	0	10000
0.70	98.52	711.6	1.00	0	0	10000
0.90	97.46	681.4	0.78	1120	1498	7382
0.95	97.48	643.3	0.74	1230	1883	6887
1.00	96.22	615.0	0.82	359	1879	7762
1.05	97.53	593.6	0.85	168	1771	8061
1.10	98.34	641.9	0.85	3	1939	8058
1.15	85.93	557.8	0.45	48	7245	2707
1.30	86.96	487.3	0.26	0	9996	4
1.50	85.19	476.4	0.26	0	9997	3

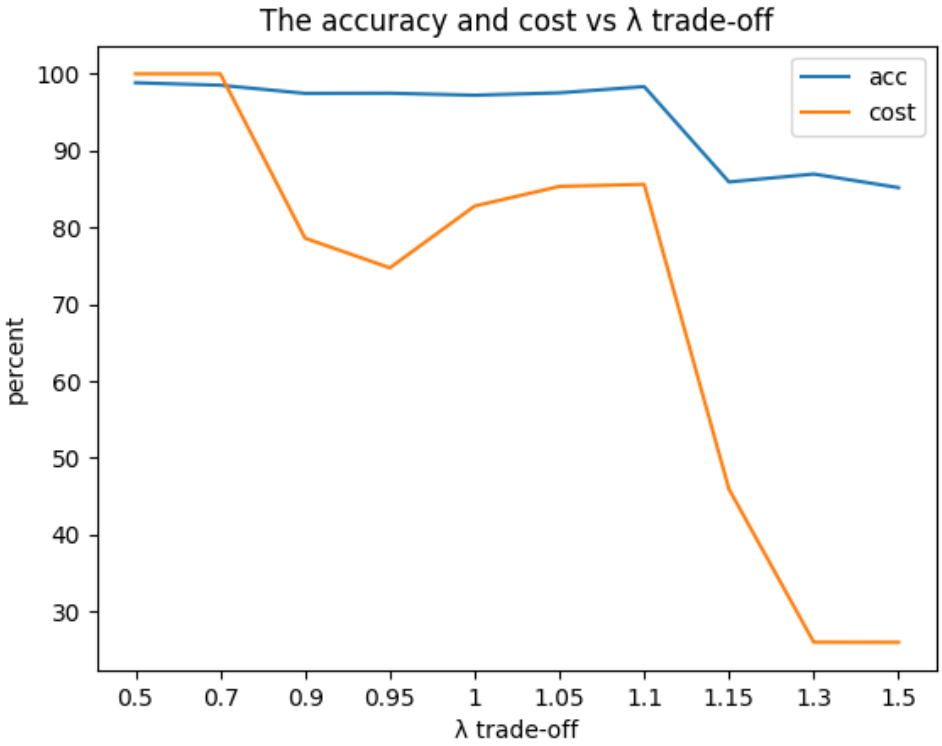

Figure 4.8: The accuracy and computational cost vs λ trade-off on EENet-8.

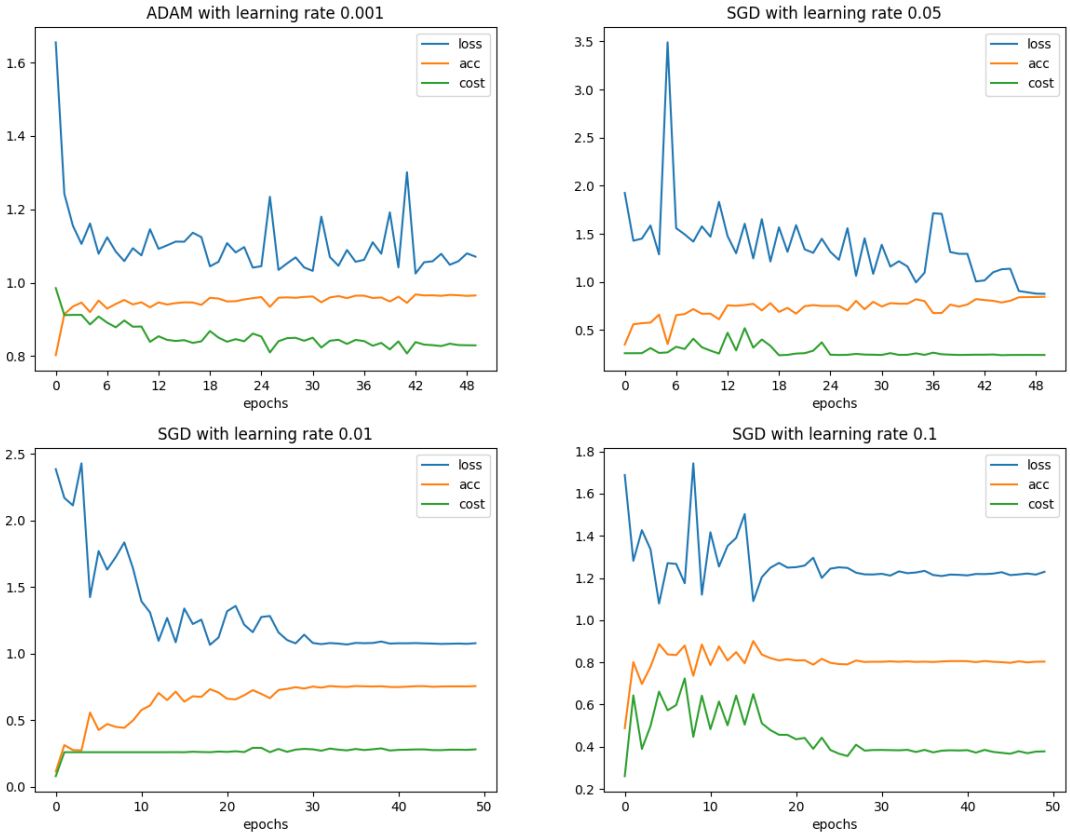

Figure 4.9: Epochs vs accuracy, loss and computational cost of the EENet-8 model starting with 2 filters on MNIST.

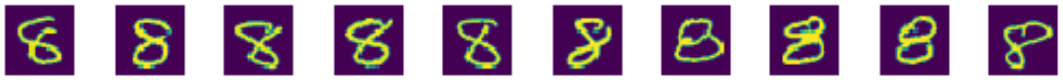

Figure 4.10: The MNIST examples are classified at the first early-exit block.

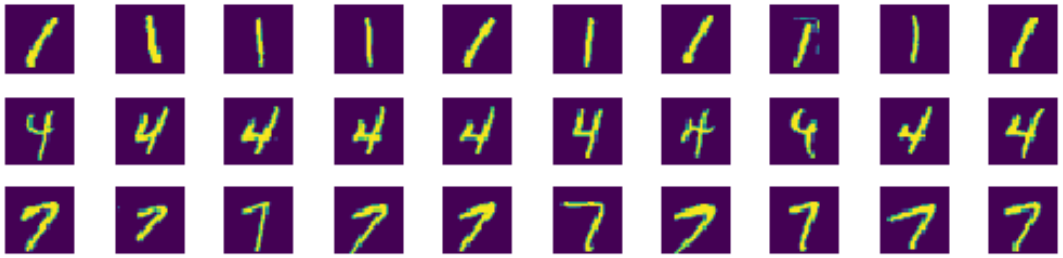

Figure 4.11: The MNIST examples are classified at the second early-exit block.

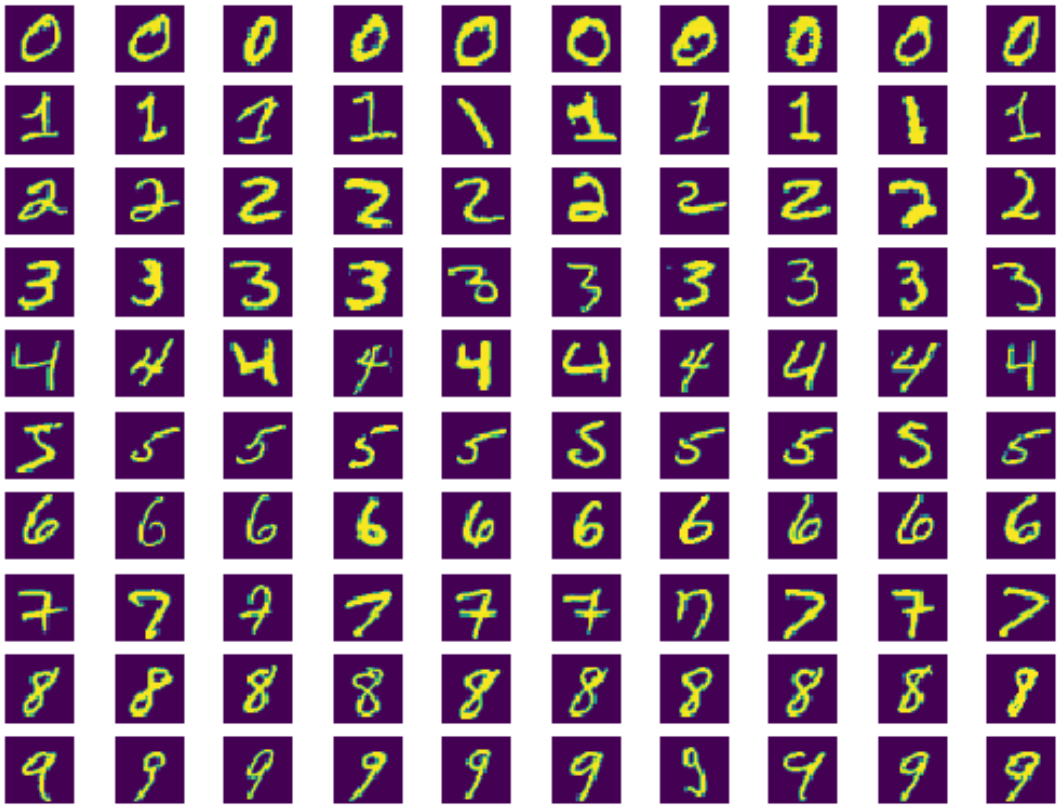

Figure 4.12: The MNIST examples are classified at the last exit block of the model.

Table 4.3: Types of the early-exit blocks. The type of early-exit blocks: *Plain*, *Pool* and *Bnpool* are evaluated on EENet-50 model with SVHN dataset. *Pool*-type is the more accurate one with a small difference.

Block Type	**Accuracy**	**Relative Cost**
Plain	90.05	0.01
Pool	94.59	0.06
Bnpool	94.45	0.06

Finally, we have examined the models on MNIST in terms of the distributions of the early-exit blocks. It is not hard to classify MNIST dataset on ResNet based models since these models enjoy a large capacity than MNIST required. Thus, keeping the first early-exit blocks in the very beginning of the model decreases the cost excessively. Because of that, the quadratic distribution with a small number of EE-blocks can be a good choice for this type of models and datasets (i.e. a large-capacity model with a simple dataset that can be easily classified).

4.5 Results on SVHN

We evaluate our proposed model with the original ResNets on SVHN dataset [15] as well.

We have firstly examined the performance of types of the early-exit blocks on EENet-50 model in Figure 4.4. The results are given in Table 4.3. It is seen that the *Pool*-type early-exit block produces the most accurate results among the exit blocks, even if there is a small difference with the result of *Bnpool*-type. Consequently, we employ the *Pool*-type early-exit blocks in the rest of experiments.

In order to show the actual performance of the early-exit idea, we have evaluated large capacity network through comprehensive experiments on SVHN. The early-exit (EE) ResNets which have the 6n+2 architecture [13] achieve similar accuracy with their non-EE versions while reducing the computational cost upto 11% of the original

Table 4.4: Results of 6n+2 based ResNets on SVHN. The average number of floating-point operations are given in the column of ***FLOP***. ***#Params*** denotes the total number of model parameters. The given accuracy in the table is the testing accuracy.

Model	**Accuracy**	**#Params**	**FLOP**
ResNet-20	95.61	0.27M	1.37M
ResNet-32	95.72	0.47M	2.34M
ResNet-44	95.79	0.67M	3.32M
ResNet-110	95.68	1.74M	8.69M

(e.g. the cost of the EENet-110 model whose EE-blocks are quadraticly distributed). In average, EENets degrade the computational cost to 20% of that of their non-EE versions. The results of these experiments are given in Table 4.4 and Table 4.5. As seen, the *quadratic* and *fine* distributed models spend less computational cost than the other distributed one. Besides, these models maintain the accuracy.

On the other hand, *linear* distributed EENets provide a little bit more accuracy. However, they spend 2-fold more computational cost than the models whose EE-block distributed by the other methods do. The main reason behind that the first early-exit block of the *linear* distributed model is located in much deeper layers than the first EE-block of the model of other distributions. For example, the first EE-block of the *linear* distributed EENet-32 spends 20% of the total computational cost while quadraticly distributed one spends only 3% of that. The other explicit observation is that the computational cost decreases while the model capacity increases in the quadraticly distributed models.

Early-exit versions of the Naive ResNet models have a huge success comparing with their non-EE counterparts [13] in terms of the computational cost. As seen from Table 4.6, EENet-18 classifies the inputs more accurately than ResNet-18 just by spending 6% of the total computational cost of the model. Moreover, EENet-152 reduces the computational cost to 2% of that of its non-EE version. Note that the Naive ResNet models have a larger capacity than the ResNets based on 6n+2 architecture. These naive models were designed and evaluated on ImageNet dataset [22] in the original

Table 4.5: Results of the 6n+2 based EENets on SVHN. The computational cost rates and the average number of floating-point operations per example are given in the columns of ***Cost*** and ***FLOP***, respectively. The distribution methods of models are given in the first column. ***#E*** denotes the number of EE-blocks. ***Cost Percent of EE-Blocks*** shows the distribution of cost percent of EE-blocks. The given accuracy in the table is the testing accuracy.

	Model	**Acc.**	**#E**	**Cost Percent of EE-Blocks**	**Cost**	**FLOP**
Quadratic	EENet20	94.94	3	<4,11,24>	0.24	0.33M
	EENet32	95.08	4	<3,8,20,52>	0.20	0.47M
	EENet44	95.37	6	<2,4,10,19,33,66>	0.17	0.56M
	EENet110	95.41	6	<2,4,11,20,40,74>	0.11	0.97M
Fine	EENet20	94.99	3	<5,11,18>	0.17	0.23M
	EENet32	95.21	5	<5,12,16,20,24>	0.23	0.54M
	EENet44	95.59	6	<5,10,16,21,24,33>	0.21	0.70M
	EENet110	95.65	10	<6,10,14,19,23,27,32,36,40,44>	0.21	1.82M
Pareto	EENet20	95.03	3	<24,45,73>	0.45	0.62M
	EENet32	95.47	5	<20,36,52,68,84>	0.46	1.07M
	EENet44	95.49	6	<21,44,55,66,78,89>	0.40	1.33M
	EENet110	95.74	6	<21,40,53,62,70,74>	0.50	4.33M
GoldenRate	EENet20	94.79	3	<24,45,73>	0.39	0.54M
	EENet32	95.26	5	<12,16,24,52,68>	0.23	0.55M
	EENet44	95.73	6	<7,10,16,24,44,66>	0.39	1.29M
	EENet110	95.51	6	<6,10,15,24,40,66>	0.34	3.03M
Linear	EENet20	95.31	2	<45,73>	0.66	0.90M
	EENet32	95.62	5	<20,36,52,68,84>	0.47	1.10M
	EENet44	95.72	6	<16,33,44,66,78,89>	0.41	1.36M
	EENet110	95.56	10	<10,19,27,40,49,57,66,74,83,91	0.19	1.62M

Table 4.6: Results of the Naive ResNet based models on SVHN. The results of the quadraticly distributed Naive ResNet based models with three EE-blocks are given below. The computational cost rates and the average number of floating-point operations per example are given in the columns of ***Relative Cost*** and ***FLOP***, respectively. ***#Params*** denotes the total number of model parameters. The given accuracy in the table is the testing accuracy.

Model	**Accuracy**	**#Params**	**Relative Cost**	**FLOP**
ResNet-18	93.77	11M	1.00	1.37M
ResNet-50	95.72	23M	1.00	2.34M
ResNet-101	95.79	42M	1.00	3.32M
ResNet-152	95.68	58M	1.00	8.69M
EENet-18	95.61	11M	0.06	3.42M
EENet-50	95.72	23M	0.06	7.27M
EENet-101	95.79	42M	0.03	7.27M
EENet-152	94.68	58M	0.02	6.09M

paper [13]. Therefore, SVHN dataset [15] may be too easy to be classified by the Naive ResNet models. The experiments show that EENets execute enough layers to classify SVHN dataset with the same accuracy and they do not waste the computation in such cases.

4.6 Results on CIFAR10

We tested a variety of EENet models on CIFAR10 [16] as well. The results are very similar to that of the tests on SVHN dataset. Table 4.7 and Table 4.8 demonstrate the results of the 6n+2 and the naive ResNet based models, respectively.

EENets designed on 6n+2 architecture of ResNets achieve similar accuracy with their non-EE counterparts while reducing the computational cost upto 12% of the original (e.g. the cost of the EENet-110 model whose early-exit blocks are quadraticly distributed). As seen from Table 4.8, the *quadratic* and *fine* distributed models spend

Table 4.7: Results of 6n+2 based ResNets on CIFAR10. The average number of floating-point operations per example are given in the column of ***FLOP***. ***#Params*** denotes the total number of model parameters. The accuracy is the testing accuracy.

Model	**Accuracy**	**#Params**	**FLOP**
ResNet-20	83.04	0.27M	1.37M
ResNet-32	84.51	0.47M	2.34M
ResNet-44	85.73	0.67M	3.32M
ResNet-110	86.04	1.74M	8.69M

less computational costs than the model having other distribution methods. However, their predictions are not accurate as much as the predictions of the early-exit models distributed through Pareto principle.

The benchmark of EENets with the related work is given in Table 4.9. The results are taken from the original papers. Note that results can change according to implementation and training parameters (e.g. optimizer and learning rate). To avoid the confusion, we have shared the results of ResNets and AlexNets [18] given in the these studies. The accuracy of our ResNets implementation is less than many of the studies. This may be due to our optimizer parameters which we have employed them for both the ResNets and EENets. When comparing the models with the their training parameters, EENets produce the close accuracy of our ResNet implementation (more accurate results can be obtained with smaller λ) by spending less computational cost than the given studies in Table 4.9. Note that the number of layers of some of the studies are not specified since these are not given in the original papers.

In experiments, we have also observed that the additional parameters coming from the early-exit blocks are very small and can be ignored. For example, the number of parameters of an early-exit block of EENet-110 is $\sim$0.0002% of the total parameters of the model with 10-classes datasets such as CIFAR10, MNIST and SVHN.

Table 4.8: Results of the 6n+2 based EENets on CIFAR10. The computational cost rates and the average number of floating-point operations per example are given in the columns of ***Cost*** and ***FLOP***, respectively. The distribution methods of models are given in the first column. ***#E*** denotes the number of EE-blocks. ***Cost Percent of EE-Blocks*** shows the distribution of cost percent of EE-blocks. The given accuracy in the table is the testing accuracy.

	Model	**Acc.**	**#E**	**Cost Percent of EE-Blocks**	**Cost**	**FLOP**
Quadratic	EENet20	80.38	3	<4,11, 24>	0.24	0.33M
	EENet32	81.80	4	<3,8,20,52>	0.20	0.47M
	EENet44	82.16	6	<2,4,10,19,33,66>	0.19	0.62M
	EENet110	82.69	6	<2,4,11,20,40,74>	0.12	1.06M
Fine	EENet20	79.51	3	<5,11,18>	0.18	0.24M
	EENet32	81.68	5	<5,12,16,20,24>	0.20	0.48M
	EENet44	81.86	6	<5,10,16,21,24,33>	0.19	0.63M
	EENet110	83.41	10	<6,10,14,19,23,27,32,36,40,44>	0.13	1.14M
Pareto	EENet20	79.34	2	<24,45>	0.24	0.33M
	EENet32	82.23	5	<20,36,52,68,84>	0.32	0.74M
	EENet44	83.37	6	<21,44,55,66,78,89>	0.36	1.20M
	EENet110	85.35	6	<21,40,53,62,70,74>	0.99	8.70M
GoldenRate	EENet20	79.98	3	<24,45,73>	0.27	0.36M
	EENet32	81.88	5	<12,16,24,52,68>	0.22	0.52M
	EENet44	81.55	6	<7,10,16,24,44,66>	0.16	0.54M
	EENet110	82.84	6	<6,10,15,24,40,66>	0.13	1.17M
Linear	EENet20	80.17	2	<45,73>	0.45	0.62M
	EENet32	82.21	5	<20,36,52,68,84>	0.32	0.75M
	EENet44	77.58	6	<16,33,44,66,78,89>	0.23	0.77M
	EENet110	78.75	10	<10,19,27,40,49,57,66,74,83,91	0.21	1.82M

Table 4.9: Benchmark of related work on CIFAR10. The computational cost rates are given in the columns of ***Cost***. The results are taken from the original papers. Note that results can change according to implementation details and training parameters (e.g. optimizer and learning rate). To avoid the confusion, we have shared the results of ResNets and AlexNets given in the these studies. Consequently, we can compare the results of only convenient work.

Model	**Accuracy**	**Relative Cost**
ResNet-110 [6]	94.39	-
AdaNet-110 [6]	94.24	0.82
AlexNet [2]	78.38	-
B-AlexNet [2]	79.19	0.42
ResNet [2]	80.70	-
B-ResNet [2]	79.17	0.53
ResNet-110 [7]	93.60	-
SkipNet-110 [7]	88.11	0.36
ResNet-18 [12]	94.60	-
ResNet-18-pruned-D [12]	93.00	0.27
Our ResNet-110	86.04	-
EENet-110 (Quadratic)	82.69	0.12
EENet-110 (Fine)	83.41	0.13
EENet-110 (GoldenRate)	82.84	0.13
EENet-110 (Linear)	78.75	0.21
Our ResNet-18	78.50	-
EENet-18	80.34	0.06

CHAPTER 5

CONCLUSION

In this thesis, we propose the Early-exit Convolutional Neural Networks (EENets) to reduce the computational cost of convolutional neural networks (CNN) during inference. EENets have multiple exit blocks and they can classify the examples based on their characteristics at early stages of processing through these exit blocks. Thus, EENets terminate the execution early and avoid wasting the computational cost per example.

The early-exit (EE) blocks of EENets consist of a confidence branch and a classification branch. The confidence branch computes the confidence score of classification and exiting (i.e. stopping the inference process) at current block. On the other hand, the classification branch outputs a classification probability vector. Both branches are learnable and they are independent of each other.

EENets are trained with our proposed loss function which takes not only the classical classification loss but also the computational cost of inference into consideration. As a result, the confidence branches are adapted to the inputs so that less computation is spent for easy examples without harming the model accuracy. In other words, the confidence scores of the EE-blocks are trained in the accuracy and cost trade-off. Thus, they maintain the balance between them during inference. Inference phase is similar to conventional feed-forward networks, however, when the output of a confidence branch reaches a constant threshold (i.e. $T = 0.5$), the inference stops for that specific example and it is classified at current exit block.

Through comprehensive experiments on MNIST, SVHN and CIFAR10 datasets, we evaluate both the 6n+2 and naive ResNet based EENet models. It is observed that

EENets significantly reduce the computational cost (to 2% of the original in ResNet-152 on SVHN) by maintaining the testing accuracy. In addition, the 6n+2 EENet models provide the same or more accuracy just by spending 20% of the total computational cost of non-early-exit versions of these in average.

Note that the idea of EENets is applicable to any feed-forward neural network. However, we demonstrated its use on convolutional neural networks. In this study, we evaluate our model on ResNet based models but other the-state-of-art networks can be easily converted to their early-exit versions. We leave this as future work.

REFERENCES

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

[2] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *International Conference on Pattern Recognition*, 2016.

[3] K. Berestizshevsky and G. Even, "Sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence," *arXiv preprint arXiv:1805.10982*, 2018.

[4] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[5] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," *Design and Automation Test in Europe*, 2016.

[6] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," *European Conference on Computer Vision*, 2018.

[7] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," *European Conference on Computer Vision*, 2018.

[8] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[9] R. T. Mullapudi, W. R.Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Spe-

cialized dynamic architectures for efficient inference," *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[10] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori, "Adaptive neural trees," *arXiv preprint arXiv:1807.06699*, 2018.

[11] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," *International Conference on Machine Learning*, 2017.

[12] W. Hua, C. D. Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," *arXiv preprint arXiv:1805.12549*, 2018.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[14] Y. LeCun, C. Cortes, and C. J. Burges, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *Advances in Neural Information Processing*, 2011.

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical report, University of Toronto*, 2009.

[17] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *ImageNet Large Scale Visual Recognition Challenge*, 2014.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *European Conference on Computer Vision*, 2016.

[21] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep networks with stochastic depth," *European Conference on Computer Vision*, 2016.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[24] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[25] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *International Conference on Machine Learning*, 2015.

[26] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[27] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[28] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *arXiv preprint arXiv:1506.02025*, 2015.

[29] S. Wan, T.-Y. Wu, W. H. Wong, C.-Y. Lee, and C.-Y. Lee, "Confnet: Predict with confidence," *International Conference on Acoustics and Speech and Signal Processing*, 2018.

[30] M. Lin, Q. Chen, and S. Yan, "Network in network," *International Conference on Learning Representations*, 2014.