The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

SAVING COMPUTATIONS BY EARLY INFERENCE TERMINATION

A Thesis in

Computer Science and Engineering

by

Tulika Parija

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2018

The thesis of Tulika Parija was reviewed and approved* by the following:

Chita R. Das
Distinguished Professor of Computer Science and Engineering
Thesis Co-Adviser

John Sampson
Assistant Professor of Computer Science and Engineering
Thesis Co-Adviser

Vijaykrishnan Narayanan
Distinguished Professor of Computer Science and Engineering

Bhuvan Urgaonkar
Associate Professor of Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Machine learning algorithms have seen a revival and a fast growth in popularity due to the recent increase in training data and processing capability of computers. They are being used in a number of different tasks such as image classification, object detection, speech recognition among others. Deep neural networks (DNNs) can be trained to achieve high inference accuracy, and deeper networks correspond to better accuracy of the overall network output, but also incur increasing costs in total computation. However, most networks are flat n-way classifiers, which expend equal effort for all classes in a dataset. This thesis proposes a framework to identify subsets of classes that can be classified with high accuracy using only features extracted from earlier network layers in order to reduce the average computational cost of inference. We apply our framework on the MNIST and CIFAR-10 datasets and demonstrate how our approach makes these networks more amenable for deployment on compute limited endpoint devices. We show up to 52% computation savings (42% latency reduction) for CIFAR-10 with accuracy losses of no more than 1.8%.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all the people who have helped me throughout my graduate program.

First and foremost, I would like to thank my advisor Dr. Chita Das for providing me with invaluable guidance and advice throughout my graduate program. I would also like to thank Dr. Vijaykrishnan Narayanan for providing me with the opportunity to collaborate with him and providing me with indispensable advice at every step of my research. I am extremely grateful to my co-advisor Dr. Jack Sampson for his step by step guidance, patient counseling and invaluable help with my project. I couldn't have achieved what I have without the priceless counsel and support of the three professors.

I would like to thank Jinhang Choi without whose expert knowledge on Caffe I wouldn't have been able to complete this project. I would also like to thank Ashutosh Pattnaik who has helped me whenever I have stumbled upon roadblocks. Last but not the least, I would like to thank all the members of HPCL lab who have provided me constant encouragement along the way.

This material is based on work supported in parts by NSF grant under Award #1317560, # 1439021 and #1629915[1].

[1] Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the awarding agency.

Chapter 1 Introduction

Artificial Neural Networks (ANNs) are computational networks inspired by the human brain. These networks have become ubiquitous in the applications around us. They can be used in a variety of tasks ranging from traditional computer science domains like image, audio, video and text processing to other domains like finance, weather, petroleum engineering, mechanical engineering, etc.

ANNs have gained popularity over the years due to the availability of large amounts of training data and the increase in the processing capabilities of computers. These networks have grown deeper and larger (Deep Neural Networks or DNNs) to increase the accuracy of the task performed. In the domain of vison, Convolutional Neural Networks (more popularly known as CNNs) have been shown to match or outperform human vision. CNNs were introduced by LeCun in [1] for the task of recognizing handwritten digits. The ability of these networks to hierarchically 'learn' feature representations from training examples makes them a frontrunner in the choice for vison algorithms. After Krizhevsky et al demonstrated the benefits of CNNs in image recognition on the historically difficult ImageNet dataset, other researchers started adapting and modifying CNNs for better performance. Over the years, it was proven that adding more layers or making the networks deeper, increased the accuracy of classifying the images. But the increase in accuracy came at the cost of larger computation requirements. Even if one considers training to be a one-time task, deploying these models and testing them, especially on resource constrained endpoints proves to be a challenge. For example, to test a single image, which involves only feed-forward computations, the operations can range from 1 G-Ops to 30 or 40 G-Ops

(AlexNet ~ 1 G-Ops, GoogLeNet ~ 2-3 G-Ops, VGG-Net ~ 15-30 G-Ops)[2]. Energy requirement for devices increases as the number of computations increase. Thus, this becomes a problem in compute limited resources like mobile phones, drones, etc. where saving battery life is important.

The focus of this thesis is to reduce the number of computations in CNNs during the process of inference with minimal loss in accuracy. Although there have been hardware accelerators tailored specifically to make CNNs more efficient and model compression techniques proposed to save memory requirements and computations, not much has been done to leverage the fact that most of the state-of-the-art CNNs have been designed as flat N-way classifiers treating every class as equally distinguishable. Since CNNs learn hierarchical representations of the input, there is a high probability that some of the classes could have had all their features learned at earlier layers in the network. For these classes, the need to go through the entire pipeline is not required. If we can predict with a certain confidence that the features of a certain class have been learned early on, then we can introduce a point of classification for that class, thereby terminating the process of. This would result in saving computations of the later layers, thus reducing energy requirements.

Building on the fact that all classes are not equally distinguishable and that intermediate layers might have learnt abstract concepts for certain classes [2], this thesis proposes a mechanism to identify those classes and terminate the process of inference early for the same classes. As proof of concept, this hypothesis was first tested on LeNet-5 which uses the MNIST dataset and showed that with almost similar accuracy as the original network (within a bound of 0.09%), the number of computations were reduced by ~31%. Similar experiments performed on the CIFAR-10 network showed a tradeoff between

[2] Calculated by using [17]

accuracy and computation with the sweet-spot being a reduction in average computations by ~21% while compromising on average accuracy by ~1%. For applications where a fall in accuracy is not tolerable [for example a pedestrian detection system], we can choose to go with the conservative approach where computations are still reduced with minimal loss in accuracy. In case of other applications [like song recognition in the Google Pixel2 Phone], where conserving battery life is more important, we can choose the method which saves more computation. Thus, we can tune our network and select our points of early termination based on the end application design.

1.1 Background and Related Work

Artificial Neural Networks are computer networks that are modeled after the animal brain. These networks "learn" to perform tasks by examining example inputs to the system. They consist of computational units called artificial neurons arranged into layers. Typically, each layer progressively transforms the input signal and produces an output signal which finally reaches the output layer to produce the desired result.

One of the first ANN algorithms started in the late 1950s was known as Perceptron [18]. It was a linear classifier which linearly transformed the inputs signal by multiplying it with a set of weights to produce an output. But neural network research reached a standpoint because of two reasons: Firstly, single layer perceptrons were not able to reproduce the exclusive-or circuit. Which meant that they could not recognize non-linear patterns. Secondly, there wasn't enough processing power in computers at that time to implement larger neural networks.

After the introduction of techniques such as back propagation [19] and the increase in processing power with the advent of GPUs and powerful servers, neural networks research saw a

rejuvenated interest. The number of hidden layers (layers apart from the input and the output layer) started growing with increased number of neuronal units in each layer.

In 1998, LeCun introduced LeNet-5 [1] which set into motion the slew of work on state-of-the-art machine learning research especially in the computer vision/image domain. LeNet-5 was a 7-layer convolutional neural network used to classify hand-written digits. This network operated on images of resolution of 32x32. To process, higher resolutions, more convolutional layers were required, thus resulting in the requirement of more computation power. Even though LeNet-5 was considered pioneering and really introduced the concept of CNNs, it was Alex Krizhevsky in 2012 who demonstrated the benefits of CNNs in the task of image classification. In his paper, "ImageNet Classification with Deep Convolutional Networks" [4], Krizhevsky introduced a "large, deep convolutional neural network" (commonly called as AlexNet) and used it to classify the historically difficult ImageNet dataset with 1000 possible classes. After AlexNet, a lot of work like VGG-Net [7], GoogLeNet [8], ResNet [9], etc. all reinforced the fact that to achieve better accuracy, more and more layers need to be added. (ResNet has up to 152 layers.)

There are two ways to compute these networks more efficiently: One is to build efficient hardware accelerators [20-22]. Second is to make software level changes [13,23,24]. This thesis focuses on the architecture of the CNN network and makes modifications to the same. Some of the works related to this thesis are mentioned below.

An important fact about Deep Neural Networks is that they progressively learn hierarchical feature representations of training data. They learn complex and abstract features starting from the input layer all the way up-to the output layer. In [2], Jin et al state that although this hierarchical property of learned features has been long identified in existing work, only the output layer features are used for the purpose of classification. Many ignore the heterogeneity and visually discriminative features for different patterns that are distributed across different layers. Thus, in the

paper, they use the information from different layers to build classifiers that work jointly with the final output classifier and aid in increasing accuracy.

As stated in the previous paragraph, features of different complexities that are known to be visually discriminative are distributed across different layers in a deep neural network. However, most of the state-of-the-art networks treat the classes in the dataset as equally distinguishable. Thus, the output is a flat N-way classifier which classifies an image only at the end of the network. Recent work like B-CNN [11] and HD-CNN [10] try to build networks which address the unevenness between the classes. But these papers target improving accuracy of classification instead of early classification to reduce computation. Ioannou et al. [6] builds a hybrid model composed of a CNN and a decision forest to reduce the number of computations. The router in this model is itself a deep CNN model which requires to be trained. Building from this idea of dynamic routing McGill et al. [12] introduce multipath CNN architectures suggesting early classification whenever possible to reduce computation. Even though they perform early classification and prove dynamic routing is more efficient within a computation budget, their baseline architecture is completely different to current state of the art CNNs and they do not seem to provide a direct comparison to those networks. Teerapittayanon et al. [26] propose a method to have branches from the main network that help in fast inference. The branches have the same number of outputs as the last output layer. Compared to the above approaches [11,12] we exploit the property that a trained network has features at early stages sufficient to accurately classify some classes. This identification of the classes makes our method more scalable as the number of classes in a dataset increases. We also use a single fully connected layer in the branches to simplify the branch computations as compared to complex dynamic routers in previous approaches. Moreover, we do not need to retrain our baseline network as done in [11]. Thus, with a static analysis, we obtain competitive accuracy and significant computation reduction which will help in adapting our framework to embedded systems.

To summarize, DNNs are computationally expensive. To make a more specific effort, this thesis considers CNNs and figures out points of early inference termination in the network to save computations. The rest of the thesis is organized in the following way. Chapter 2 explains CNNs and the architecture of the networks used in the thesis. Chapter 2.3 gives brief examples of the dataset. Chapter 3 explains the methodology and experimentation adapted to achieve results. Chapter 4 explains the results achieved and Chapter 5 is the conclusion.

Chapter 2 Convolutional Neural Network Architectures

Early classification in deep neural networks can help us save computations in compute limited resources. Since CNNs are broadly used in speech and image domain, let's use these networks to demonstrate that early termination can lead to computation savings with minimal loss in accuracy. Some of the popular state-of-the-art CNN architectures are AlexNet, GoogleNet, VGGNet which are known to perform reasonably well in the task of image classification. They all involve some layers of convolution, activation, pooling followed by layers of fully-connected neurons. The sections below will explain the basics of a CNN and the architectures of the networks used in this thesis.

2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a kind of neural network, where the data has a grid-like topology. These kinds of networks have been immensely successful in image-based applications like object detection, classification, etc. These networks use convolution in at least one of the layers instead of general matrix multiplication. A typical CNN architecture consists of a stacking a series of convolution layer, activation layer, pooling layer, all followed by a few fully connected layers. Each of the layer is briefly described in the following subsections. For a more detailed discourse on CNNs please see [5,25].

2.1.1 Convolution

Convolution is a mathematical operation of two functions which produces a third modified function. This function is a modified version of one of the original functions, giving the integral of the pointwise multiplication of the two functions as a function of the amount that one of the original functions is translated.

In CNN terminology, the first function is referred to as **input** and the second function is referred to as **kernel**. The output maybe referred to as **feature map**. So, in machine learning, the algorithm will 'learn' the kernel during training. This kernel connects the outputs of a neurons in a local region in the input layer to a single neuron in the next layer. This results in sparse interactions, parameter sharing and equivariant representations [5]. In addition to this, convolution allows us to work with input data of variable size.

2.1.2 Pooling

Pooling basically replaces the output at a certain location with a summary statistic of the network outputs [5]. For example, max pooling gives the maximum output within a rectangular neighborhood. This introduces invariance to small translations in the input. Alternatively, we can say pooling performs down-sampling on the input features.

2.1.3 Activation

The convolution and pooling layers are optionally followed by an activation layer. This is an element-wise function used to introduce non-linearity into the system. There are various types of activation functions that can be used namely sigmoid, tanh, ReLU, etc. [4] showed that using ReLUs allowed CNNs to be trained effectively and introduced data-dependent sparsity.

2.1.4 Fully Connected Layers

This is a layer where all neurons from one layer are fully connected to all the neurons from the previous layer. The last fully connected layer is called the output layer and gives a probability of the classes in the dataset. Thus, the final output can be seen as a one-hot encoding represented by the class scores.

2.1.5 Back Propagation

Although this is not technically a layer, backpropagation is the algorithm used for training. The technique used is stochastic gradient descent. It is implemented in the loss layer. The loss layer used in all the architectures mention below is the SoftMax loss. More details can be found in *Theory of the backpropagation neural network* [19].

2.2 Structure of CNN Architectures

This thesis experiments on three different networks which operate on three different datasets. The architectures of the different baseline networks are mentioned below. The next chapter describes the datasets used.

2.2.1 LeNet-5

LeNet-5 , a pioneering 7-level convolutional network by LeCun et al.[1] classifies digits and was introduced to recognize hand-written numbers on digitized checks. The network used in this thesis consists of two sets of alternating convolution and pooling followed by two fully connected layers. The first fully connected layer (named as ‘ip1’ in the architecture) is followed by

a ReLU activation layer. The final fully connected layer or the output layer consists of ten output classes. This network achieves a classification accuracy of 99.8%. The dataset used here was MNIST.

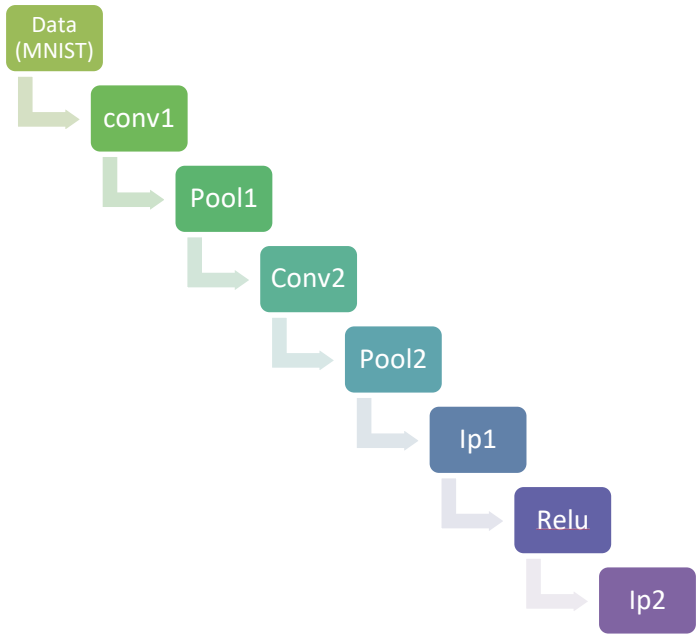

Figure 2-1: Architecture of LeNet-5

2.2.2 CIFAR-10 Network

Very similar to the structure of the LeNet-5, the baseline CIFAR-10 network used here has three sets of alternating convolution and pooling layers followed by two fully connected layers. The difference is in the use of activation layers in between these sets of convolution and pooling layers. For this thesis we use the 'cifar10_quick_train_test' model provided by CAFFE [3]. The decision to use this model instead of state-of-the-art CIFAR10 networks was made because of the quick iterations to test out ideas and experiment with a faster cycle. The idea here was to prove that the hypothesis works and can be extended to various state-of-the-art models. This model has a baseline accuracy of 95.12%

2.3 Datasets

There are several benchmark datasets available publicly for image-based applications. Some of them include MNIST, CIFAR-10, CIFAR -100, ImageNet etc.

2.3.1 MNIST

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. This dataset is widely used for training and testing in the field of machine learning. The MNIST database contains 60,000 training images and 10,000 testing images. The image pixel size is 28x28. These are gray scale images. There are 10 classes ranging from 0-9.

Figure 2-2: MNIST dataset of handwritten digits.[3]

[3] Image Source: https://commons.wikimedia.org/wiki/File:MnistExamples.png

2.3.2 CIFAR-10

The CIFAR-10 dataset is a collection of 60,000 colored images. This database contains 50,000 training images and 10,000 testing images. The image pixel size is 32x32. There are 10 classes. The test set contains 1000 randomly-selected images from each class and training set contains 5000 images from each class.

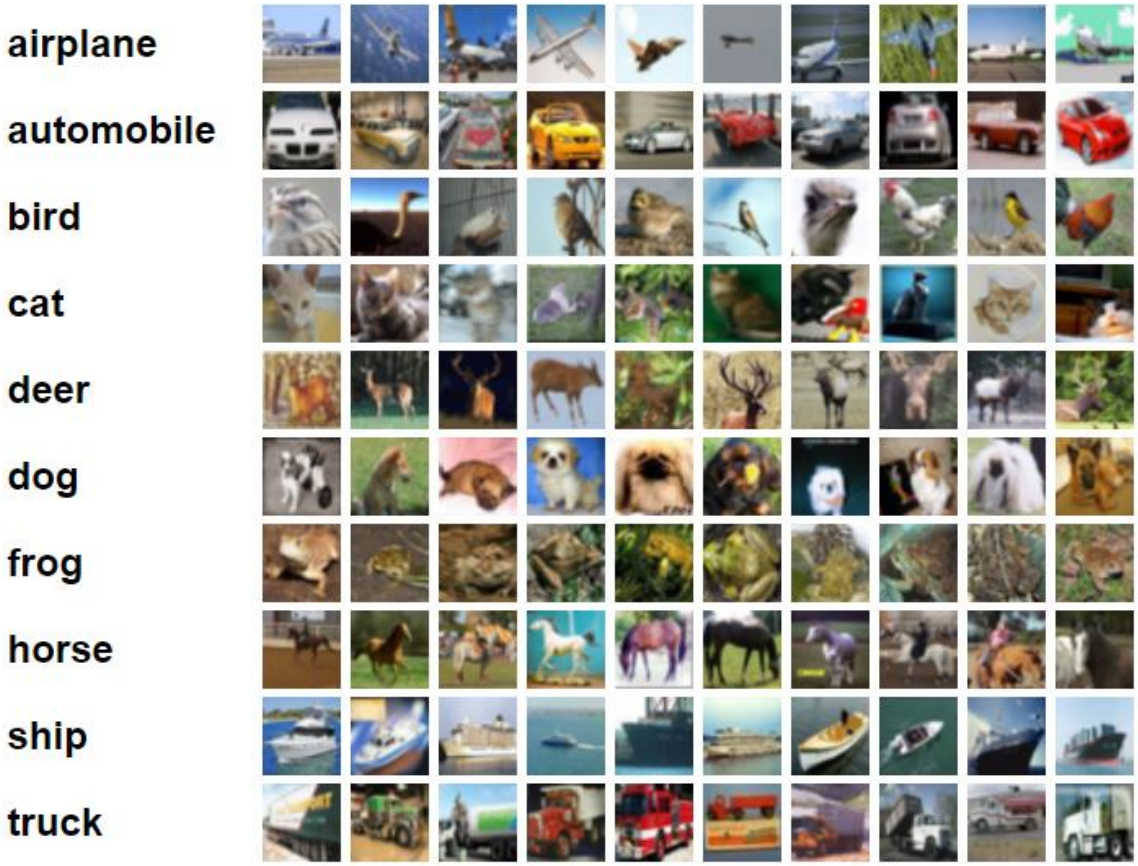

Figure 2-3 CIFAR-10 dataset[4]

[4] Image Source: https://www.cs.toronto.edu/~kriz/cifar.html

Chapter 3 Methodology

This chapter discusses the details of the framework and how the framework is applied to various datasets.

3.1 Details of the framework

We apply our framework, Early Inference Termination (EIT), to obtain a modified network. In this chapter we describe: 1) the scheme for the modified network, 2) the different thresholding mechanisms used to determine if a class can be terminated early, and 3) the datasets and networks used for the experiments.

3.1.1 Scheme

There are two properties that form the basis of our framework: 1. every class is not equally distinguishable and 2. intermediate layers have visually discriminative information [2]. This implies that it might be possible to classify some classes accurately with features learned at earlier layers. To verify this, we perform a characterization study on LeNet and CIFAR-10 network to figure out what classes can be reliably classified at which layer. As we can see in Figure 3-1, we take a pre-trained model for a particular dataset. A typical pre-trained model consists of alternating convolution (conv) and pooling (pool) layers followed by fully connected (FC) layers. At the output of every layer in Figure 3-1(b), we connect a fully connected softmax classifier. These classifiers are trained with the output of the layer they were connected to. Once these classifiers are trained, the outputs are compared to the pre-trained classifier output. The decision regarding whether a class

could be terminated early is made based on a threshold, chosen by one of the methods described below.

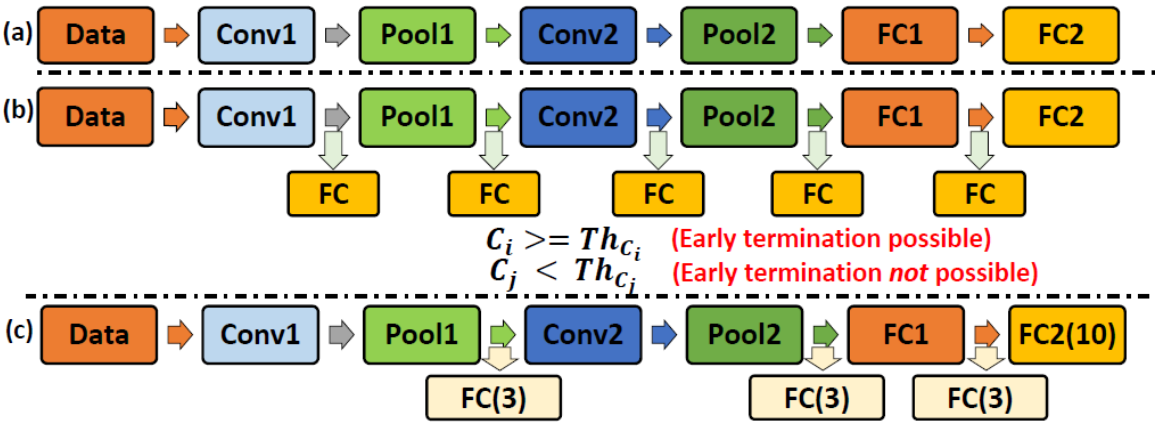

Figure 3-1: Overview of the methodology. (a) takes a pre-trained CNN. (b) Characterize which layers can serve as early termination points for which classes by attaching a classifier similar to the final output layer. (c) Modified Network with points of early terminations. This is obtained after characterizing and figuring out which classes can be terminated where in step (b).

3.1.2 Thresholding mechanisms

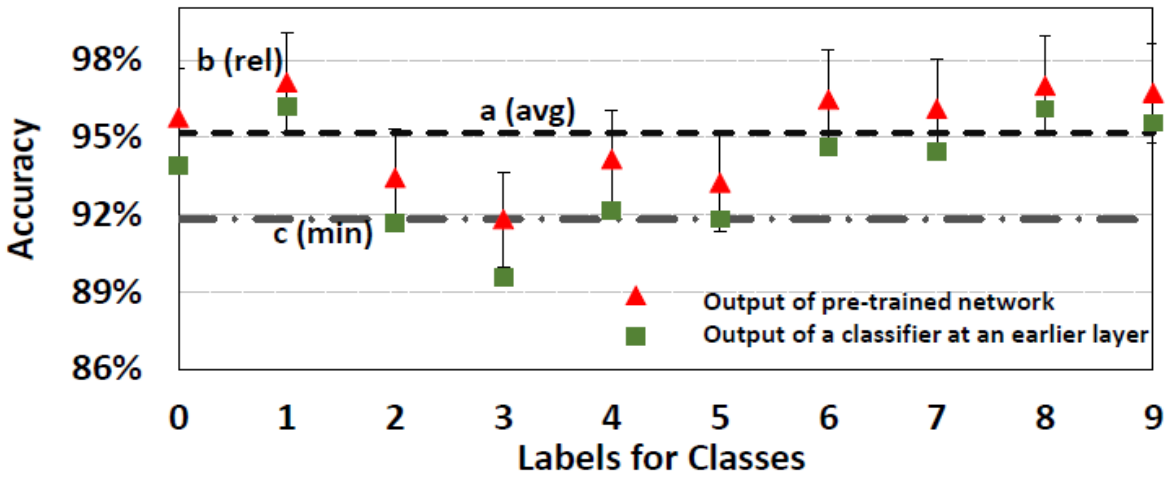

Figure 3-2: Three types of thresholding mechanisms on the output of a classifier connected to the second pooling layer of a CIFAR-10 network. `a' is the average accuracy, `b' is the relative accuracy and `c' is the minimum accuracy of the pre-trained network.

To achieve early termination, we use different thresholds for classification. We consider three mechanisms shown in Figure 3-2:

- (a)Average accuracy as threshold: This means that the classes which have accuracy above the average accuracy of the baseline network qualify for early termination.
- (b) Relative accuracy: This threshold refers to the relative tolerable degradation of accuracy compared to baseline network. In our experiments, we heuristically choose the percentage bound for the threshold depending on the dataset.
- (c) Minimum accuracy as threshold: In this scheme, the minimum accuracy across all classes in the baseline network is chosen as the threshold for acceptable classification accuracy.

The effect of using the different thresholding mechanisms is explained in 3.2.2 and Chapter 4.

3.1.3 Inference

After the classifiers at the points of early termination are trained, the test dataset is used to calculate the accuracy of the network. Figure 3-1(c) shows the process of inference. The data goes through the pipeline up to the first classifier, which is the fully connected layer after pool1. At this point, the data is classified as one of the three classes identified or as a fourth class labeled as "others". Every image classified as "others" is sent further down the pipeline to the next stage. The same process is repeated at each stage. In this work, we have non-overlapping set of classes at the early termination exit points. However, the original output layer provides the entire subset of classes and becomes a point for classification for those images that reach the end of the pipeline.

To measure the efficiency of the scheme, we calculate the average accuracy and the average computations. Accuracy is defined as the number of times the classifier is correct. The accuracy

for each class is calculated based on the outcome of the entire system including the last stage classifier. The total amount of computations is calculated using the NetScope Analyzer [17]. The number of computations is independent of platform and framework used to deploy the neural networks and it directly translates to performance and dynamic energy savings.

3.2 Framework applied to Different Datasets

3.2.1 LeNet-5 with MNIST

As mentioned in Chapter 2.3.1, MNIST is a dataset consisting of 28x28 gray scale images. The following steps were implemented for this network.

3.2.1.1 Feature Extraction

Using the feature extraction tool provided by CAFFE [3], all the different layers' features were extracted. Each extracted feature acted as a new dataset to train the next level classifiers. The extracted features were stored in the LMDB format. This feature extraction was done for both training and testing dataset.

3.2.1.2 Training characterization classifiers

Using the new features, every layer's features acted as a new dataset to train a single layer classifier. The architecture used for this classifier was a single fully connected layer and a SoftMax loss layer for backpropagation to be used during training. This was run for a total of 10000 iterations so that the loss function converged.

3.2.1.3 Thresholding and deciding points of early termination

After training the classifiers, while testing the images, the class-wise precision, recall and accuracy numbers were calculated using a python script. This was useful in the determination of which classes to classify at what point.

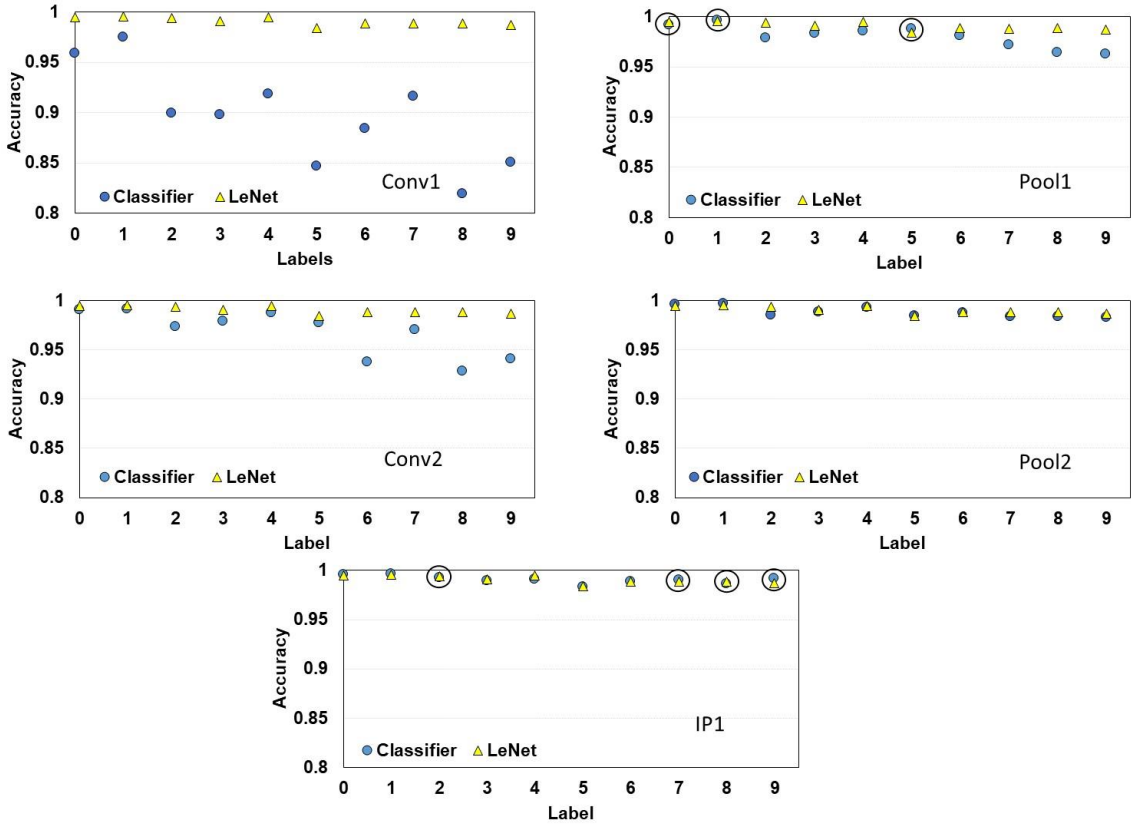

Figure 3-3: Characterization results of MNIST on LeNet layer-wise

Classes	pool1	pool2	pool3
0	x		
1	x		
2			x
3		x	
4		x	
5	x		

6		**x**	
7			**x**
8			**x**
9			**x**

Table 1: Table showing the subset of classes of MNIST that are classified early at different layers of a LeNet

3.2.1.4 Modified Network

From the above characterization curves in Figure 3-3 and Table 1, it can be seen that at pool1 layer, three classes can be classified within a bound of 0.05% of the last layer output. These are classes 0, 1 and 5. Similar classifications can be done for pool2 layer and the penultimate fully connected layer called ip1 in the architecture. Therefore, the modified network has four points of classification much like the network shown in Figure 3-1(c). The ip1 layer has two branches: One is the original network and the other is the modified branch. The reason to branch out at ip1 is that the new branch has only 4 classes. Therefore, the number of computations is still lower than the original branch.

3.2.2 CIFAR-10 Network

The CIFAR-10 Network as mentioned in 2.2.2 is the “cifar10 quick network” provided by CAFFE. Most of the steps followed are similar to the LeNet model. The difference is in the way the thresholding is done. Since the CIFAR-10 dataset is more complex than the MNIST dataset, the distribution of accuracies across classes is not as uniform. This calls for different thresholding mechanisms which can result in tradeoff between accuracy and computation as will be seen in the Results chapter.

3.2.2.1 Feature Extraction

Like the previous case, all the different layers features were extracted using the feature extraction tool provided by CAFFE. Each extracted feature acted as a new dataset to train the next level classifiers. The extracted features were stored in the LMDB format.

3.2.2.2 Training characterization classifiers

Using the new features, every layer's features acted as a new dataset to train a single layer classifier. The architecture used for this classifier were two fully connected layers and a SoftMax loss layer for backpropagation to be used during training. This was run for a total of 10000 iterations so that the loss function converged. Two fully connected layers were selected instead of one to prevent overfitting.

3.2.2.3 Thresholding and deciding points of early termination

After training the classifiers, while testing the images, the class-wise probabilities were calculated using a python script. This helped analyze which layer gave what accuracy for which class.

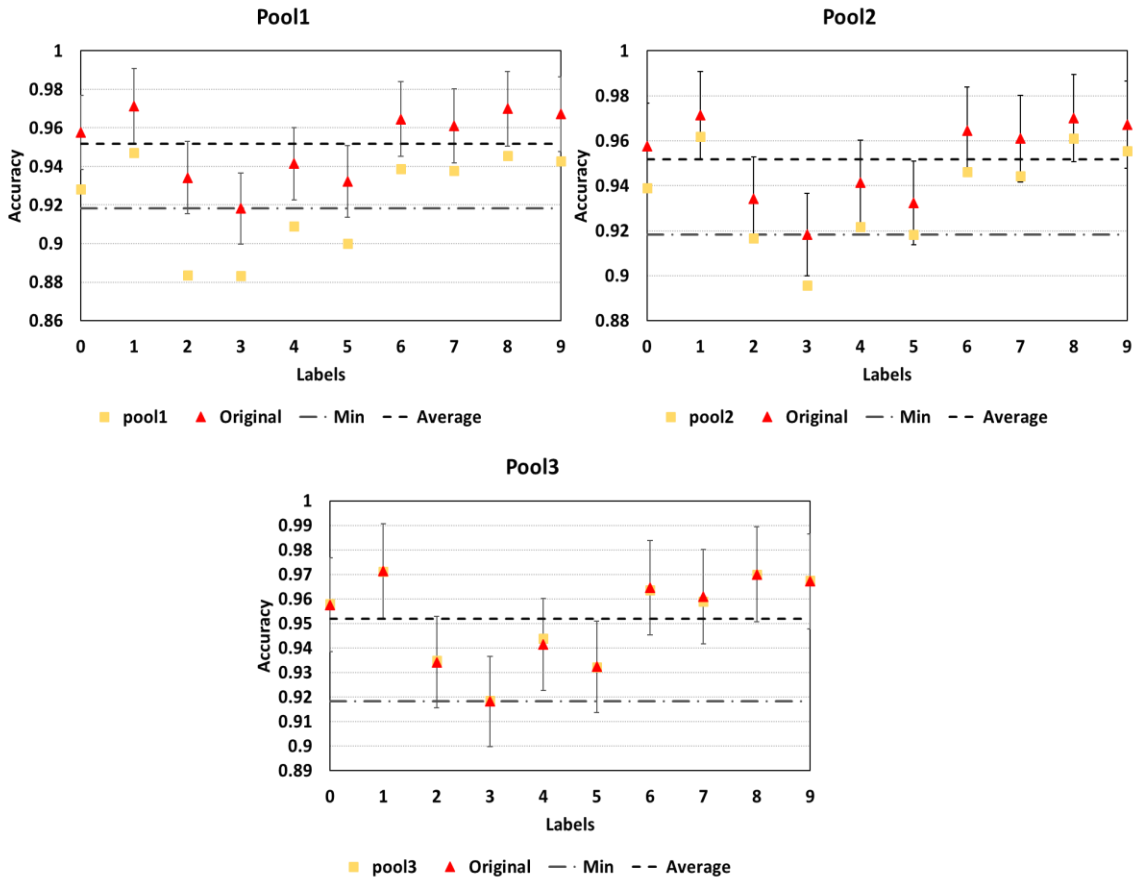

Figure 3-4: Characterization Results for Pooling layers in CIFAR-10 Network

The above figure shows three methods of thresholding and then partitioning the classes. “Min” refers to the minimum accuracy among all the classes in the baseline/Original network. Similarly, “Avg” is the average accuracy considering all the classes. The error bars show the classes that fall within a range of 2% of the original network.

Classes	pool1	pool2	pool3
0	**x**	^	**o**
1	**x**	**o** ^	
2		^	**x**
3			**x** ^
4		**x**	^
5		**x** ^	
6	**x**	^	**o**
7	**x**	^	**o**
8	**x**	**o** ^	
9	**x**	**o** ^	
x=Min o=Average ^=2percent			

Table 2: Table showing the subset of classes of CIFAR-10 that are classified early at different layers of a network

3.2.2.4 Modified Network

Due to the above thresholding, there are a resultant of three different networks. Table 2 shows the result of all three thresholding mechanisms applied to the CIFAR-10 network. For example, in case of the minimum based thresholding, six classes can be classified at the first pooling layer (pool1), two classes at pool2, and two classes after pool3. The results of the accuracy vs computation tradeoffs will be discussed in the next chapter.

Chapter 4 Results and Discussion

In this chapter, we discuss the effects of EIT for MNIST and CIFAR-10 datasets.

4.1 Effects of proposed framework for MNIST

The version of LeNet network used has two sets of alternating convolution and pooling layers, followed by two fully connected layers. We identify three early points of termination and the subset of classes. Out of the 10 classes, three classes are terminated after first pooling layer, three classes after second pooling layer and four classes after the first fully connected layer. We used the third thresholding mechanism, i.e., the relative method with the margin set to 0.05%.

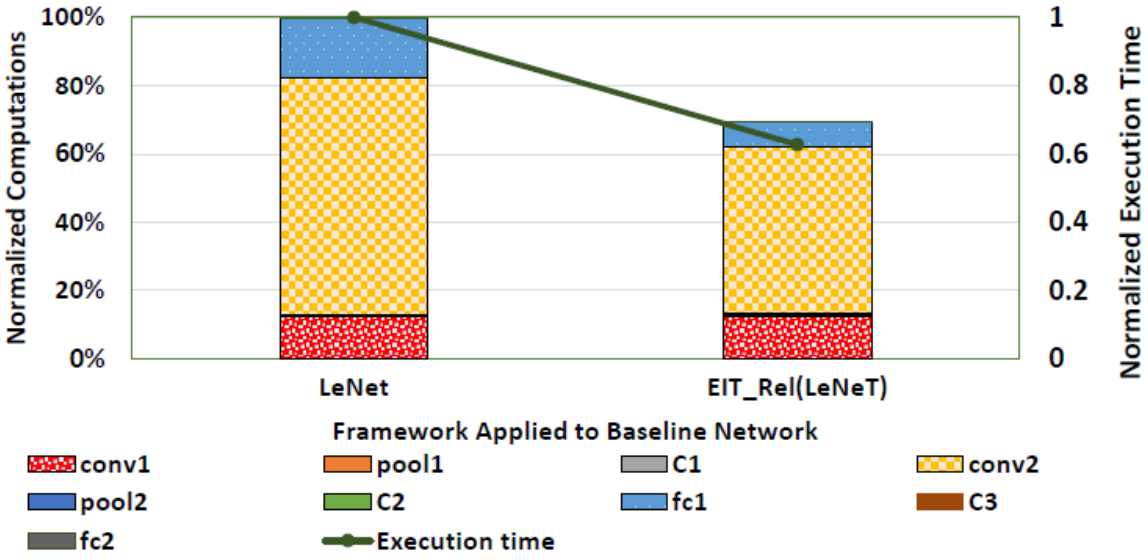

Figure 4-1: Normalized computations of the proposed framework EIT(LeNet) to baseline LeNet. Here conv refers to convolution layers, pool refers to pooling, fc refers to fully connected layers. C_i is the classifier at point for early termination. The green line curve shows the execution time of proposed framework normalized to baseline network.

In Figure 4-1and Figure 4-2

Figure 4-1, our framework reduces computation by 31% while the average accuracy across all classes reduces by 0.09%. This reduction in computation results in a reduction in execution time on a CPU by 38%. The disparity between computation reduction and execution time is a result of underlying architecture and memory hierarchy behaviors.

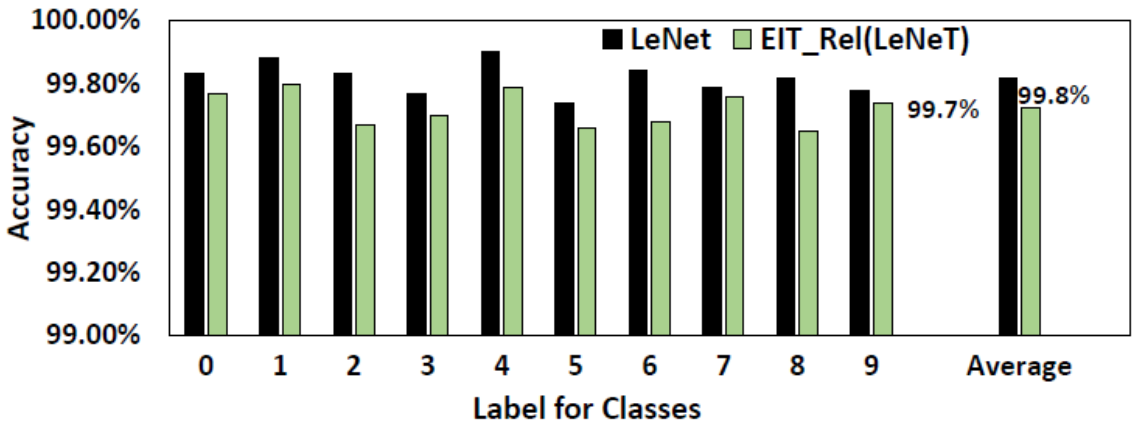

Figure 4-2 : Accuracy of baseline LeNet network vs proposed EIT(LeNet).

4.2 Effects of proposed framework in CIFAR-10

The CIFAR-10 dataset is more diverse than MNIST. We use the three thresholding mechanisms mentioned in 3.1.2 to obtain the subset of classes for early termination. The results are described below:

4.2.1 CIFAR-10 with average accuracy as threshold

We apply this method of thresholding to the baseline network and find classes that have accuracies above the average accuracy of the entire dataset. This method results in the first points of early termination to occur after the second pooling layer. We can see the breakdown in Table 2.

This results in ≈9% computation saving and 13.7% reduction in execution time as seen in Figure 4-3 and Figure 4-4. As a result of this thresholding scheme, the accuracy is very similar with the average accuracy being 94.7% in comparison to 95.17% for the original network.

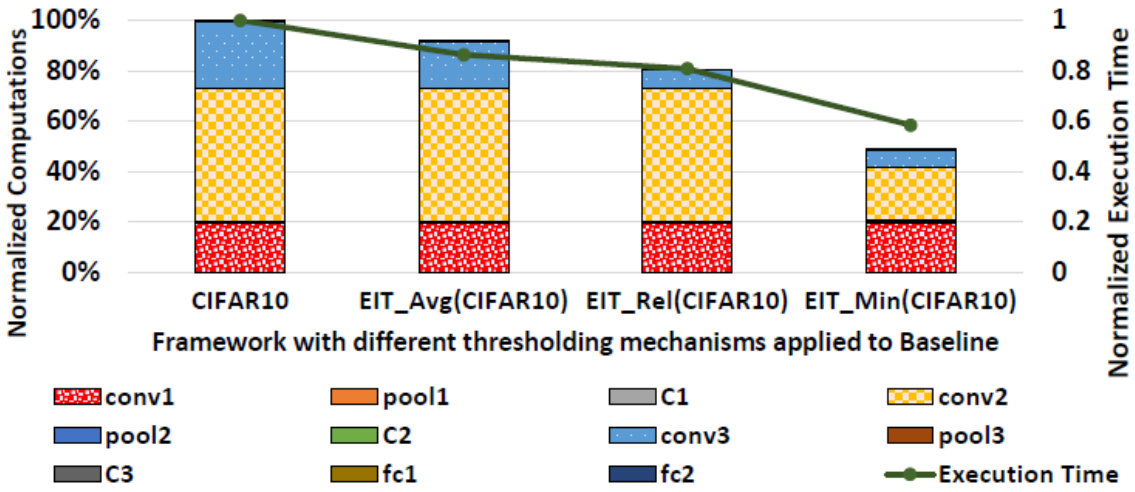

Figure 4-3: Normalized layer-wise computations of EIT(CIFAR10) to baseline CIFAR10 based on a.) Average b.) Relative c.) Minimum thresholding mechanisms. The green line curve shows the respective execution times for modified networks normalized to baseline network.

4.2.2 CIFAR-10 with relative accuracy thresholding

We use 2% as the relative accuracy threshold to select the subset of classes for early termination. This means when the class accuracies are within a range of 2% of the class accuracies in the original network, they can be terminated early. As seen in Figure 4-3 and Figure 4-4, this network results in ≈21% computation savings along with a 20% reduction in execution time with a trade-off of ≈1% in accuracy. Like the average based network, the first point of early termination is after the pool2 layer. However, the majority of the savings comes from the fact that a large number of classes (here, eight classes) are classified at this point. This 2% bound is selected heuristically and can be adjusted depending on the end application design.

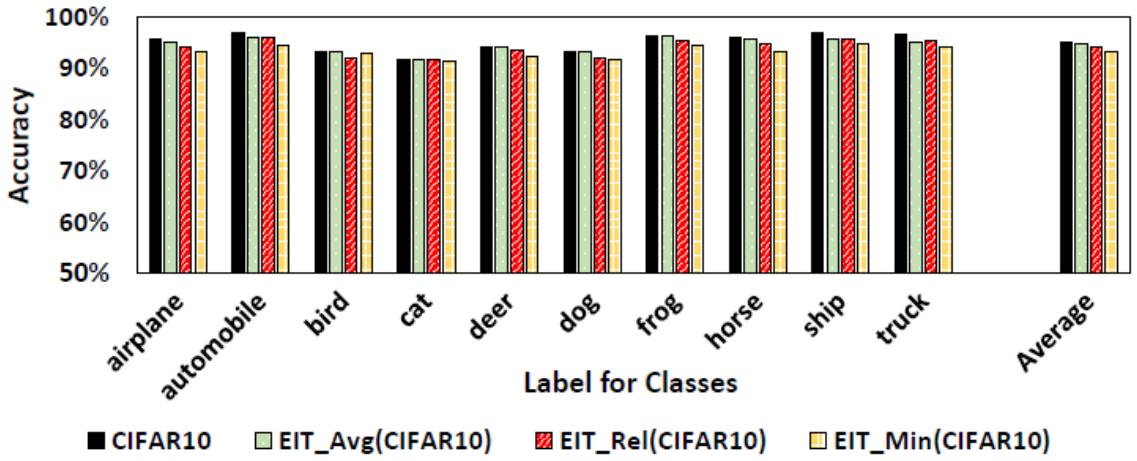

Figure 4-4: Accuracy of baseline CIFAR10 network vs proposed framework applied with different thresholding mechanisms

4.2.3 CIFAR-10 with minimum accuracy as threshold

This method of thresholding is designed to save the maximum computation. As seen in Figure 4-3 and Figure 4-4, we reduce the computation by ≈52% while the drop in accuracy is only 1.8%. The reason for this huge reduction in computation is due to the fact that most classes are classified after the first pooling layer. Therefore, if we look at the breakdown in Figure 4-3, we reduce the computations in the conv2 and conv3 layers by a significant amount. This method is the most aggressive thresholding method and can be used to save maximum computation.

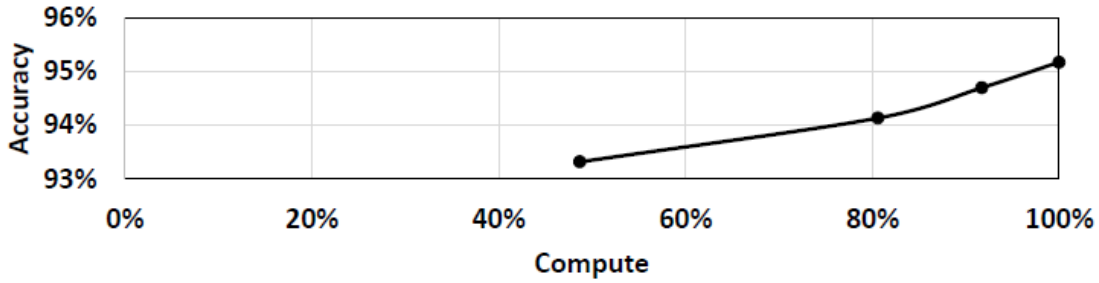

Figure 4-5: Trade-off between accuracy and computation.

The first few layers in a CNN learn simple features like edges and circles [2]. Sometimes these features are enough to identify certain classes in a dataset. Deeper layers or more complex features are useful in improving the accuracy of certain classes. From the above results, we find that if we adopt a more aggressive scheme like minimum based partitioning, we can identify maximum number of classes after the first pooling layer as seen in Table 2. However, if we want to maintain similar accuracy as the baseline network, we might need to go deeper than the first couple of layers. Thus, the other two methods show lower computation saving than the minimum based method.

From Figure 4-5 we can see that there is a trade-off between accuracy and computation depending on the thresholding mechanism used. This trade-off can be treated as a knob depending on the end application design. For applications where a decrease in accuracy is not tolerable (for example, a pedestrian detection system), we can choose a conservative approach where computations are still reduced with minimal loss in accuracy. In case of other applications (e.g. Audio Identification on mobile devices), where conserving battery life is more important, we can choose a more aggressive method that saves more computation. Thus, we can tune our network and select our points of early termination based on the end application design.

Chapter 5 Conclusion

As DNNs become wider and deeper, the increasing amount of computations and intermediate data make it very difficult to map them onto embedded and resource constrained systems. We observe that, during inference, some classes can be terminated in intermediate layers without going through the entire network, which can significantly reduce the amount of computations and temporary data. To this end, we present an approach to identify the classes and the corresponding layers where they are able to terminate without hurting the network accuracy. With our framework, larger DNN networks can thus be applied to resource constrained endpoints (embedded systems, wearables, etc.) to improve responsiveness and save computation effort and associated energy expenditure. For a network trained on the CIFAR-10 dataset, we demonstrate savings of up to 52% computation reduction and 42% improvement in latency.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, 1998.

[2] X. Jin, Y. Chen, J. Dong, J. Feng, S. Yan, “Collaborative Layer-Wise Discriminative Learning in Deep Neural Networks”, 2016. arXiv:1607.05440

[3] http://caffe.berkeleyvision.org/

[4] A. Krizhevsky, I. Sutskever, G. Hinton, “ImageNet Classification with Deep Convolutional Neural Network”, in Advances in Neural Information Processing Systems 25 (NIPS), 2012, pp. 1097–1105.

[5] “Deep Learning” book, Chapter 9.

[6] Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. “Decision forests, convolutional networks and the models in-between.”, arXiv preprint arXiv:1603.01250, 2016.

[7] K. Simonyan and A. Zisserman. “Very Deep Convolutional networks for Large-Scale Image Recognition.” In Proc. Intl Conf. on Learning Representations (ICLR), 2015.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going Deeper with Convolutions.” In Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2015.

[9] K. He, X. Zhang, S. Ren, J. Sun, “Deep Residual Learning for Image Recognition”, http://arxiv.org/abs/1512.03385, 2015.

[10] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, Y. Yu, “HD-CNN: Hierarchical Deep Convolutional Neural Networks for Large Scale Visual Recognition,” In Proceedings of The IEEE International Conference on Computer Vision (ICCV), 2015.

[11] X. Zhu, M. Bain “B-CNN: Branch Convolutional Neural Network for Hierarchical Classification”, arXiv:1709.09890

[12] M. McGill, “Deciding how to decide: Dynamic routing in artificial neural networks,” in Proceedings of the 34th International Conference on Machine Learning, ICML 2017.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I Sutskever, and R. Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” J. Mach. Learn. Res. 15, 1 (January 2014), 1929-1958.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, 1998.

[15] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, 2009.

[16] O. Russakovsky*, J. Deng*, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg and L. Fei-Fei. (* = equal contribution) “ImageNet Large Scale Visual Recognition Challenge”, arXiv:1409.0575, 2014.

[17] David Gschwend, “Netscope CNN Analyzer,” 2016. [Online]. Available: https://github.com/dgschwend/netscope/

[18] F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”, Journal of Psychological Review, 1958.

[19] R. Hecht-Nielsen, "Theory of the backpropagation neural network," International 1989 Joint Conference on Neural Networks, Washington, DC, USA, 1989, pp. 593-605 vol.1.

[20] M. Alwani, H. Chen, M. Ferdman, and P. Milder, “Fused-layer CNN accelerators” in Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO),2016.

[21] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” International Conference on Learning Representations (ICLR),2016.

[22] N. P. Jouppiet al., “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in Proceedings of the 44th ACM Annual International Symposium on Computer Architecture (ISCA), 2017.

[23] S. Han et al., “EIE: Efficient Inference Engine on Compressed Deep Neural Network,” in Proceedings of the 43rd ACM Annual International Symposium on Computer Architecture (ISCA), 2016.

[24] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally and Kurt Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 1 MB model size”, http://arxiv.org/abs/1602.07360, 2016.

[25] http://cs231n.github.io/convolutional-networks/

[26] S. Teerapittayanon, B. McDanel, and H. Kung, “Branchynet: Fast inference via early exiting from deep neural networks.”, CVPR, 2017.