BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks

Surat Teerapittayanon  
Harvard University  
Email: steerapi@seas.harvard.edu

Bradley McDanel  
Harvard University  
Email: mcdanel@fas.harvard.edu

H.T. Kung  
Harvard University  
Email: kung@harvard.edu

Abstract—Deep neural networks are state of the art methods for many learning tasks due to their ability to extract increasingly better features at each network layer. However, the improved performance of additional layers in a deep network comes at the cost of added latency and energy usage in feedforward inference. As networks continue to get deeper and larger, these costs become more prohibitive for real-time and energy-sensitive applications. To address this issue, we present BranchyNet, a novel deep network architecture that is augmented with side branches. The architecture allows prediction results for a large portion of test samples to exit the network early via these branches when samples can already be inferred with high confidence. BranchyNet exploits the observation that features learned at an early layer of a network may often be sufficient for the classification of many data points. For more difficult samples, which are expected to be infrequent, BranchyNet will use further or all network layers to provide the best likelihood of correct prediction. We study the BranchyNet architecture using several well-known networks (LeNet, AlexNet, ResNet) and datasets (MNIST, CIFAR10) and show that it can both improve accuracy and significantly reduce the inference time of the network.

I. INTRODUCTION

One of the reasons for the success of deep networks is their ability to learn higher level feature representations of data at successive nonlinear layers. In recent years, advances in both hardware and learning techniques have emerged to train even deeper networks, which have improved classification performance further [2, 6]. The ImageNet challenge exemplifies the trend to deeper networks, as the state of the art methods have advanced from 8 layers (AlexNet), to 19 layers (VGGNet), and to 152 layers (ResNet) in the span of four years [5, 11, 16]. However, the progression towards deeper networks has dramatically increased the latency and energy required for feedforward inference. For example, experiments that compare VGGNet over AlexNet on a Titan X GPU have shown a factor of 20x increase in runtime and power consumption for a reduction in error rate of around 4% (from 11% to 7%) [9]. The trade off between resource usage efficiency and prediction accuracy is even more noticeable for ResNet, the current state of the art method for the ImageNet Challenge, which has an order of magnitude more layers than VGGNet. This rapid increase in runtime and power for gains in accuracy may make deeper networks less tractable in many real world scenarios, such as real-time control of radio resources for next-generation mobile networking, where latency and energy are important factors.

To lessen these increasing costs, we present BranchyNet, a neural network architecture where side branches are added to the main branch (the original baseline neural network) to allow certain test samples to exit early. This novel architecture exploits the observation that it is often the case that features learned at earlier stages of a deep network can already correctly infer a large subset of the data population. By exiting these samples with prediction at earlier stages and thus avoiding layer-by-layer processing for all layers, BranchyNet significantly reduces the runtime and energy use of inference for the majority of samples. Figure 1 shows how BranchyNet modifies a standard three-layer network by adding two branches with their respective exit points.

BranchyNet is trained by solving a joint optimization problem on the weighted sum of the loss functions associated with the exit points. Once the network is trained, BranchyNet utilizes the exit points to allow early exit of prediction results to reduce the cost of inference. At each exit point, BranchyNet uses the entropy of a classification result by a classifier (e.g., by softmax) as a measure of confidence in the prediction. If the entropy of a test sample is below a learned threshold value, meaning that the classifier is confident in the prediction, the sample exits the network with the prediction result at this exit point, and is not processed by the higher network layers. If the entropy value is above the threshold, then the classifier at this exit point is deemed to be not confident enough, and the sample continues to the next exit point in the network. If the sample reaches the last exit point, which is the last layer of the
baseline neural network, it will always perform classification at that point.

Three main contributions of this paper are:

- **Fast Inference with Early Exit Branches**: BranchyNet exits the majority of the samples at earlier exit points, thus reducing layer-by-layer weight computation and I/O costs, resulting in runtime and energy savings.

- **Regularization via Joint Optimization**: BranchyNet jointly optimizes the weighted loss of all exit points. Each exit point provides regularization on the others, thus preventing overfitting and improving test accuracy.

- **Mitigation of Vanishing Gradients**: Early exit points provide additional more immediate gradient signal in back propagation, resulting in more discriminative features in lower layers, thus improving accuracy.

II. **BACKGROUND AND RELATED PRIOR WORK**

LeNet-5 [13] introduced the standard convolutional neural networks (CNN) structure which is composed of stacked convolutional layers (weight layers), optionally followed by contrast normalization and maxpooling, and then finally followed by one or more fully-connected layers. This structure has performed well in several image tasks such as image classification. AlexNet [11], VGG [16], ResNet [5] and others have expanded on this structure with their own innovative approaches to make it deeper and larger for improved classification accuracy. However, deeper and larger models incur additional computations which may not be suitable for fast and energy-efficient inference.

Due to the computational costs of deep networks, improving the efficiency of feedforward inference has been heavily studied. Two such approaches are network compression and implementation optimization. Network compression schemes aim to reduce the the total number of model parameters of a deep network and thus reduce the amount of computation required to perform inference. Bucilua et al. (2006) proposed a method of compressing a deep network into a smaller network that achieves a slightly reduced level of accuracy by retraining a smaller network on synthetic data generated from a deep network [1]. More recently, Han et al. (2015) have proposed a pruning approach that removes network connections with small contributions [3]. However, while pruning approaches can significantly reduce the number of model parameters in each layer, converting that reduction into a significant speedup is difficult using standard GPU implementations due to the lack of high degrees of exploitable regularity and computation intensity in the resulting sparse connection structure [4]. Kim et al. (2015) use a Tucker decomposition (a tensor extension of SVD) to extract shared information between convolutional layers and perform rank selection [9]. This approach reduces the number of network parameters, making the network more compact, at the cost of a small amount of accuracy loss. These network compression methods are orthogonal to the BranchyNet approach taken in this paper, and could potentially be used in conjunction to improve inference efficiency further.

Implementation optimization approaches reduce the runtime of inference by making the computation algorithmically faster. Vanhoucke et al. (2011) explored code optimizations to speed up the execution of convolution neural networks (CNNs) on CPUs [20]. Mathieu et al. (2013) showed that convolution using FFT can be used to speed up training and inference for CNNs [15]. Recently, Lavin et al. (2015) have introduced faster algorithms specifically for 3x3 convolutional filters (which are used extensively in VGGNet and ResNet) [12]. In contrast, BranchyNet makes modifications to the network structure to improve inference efficiency.

Deeper and larger models are complex and tend to overfit the data. Dropout [17], L1 and L2 regularization and many other techniques have been used to regularize the network and prevent overfitting. Additionally, Szegedy et al. (2015) introduced the concept of adding softmax branches in the middle layers of their inception module within deep networks as a way to regularize the main network [19]. While also providing similar regularization functionalities, BranchyNet has a new goal of allowing early exits for test samples which can already be classified with high confidence.

One main challenge with (very) deep neural networks is the vanishing gradient problem. Several papers introduced ideas to mitigate this issue including normalized network initialization [2, 14] and intermediate normalization layers [8]. Recently, new approaches such as Highway Networks [18], ResNet [5], and Deep Networks with Stochastic Depth [7] have been studied. The main idea is to add skip (shortcut) connections in between layers. This skip connection is an identity function which helps propagate the gradients in backpropagation training for neural networks. Although performance gains of BranchyNet such as being able to improve classification accuracy of the baseline network can in part be attributed to shortcuts (see Section V-D), BranchyNet has the distinct design goal of allowing early exits as noted earlier.

III. **BranchyNet**

BranchyNet modifies the standard deep network structure by adding exit branches (also called side branches or simply branches for brevity), at strategic locations throughout the network. These early exit branches allow samples which can be accurately classified in early stages of the network to exit early. In training the classifiers at these exit branches, we in addition consider network regularization and mitigation of vanishing gradients in backpropagation. For the former, branches will provide regularization on the main branch (baseline network), and vice versa. For the latter, a relatively shallower branch at a lower layer will provide more immediate gradient signal in backpropagation, resulting in discriminative features in lower layers of the main branch, thus improving its accuracy.

In designing the BranchyNet architecture for allowing early exit of predictions on certain test samples, we address a number of design considerations, including (1) locations of branch points, (2) structure of a branch (weight layers, fully-connected layers, etc.) as well as its size and depth, (3) classifier at the exit point of a branch, (4) exit criteria for a branch and the associated test cost against the criteria, and (5) training of classifiers at exit points of all branches. In
general, this “branch” notion can be recursively applied, that is, a branch may have branches, resulting in tree branches. For simplicity, in this paper we focus a basic scenario where there are only one-level branches which do not have nested branches, meaning there are no tree branches.

In this paper, we will describe BranchyNet with classification tasks in mind; however, the architecture is general and can also be used for other tasks such as image segmentation and object detection.

A. Architecture

In a BranchyNet network, an entry point is where the input enters the network and an exit point is where the output exits the network. A BranchyNet can have one or more exit points. A branch is a subset of the network containing contiguous layers which are not overlapped with other branches, and an exit point. The main branch is the baseline (original) network before side branches are added. From the lowest branch to highest branch, we number branches and their associated exit points with increasing integers starting with 1. For example, the branch for exit point 1 has the shortest path to the entry point, as illustrated in Figure 1.

B. Training BranchyNet

For a classification task, softmax cross entropy loss function is commonly used as the optimization objective. Here we describe how BranchyNet uses this loss function. Let \( y \) be a one-hot ground-truth label vector, \( x \) be an input sample and \( C \) be the set of all possible labels. The objective function can be described as:

\[
L(y, \hat{y}; \theta) = -\frac{1}{|C|} \sum_{c \in C} [y_c \log \hat{y}_c + (1 - y_c) \log(1 - \hat{y}_c)],
\]

where

\[
\hat{y} = \text{softmax}(z) = \frac{\exp(z)}{\sum_{c \in C} \exp(z_c)},
\]

and

\[z = f_{\text{exit}_n}(x; \theta),\]

where \( f_{\text{exit}_n} \) is the output of the \( n \)-th exit branch and \( \theta \) represents the parameters of the layers from an entry point to the exit point.

The design goal of each exit branch is to minimize this loss function. To train the entire BranchyNet, we form a joint optimization problem covering all exit branches in the network as a weighted sum of loss functions of all exit branches:

\[
L_{\text{branchynet}}(\hat{y}, y; \theta) = \sum_{n=1}^{N} w_n L(\hat{y}_{\text{exit}_n}, y; \theta),
\]

where \( N \) is the total number of exit points. Section V-A discusses how one might choose weights \( w_n \).

The backpropagation algorithm is used to solve this optimization. The algorithm consists of two steps: the feedforward pass and the backward propagation. In the feedforward pass, the training data set is passed through the network, including both main and side branches, and the output from the neural network at all exit points is recorded and the error of the network is calculated. In backward propagation, the error is passed back through the network and the weights of the neural network are updated using gradient descent. For gradient descent, we use Adam algorithm [10], though other variants of Stochastic Gradient Descent (SGD) can also be used.

C. Fast Inference with BranchyNet

Once trained, BranchyNet can be used for fast inference with early exiting based on the algorithm in Figure 2. The main idea is that if the classifier at an exit point of a branch has high confidence about its classification of a test sample \( x \), the sample can be exited and return predicted result early and no further computation performed by the higher branches in the network. We will use entropy as a measure of how confidence the classifier at an exit point is about the sample. Entropy is defined as:

\[
\text{entropy}(y) = \sum_{c \in C} y_c \log y_c,
\]

where \( y \) is a vector containing computed probabilities for all possible class labels and \( C \) is a set of all possible labels.

To perform fast inference on a given BranchyNet network, we follow the procedure as described in Figure 2. The procedure takes two arguments: \( x \) is an input sample and \( T \) is a vector where each \( n \)-th entry \( T_n \) is the threshold for determining whether to exit the sample at the \( n \)-th exit point and \( N \) is the number of exit points of the network.

Fig. 2: Branchy Fast Inference Algorithm. \( x \) is an input sample, \( T \) is a vector where each \( n \)-th entry \( T_n \) is the threshold used as a criteria whether to exit the sample at the \( n \)-th exit point. For each exit point, the input sample is fed through the corresponding branch. The procedure then calculates softmax and entropy of the output and checks the entropy against the threshold \( T_n \) of that exit point. If the entropy is less than the threshold, the class label with the maximum score (probability) is returned. Otherwise, the procedure continues to the next exit point. If the samples reaches the last exit point, the label with the maximum score is always returned.

IV. RESULTS

In this section, we demonstrate the effectiveness of BranchyNet by adapting three different widely-studied convolutional neural networks on image classification task: LeNet, AlexNet and ResNet. We evaluate BranchyLeNet on the
Our experiments show that BranchyNet running on GPU is faster than BranchyNet running on CPU. For B-LeNet, the accuracy is slightly better than a single network, but the inference is performed significantly faster. For B-ResNet, the accuracy is slightly lower than the baseline. A different threshold could be chosen which gives accuracy higher than ResNet but with much less savings in inference time. The performance characteristics of BranchyNet running on CPU follow a similar trend to the performance of BranchyNet running on GPU.

For simplicity, we only describe convolutional layers and fully-connected layers of the networks. Generally, these networks may also contain max pooling, non-linear activation functions (e.g., a rectified linear unit and sigmoid), normalization (e.g., local response normalization, batch normalization), and dropout.

For LeNet-5 [13] which consists of 3 convolutional layers and 2 fully-connected layers, we add a branch consisting of 1 convolutional layer and 1 fully-connected layer after the first convolutional layer of the main network. We call this BranchyLeNet (B-LeNet). For AlexNet [11] which consists of 5 convolutional layers and 3 fully-connected layers, we add 2 branches. One branch consisting of 2 convolutional layers and 2 fully-connected layers is added after the 1st convolutional layer of the main network, and another branch consisting of 1 convolutional layer and 2 fully-connected layers is added after the 2nd convolutional layer of the main network. We call this BranchyAlexNet (B-AlexNet). For ResNet-110 [5] which consists of 109 convolutional layers and 1 fully-connected layer, we add 2 branches. One branch consisting of 3 convolutional layers and 1 fully-connected layer is added after the 2nd convolutional layer of the main network, and the second branch consisting of 2 convolutional layers and 1 fully-connected layer is added after the 37th convolutional layer of the main network. We call this BranchyResNet (B-ResNet). For B-AlexNet and B-ResNet, we use the same number of fully-connected layers as in the original network. For B-LeNet, we found that one fully-connected layer provided good performance for the branch. We initialize B-LeNet, B-AlexNet and B-ResNet with weights trained from LeNet, AlexNet and ResNet respectively. We found the initializing each BranchyNet network with the weights trained from the baseline network improved the classification accuracy of the network by several percent over random initialization. To train these networks, we use Adam algorithm with a step size ($\alpha$) of 0.001 and exponential decay rates for first and second moment estimates ($\beta_1$, $\beta_2$) of 0.99 and 0.999 respectively.

Figure 3 shows the GPU performance results of BranchyNet when applied to each network. For all of the networks, BranchyNet outperforms the original baseline network. The reported runtime is the average among all test samples. B-LeNet has the largest performance gain due to a more efficient branch which achieves almost the same level of accuracy as the last exit branch. For AlexNet and ResNet, we see that the performance gain is still substantial, but since more samples are required to exit at the last layer, smaller than B-LeNet. The knee point denoted as the green star represents an optimal threshold point, where the accuracy of BranchyNet is comparable to the main network, but the inference is performed significantly faster. For B-ResNet, the accuracy is slightly lower than the baseline. A different threshold could be chosen which gives accuracy higher than ResNet but with much less savings in inference time. The performance characteristics of BranchyNet running on CPU follow a similar trend to the performance of BranchyNet running on GPU.

Table I highlights the selected knee threshold values, exit (%), and gain in speed up, for BranchyNet for each network for both CPU and GPU. The $T$ column denotes the threshold values for each exit branch. Since the last exit branch must exit all samples, it does not require an exit threshold. Therefore, for a 2-branch network, such as B-LeNet, there is a single $T$ value and for a 3-branch network, such as B-AlexNet and B-ResNet, there are two $T$ values. Further analysis of the sensitivity of the $T$ parameters is discussed in Section V. The Exit (%) column shows the percentage of samples exited at each branch point. For all networks, we see that BranchyNet is able to exit a large percentage of the test samples before the last layer, leading to speedups in inference time. B-LeNet exits 94% of samples at the first exit branch, while B-AlexNet and B-ResNet exit 65% and 41% respectively. Exiting these samples early translate to CPU/GPU speedup gains of 5.4/4.7x over LeNet, 1.5/2.4x over AlexNet, and 1.9/1.9x over ResNet. The current branch structure for B-ResNet mimics that of B-AlexNet. The achieved speedup (1.9x) is already significant. However, optimized placements of branches for ResNet will likely yield higher speedups given the substantial accuracy gains of Branchy-20 over ResNet-20 as reported in Figure 8.

### TABLE I: Selected performance results for BranchyNet on different network structures. The BrachyNet rows correspond to the knee points denoted as green stars in Figure 3.

<table>
<thead>
<tr>
<th>Network</th>
<th>Acc. (%)</th>
<th>Time (ms)</th>
<th>Gain Thrshld.</th>
<th>$T$</th>
<th>Exit (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>99.20</td>
<td>3.37</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-LeNet</td>
<td>99.25</td>
<td>5.62</td>
<td>0.005</td>
<td>0.94</td>
<td>56.3</td>
</tr>
<tr>
<td>AlexNet</td>
<td>78.18</td>
<td>9.56</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-AlexNet</td>
<td>79.19</td>
<td>6.32</td>
<td>1.5x</td>
<td>0.0001</td>
<td>65.6</td>
</tr>
<tr>
<td>ResNet</td>
<td>80.70</td>
<td>13.70</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-ResNet</td>
<td>79.17</td>
<td>73.5</td>
<td>1.9x</td>
<td>0.3</td>
<td>41.5</td>
</tr>
<tr>
<td>LeNet</td>
<td>99.20</td>
<td>1.58</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-LeNet</td>
<td>99.25</td>
<td>0.34</td>
<td>4.7x</td>
<td>0.0025</td>
<td>94.3</td>
</tr>
<tr>
<td>AlexNet</td>
<td>79.19</td>
<td>3.15</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-AlexNet</td>
<td>79.19</td>
<td>1.30</td>
<td>2.4x</td>
<td>0.0001</td>
<td>65.6</td>
</tr>
<tr>
<td>ResNet</td>
<td>80.70</td>
<td>70.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B-ResNet</td>
<td>79.17</td>
<td>37.2</td>
<td>1.9x</td>
<td>0.3</td>
<td>41.5</td>
</tr>
</tbody>
</table>

### V. ANALYSIS AND DISCUSSION

In this section, we provide additional analysis on key aspects BranchyNet.

#### A. Hyperparameter sensitivity

Two important hyperparameters of BranchyNet are the weights $w_n$ in joint optimization (Section III-B) and the exit thresholds $T$ for the fast inference algorithm described in Figure 2. When selecting the weight of each branch, we observed that giving more weight to early branches improves the accuracy of the later branches due to the added regularization. On a simplified version of BranchyAlexNet with only the first and last branch, weighting the first branch with 1.0 and the last branch with 0.3 provides a 1% increase in classification accuracy over weighting each branch equally. Giving more weight to earlier exit branches encourages more discriminative feature learning in early layers of the network and allows more samples to exit early with high confidence.

Figure 4 shows how the choice of $T$ affects the number of samples exited at the first branch point in B-AlexNet. We
observe that the entropy value has a distinctive knee where it rapidly becomes less confident in the test samples. Thus in this case it is relatively easy to identify the knee and learn a corresponding threshold. In practice, the choice of exit threshold for each exit point depends on applications and datasets. The exit thresholds should be chosen such that it satisfies the inference latency requirement of an application while maintaining the required accuracy. In general, these thresholds should be screened and an appropriate exit threshold selected for the application.

B. Effects of structure of branches

Figure 5 shows the impact on the accuracy of the last exit by adding additional convolutional layers in an earlier side branch for a modified version of B-AlexNet with only the first side branch. We see that there is a optimal number of layers to improve the accuracy of the main branch, and that adding too many layers can actually harm overall accuracy. In addition to convolutional layers, adding a few fully-connected layers after convolutional layers to a branch also helps since this allows local and global features to combine and form more discriminative features. The number of layers in a branch and the size of an exit branch should be chosen such that the overall size of the branch is less than amount of computation needed to do to exit at a later exit point. Generally, we find that earlier branch points should have more layers, and later branch points should have fewer layers.

C. Effects of cache

Since the majority of samples are exited at early branch points, the later branches are used more rarely. This allows weights at these early exit branches to be cached more efficiently. Figure 6 shows the effect of cache based on various $T$ values for B-AlexNet. We see that the more aggressive $T$ values have faster runtime on the CPU and also less cache miss rates. One could use this insight to select a branch structure that can fit more effectively in a cache, potentially speeding up inference further.

D. Relationship to ResNet

A restricted case of BranchyNet can be considered as achieving the effect of shortcuts (skip connections) as in ResNet [5], where we constrain the layers after the skip connection in the original network to be the same as the layers in the exit branch.

To illustrate how BranchyNet and ResNet are related, consider Figure 10. Assume a restricted case of BranchyNet where the joint optimization uses the same weights for the main branch and the side branch, and yields the same softmax classifier for both branches. Suppose we fold the side branch on top of the main branch as shown in the figure. Then, we can see that backward propagation on the side branch in BranchyNet follows the shortcut (identity) in ResNet. As ResNet papers (e.g., [5, 18]) have elegantly pointed out, learning of the residual function is effective in mitigating the vanishing gradient problem. This advantage partially explains why early exits in BranchyNet can support accurate classification and the somewhat surprising result that joint optimization used by BranchyNet can help improve the accuracy achieved by the main branch. Figure 7, 8 and 9 compare the training losses,
test accuracies and speedups on CIFAR-10 dataset between the ResNet-20 (19 convolutional layers and 1 fully-connected layer) and the Branchy-20 (ResNet-20 where each shortcut is turned into a branch with 1 fully-connected layer). Branchy-20 is 4.6x faster and has 16.5% higher accuracy than ResNet-20, due to its additional flexibility in joint optimization through the use of different weights for each exit point. In addition, Branchy-20 provides regularization benefits as described in Section III-A.

Fig. 7: Comparison of training loss for Branchy-20 and ResNet-20 on the CIFAR-10 dataset. Branchy-20 is ResNet-20 where each shortcut is turned into a branch with 1 fully-connected layer.

Fig. 8: Comparison of main network accuracy for Branchy-20 and ResNet-20 on the CIFAR-10 dataset.

Fig. 9: GPU performance results of Branchy-20 and ResNet-20. Branchy-20 is 4.6x times faster with 16.5% higher accuracy than ResNet-20.

VI. CONCLUSION
We have proposed BranchyNet to promote early return of predicted results via early exits from branches. Through proper branching structures and exit criteria as well as joint optimization of loss functions for all exit points, the architecture is able to leverage the fact that some test samples can be correctly classified early and hence need not to go deeper into the network. We have evaluated this approach on several popular network architectures and shown that BranchyNet can reduce the inference cost of deep neural networks by provide 2x-6x speed up on both CPU and GPU.

BranchyNet is a toolbox for researchers to use on any deep network models for fast inference. BranchyNet can be used in conjunction with prior works such as network pruning and network compression [1, 3]. BranchyNet can be adapted to solve other types of problems such as image segmentation, and is not just limited to classification problems.

We plan to further explore the following aspects of BranchyNet: automatic discovery of exit threshold that will maximize the overall accuracy and more complex structures of BranchyNet such as branch trees.

REFERENCES


