EPNet: Learning to Exit with Flexible Multi-Branch Network

Xin Dai
Worcester Polytechnic Institute
Worcester, MA
xdai5@wpi.edu

Xiangnan Kong
Worcester Polytechnic Institute
Worcester, MA
xkong@wpi.edu

Tian Guo
Worcester Polytechnic Institute
Worcester, MA
tian@wpi.edu

ABSTRACT

Dynamic inference is an emerging technique that reduces the computational cost of deep neural network under resource-constrained scenarios, such as inference on mobile devices. One way to achieve dynamic inference is to leverage multi-branch neural networks that apply different computation on input data by following different branches. Conventional research on multi-branch neural networks mainly targeted at improving the accuracy of each branch, and use *manually* designed rules to decide which input follows which branch of the network. Furthermore, these networks often provide a small number of exits, limiting their ability to adapt to external changes. In this paper, we investigate the problem of designing a flexible multi-branch network and early-exiting policies that can adapt to the resource consumption to individual inference request without impacting the inference accuracy. We propose a lightweight branch structure that also provides fine-grained flexibility for early-exiting and leverage Markov decision process (MDP) to automatically learn the early-exiting policies. Our proposed model, EPNet, was effective in reducing inference cost without impacting accuracy by choosing the most suitable branch exit. We also observe that EPNet achieved 3% higher accuracy with an inference budget, compared to state-of-the-art approaches.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Neural networks**;

KEYWORDS

Dynamic neural network; efficient neural network; early exiting

ACM Reference Format:
Xin Dai, Xiangnan Kong, and Tian Guo. 2020. EPNet: Learning to Exit with Flexible Multi-Branch Network. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411973

1 INTRODUCTION

Deep convolutional neural networks (CNNs) have achieved good accuracy on computer vision tasks such as image classification [8, 11, 14]. However, the accuracy improvement is often accompanied with higher demand for computational and memory cost. To utilize these resource-intensive models in deployment scenarios such as mobile devices, where desired resources are constrained, prior work proposed techniques such as model pruning [5, 15], low-rank factorization [18, 19], knowledge distillation [3, 9], efficient CNN [12, 24], and hard attention model [1, 17].

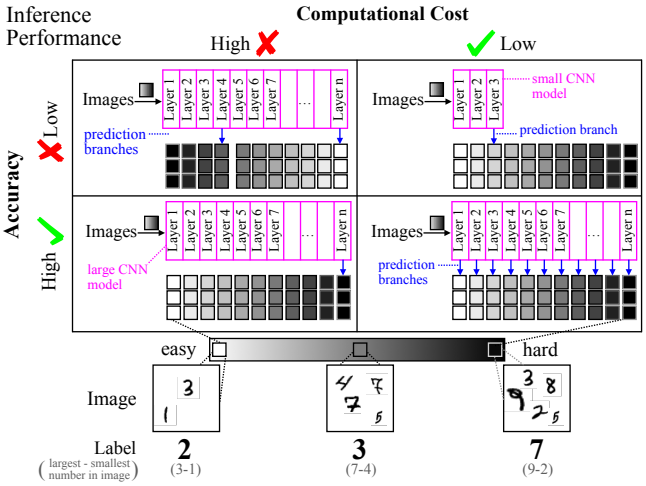


Figure 1: The problem of dynamic inference for image classification tasks. Our goal is to design a CNN that can adapt its execution to the inference request difficulty, to achieve high inference accuracy and low computational cost.

Dynamic inference is an emerging technique that aims to reduce the resource consumption, e.g., computational cost, of deep neural network during inference. A prominent approach of dynamic inference centers around the use of multi-branch networks [10, 16, 20], i.e., networks that consists of more than one output layer, for handling the natural difficulty variations exhibited in real-world samples. Ideally, the multi-branch network spends *just enough computation* for each sample, instead of applying the same amount of computation, as illustrated in Figure 1. For example, easier samples can use earlier prediction branches while the harder ones go through the normal forward propagation. We refer to the scenarios of using branch classifiers as *early-exiting*. One of the key challenges in achieving efficient dynamic inference is being able to adapt the resource consumption to individual inference request without impacting the inference accuracy. Existing work on *dynamic inference with multi-branch network* [10, 16, 20] used handcrafted policies for deciding the exiting branch per inference, as shown in Figure 2. As these policies require domain experts in setting the threshold, the performance is subject to external factors such as resource fluctuation or sample difficulty. Further, prior networks only provide a

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411973

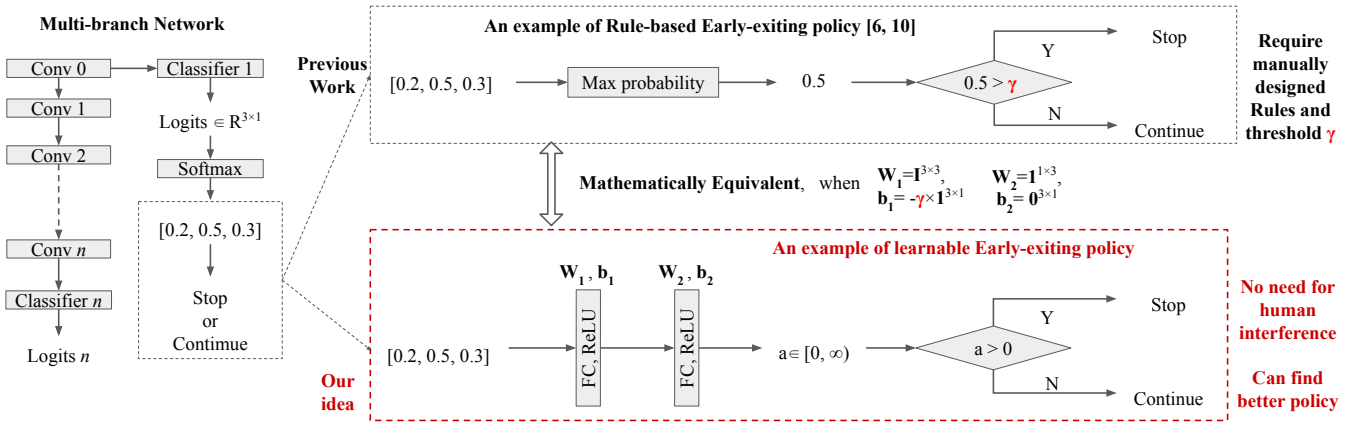


Figure 2: Early-exiting policies for multi-branch networks. A learning-based policy has two key advantages over the rule-based policies: *(i)* does not require human interference; *(ii)* can lead to better policies.

few adapting options, with a smaller number of branch exits. This limits the network's ability to adapt to smaller changes, i.e., less flexible.

In this work, we investigate the problem of designing a flexible multi-branch network and early-exiting policies that can be learned in conjunction from the training dataset. Designing a flexible multi-branch network requires attending to the tension between the number of branches and the additional parameters and computational cost. In particular, we need an efficient network structure that sufficiently represents the search space of the early-exiting policies, while being mindful about the additional cost associated with the early-exiting controllers. Additionally, when designing the controllers, we need to explicitly consider the trade-off between the classification accuracy and resource efficiency.

In designing our multi-branch network EPNet, we make the following main contributions.

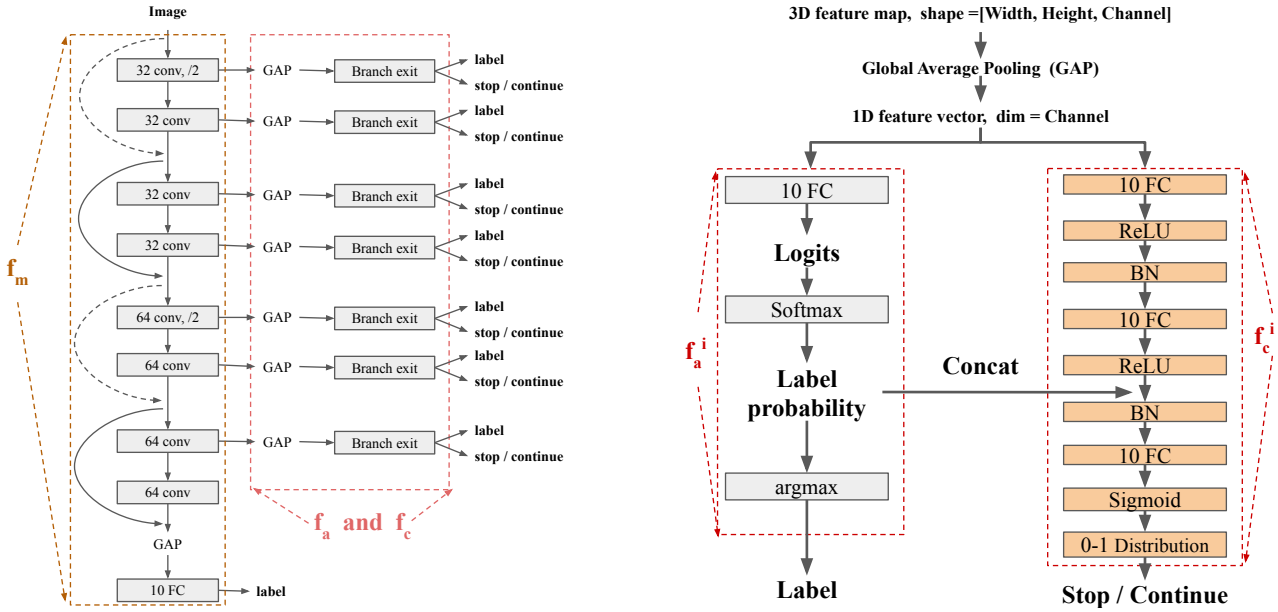
- We formulate the trade-off between the classification accuracy and efficiency as the computational-sensitive classification problem. Specifically, we use a *benefit* score which is a weighted sum of accuracy and efficiency to guide the learning of early-exiting controllers. We provide a tunable hyperparameter called *cost sensitivity* which can be set to represent the available resources.
- We design a novel multi-branch network structure that provides fine-grained flexibility for early-exiting with negligible resource increase. Concretely, we use a lightweight branch design with low computational and parameters cost and therefore are able to attach the exit after every convolutional layer.
- We formulate the early-exiting problem as a Markov decision process (MDP) and use a policy gradient without sampling method to efficiently train good performant policies.
- We compare the performance of our EPNet to state-of-the-art rule-based early-exiting policies using three different datasets. The evaluation results show that our multi-branch network, guided by the early-exiting controller, outperformed baseline policies and was able to adapt to changing resource requirement.

2 DYNAMIC INFERENCE

Dynamic inference, adapting the resource required for performing an inference request, provides opportunities to efficiently leverage deep neural networks in real-world scenarios. In this work, we focus on one such adaption that is based on the sample difficulty. In Figure 1, we illustrate our targeted problem space and four different inference scenarios. Our goal is to reconcile the conflicting goals of inference accuracy and resource consumption with a flexible multi-branch network design and automatic early-exiting policies.

The typical structure of multi-branch neural network consists of two parts: main branch network f_m and additional branches f_a. The main branch f_m can be a normal convolutional neural network, *e.g.*, AlexNet [14], ResNet [8] or DenseNet [11]. Given a convolutional layer $conv_i$ in f_m, the additional branches f_a^i is a neural network taking the activation of $conv_i$ as input and ending with a classifier layer. The form of f_a could be CNN [20] or even more complex structure [10].

Early-exiting policies, as illustrated in Figure 2, define which part of the multi-branch neural network will be used for the inference computation. It means that the multi-branch neural network may *automatically* stop the forward propagation computation in the main branch f_a at a convolutional layer $conv_i$, and return the prediction of the branch f_a^i. In the previous studies [10, 16, 20], the early-exiting policy is rule-based, e.g., if the uncertainty of the logit yield by f_a^i is below than a given threshold, the model directly early exits at the f_a^i [20]. In short, using multi-branch networks with early-exiting policies is a form of dynamic inference that can adapt to inference difficulty, therefore lead to lower computational cost. There are other types of dynamic inference defined based



(a) Overall network architecture **(b) The i-th branch exit with a classifier and a controller.**

Figure 3: An example of our proposed EPNet, a multi-branch network with learnable early-exiting policies. The overall structure consists of three parts: main branch f_m, branch classifiers f_a, and early-exiting controllers f_c.

on conventional CNNs [2, 22] which we will further discuss in Section 6.

3 PROBLEM FORMULATION

In this work, we target the dynamic inference problem of image classification. Given a set of samples $\mathcal{D}$=$\{\mathbf{I}_i\}$, where $\mathbf{I}_i \in \mathbb{R}^{x \times y \times c}$ denotes a image, and a user-defined resource sensitivity value β, our goal is to design a multi-branch model $\mathbf{M}$ that balances the classification accuracy and resource cost trade-offs.

To learn $\mathbf{M}_\theta$ that is parameterized by θ, we design an optimization metric called *benefit score* B as:

$$B(\mathcal{D}, \mathbf{M}_\theta, \beta) = Acc(\mathcal{D}, \mathbf{M}_\theta) - \beta \times C(\mathcal{D}, \mathbf{M}_\theta), \quad (1)$$

where $Acc(\mathcal{D}, \mathbf{M}_\theta)$ is the average accuracy of $\mathbf{M}_\theta$ on $\mathcal{D}$, and $C(\mathcal{D}, \mathbf{M}_\theta)$ is the average computational cost of $\mathbf{M}_\theta$ on $\mathcal{D}$. The parameters θ is obtained by maximizing the *benefit score* S, and will be used to decide how to perform the classification task. In our formulation, if the resource is ample, we can set β to be 0 which turns to a traditional classification problem. We can set β to a larger value when the resource is more constrained.

4 MULTI-BRANCH MODELS DESIGN

We first describe the overall architecture and design of our proposed multi-branch models EPNet. We then detail our formulation of the early-exiting problem in Section 4.2, followed by how to effectively train the early-exiting controllers in Section 4.2.

4.1 Model Structure Overview

Our proposed EPNet consists of three components: the main branch network f_m, branch classifiers f_a, and early-exiting controllers f_c. Figure 3 illustrates an example structure for image classification on *CIFAR-10*.

4.1.1 Main Branch Network. The **main branch network** f_m (the leftmost component in Figure 3(a)) takes the image as the input and can produce classification result independently with the final classifier layer. To enable as many exits as possible, we attach additional branch classifiers f_a (described below) at every convolutional layer except the last one. Larger number of branch classifiers provides the early-exiting controllers f_c more flexibility to adapt to dynamic inference environments such as varying sample difficulties and fluctuating system resources.

Additionally, the ideal property of f_m is the monotonically increasing accuracy with the number of the layers, i.e., the branch classifiers attached to the latter layers should have higher accuracy than earlier layers. In this work, we chose the ResNet [8] as the main branch network architecture. Other potential implementations of f_m include networks with short-cut connections such as DenseNet [11].

In short, the main branch network should be designed around two key principles: *(i)* enabling as many exits as possible; *(ii)* maintaining monotonically accuracy increase with exits.

4.1.2 Branch Classifiers. As shown in Figure 3(b), we need a **branch classifier** f_a^i for each exit i. In our current implementation, f_a^i is attached to the i-th convolutional layer of f_m. For a main branch network with a total of N convolutional layers, we need a set of branch classifiers f_a represented as $\{f_a^i, i \in \{1, 2, .., N-1\}\}$.

One of the key design challenges for branch classifiers is to balance its resource requirement and classification accuracy. Considering the following example. In order for an exit i to be a valid exit, the total computational cost of exiting through f_a^i should not

exceed that of exiting through f_a^{i+1}. This indicates an upper bound of computational budget, e.g., the difference between the two consecutive convolutional layers, when designing the branch classifiers. This computational budget has to be shared with the early-exiting controllers, further restricting our design space.

Similarly, the memory consumption of branch classifiers, i.e., number of parameter, is also a big problem. In classic CNN structure, the 3D feature map outputed by last convolutional layer is flattened and fed to a fully connected layer, where the most of the parameters belong to. This suggests that simply attaching a classifier layer to every convolutional layer may lead the memory consumption to increase by multiple times.

Instead, we design the f_a^i with the structure of GAP-FC-SoftMax. Here the GAP is a global average pooling layer and FC is a fully connected layer. We chose to use the GAP layer because it significantly reduce the resource requirement of the branch classifiers.The input of f_a^i is the 3D activated feature map generated by the i-th convolutional layer.

In short, the branch classifiers should: *(i)* comply to the resource consumption pattern of the main branch network layers; and *(ii)* without impacting the accuracy.

4.1.3 Early-exiting Controllers. Lastly, our EPNet requires a set of **early-exiting controllers** $f_c = \{f_c^i, i \in \{1, 2, .., N-1\}\}$ that regulates the usage of each exit i.

We design f_c^i as a two-part network, i.e,. f_{in}^i and f_{cat}^i, to preserve the information of features outputted by both the GAP layer and the logits outputted by f_a^i. This allows our controllers to perform at least as well as previously proposed rule-based policy [10, 16, 20]. Both of the f_{in} and f_{cat} are in the form of stacked blocks FC-BN-ReLU, except for the last activation of f_{cat} which should be Sigmoid function. Here BN is a batch normalization layer.

Specifically, f_{in}^i takes the 3D activated feature map generated by the i-th convolutional layer as input and outputs a 1D vector v. This 1D vector v and the logits outputted by f_a^i are concatenated and used as the input to f_{cat}^i who then output a scalar signal $p \in [0, 1]$. From the Bernoulli distribution paramiterized by p, we sample a stopping signal $s \in \{0, 1\}$. If $s = 0$, the forward propagation in main branch f_m will continue until another controller f_c^j at j-th convolutional layer outputs $s = 1$, or reaches the final classifier in f_m. If $s = 1$, the forward propagation is immediately stopped and the model output the label predicted by the current branch classifier f_c^i.

4.2 Learning the Early-exiting Policy

We formulate the early-exiting problem as a Markov decision process (MDP) problem $M = (S, A, T, R)$, where the environment is $E = (f_m, f_a, D)$. We describe the *state set S*, *Action set A*, *Transformation table T* and *Reward R* in detail below. The early-exiting policy π can be learned through maximizing the expected reward $E_\pi(R)$, once f_m and f_a are trained.

States set S. We define a state s_i as (m_i, y_i) where m_i is the outputted vector at the GAP layer after the i-th convolutional layer of f_m, and y_i is the logits outputed by f_a^i. Additionally, S contains a distinguish state s_{ab} called absorbing state. The MDP stops when any states transition to s_{ab}. In our case, s_{ab} represents the state when the controller decides to stop and exit from exit i. Lastly, we define the start sate $s_0 = I$ where I denotes an image from D.

Action set A. The MDP only has two actions: "stop at current exit" or "continue to forward propagation". Here we denote it as $A = \{0, 1\}$, where 0 is "stop" and 1 is "continue". Ones the agent takes action $a = 1$, the state transfer to s_{ab}. So given a image x, the trajectory set $\mathcal{T}$ of agent can be denotes as $\mathcal{T} = \{(x, 0^n, 1)|n \in \{0, 1, ..., N-1\}\}$, where 0^n means a succession of 0 of length n, and N is the total number of branch exits.

Transformation table T. $T = \{P(s, a, s')|s, s' \in S, a \in A\}$, where $P(s, a, s')$ is the probability that state s transfer to s' by taking action a. In this study, the T is deterministic so that all $P(s, a, s') = 1$ if $\pi(s, a) = s'$, otherwise $P(s, a, s') = 0$.

Reward R. Given a Image x from D and a cost sensitivity β. If the agent stop at the i-th exit, the trajectory is $\tau = (x, 0^{i-1}, 1)$. The reward $R(\tau)$ is the $Benefit(\{x\}, f_i, \beta)$ defined in Eq 1. Here f_i is the sub-network from the input layer of the main network f_m to the output layer of i-th branch classifier f_a^i.

4.3 Training Consideration of the Controllers

The main branch f_m and additional branch classifiers f_a can be trained by simply summing their cross entropy loss together [20]. Here We mainly describe two approaches to train the controllers f_c. We compare their ability to find early-exiting policy in Section 5.5.

The **first option** is to leverage REINFORCE algorithm [23] to train the early-exiting controllers as following.

$$\nabla_\theta E_\pi(R) \approx \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log \left(\pi \left(a_i^j | s_i^j; \theta \right) \right) R^j \tag{2}$$

$$\pi\left(a_i|s_i;\theta\right) = \begin{cases} f_c^i(s_i) & a_i = 1, s_i \neq s_{ab} \\ 1 - f_c^i(s_i) & a_i = 0, s_i \neq s_{ab} \\ 1 & s_i = s_{ab} \end{cases} \tag{3}$$

Here m is the number of episode, and n denotes the length of a trajectory, *i.e.*, the number of exits. s, a, R are the states, actions, rewards defined in the previous section.

But the classic REINFORCE rule is based on sampling and Markov Chain Monte Carlo approach (MCMC), which could be inefficient in our task. For example, if the dynamic model has 9 additional branches, the trajectory $\tau = (x, 0^9, 1)$ may have very low chance to be sampled. This is because it requires all the controllers to output *continue*. The low sampling efficiency can cause well known drawback of REINFORCE, the high variance of policy gradient.

The **second option**, which we used for training the controllers in this work, is to directly compute the exact gradient of $E_{\pi_\theta}(R)$ as following.

$$\nabla_\theta E_\pi(R) = \sum_{j=1}^{n} \nabla_\theta \prod_{i=1}^{n} \left(\pi \left(a_i^j | s_i^j; \theta \right) \right) R^j \tag{4}$$

The gradient computation is feasible because of two important properties of our MDP. *First,* The environment $E = (fm, fa, D)$ is a given and the only randomness comes from the policy π itself. *Second,* given an image and a multi-branch network of N branches, the size of trajectories set is $\mathcal{T} = \{(x, 0^i, 1)|i \in \{0, 1, ..., N-1\}\}$

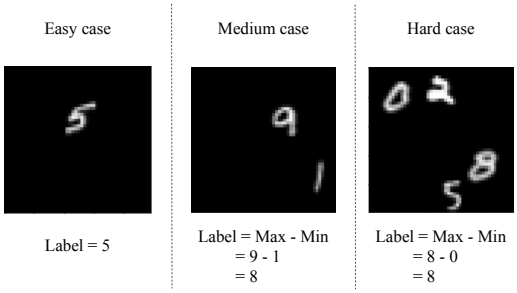


Figure 4: Example of samples in our Max-Min MNIST dataset. This dataset contains samples of three difficulty levels. The ratios for easy, medium, and hard levels are 2:1:1.

with size N. In our current design N is bounded by the number of convolutional layers which varies from tens to a couple hundreds given current popular CNNs. As such, N is small enough that we don't need sampling.

5 EXPERIMENT

We evaluate our proposed EPNet with three datasets and compare to two types of baselines. We summarize and highlight our key results below.

• **Accuracy and resource comparisons to baselines.** In Section 5.3, we show that EPNet outperformed all baselines including ResNet, and both rule-based and manually-tuned early-exiting policies applied to multi-branch models [10, 16, 20].

• **Adaptivity of EPNet.** In Section 5.4, we demonstrate EPNet's ability to effectively choose the appropriate branch exit based on both the classification difficulty of samples and the computational cost sensitivity of the resource-constrained platform.

5.1 Data Sets

We used the following three widely used datasets for image classification task to evaluate the performance of our EPNet.

• **Max-Min MNIST dataset.** We created this dataset of three difficulty levels, based on the original MNIST, to evaluate the effectiveness of early-exiting policy. Figure 4 illustrates corresponding image examples. We constructed the *easy* samples by embedding the original MNIST digits into a 50×50 black background and reusing the original labels. The *medium*-level samples had two digits embedded in the 50×50 black background, with the labels being the absolute difference between the two digits. Finally, the *hard*-level samples had twice as many digits as the *medium*-level ones and were assigned the labels in the same way. For each sample, the locations of digits were generated from the 2D uniform distribution. We used the ratios of *2:1:1* for *easy*, *medium* and *hard* levels for both training and test sets, respectively. The total numbers of training and test images were 60000 and 10000 respectively, the same as the original MNIST dataset.

• **Multi-scale Fashion MNIST.** We created this dataset based on the original Fashion MNIST. Figure 5 illustrates corresponding image examples, which could be categorized to four types. We constructed the Type 1 by embedding the original Fashion MNIST objects into a 50×50 black background and reusing the original labels, the objects are scaled to the 0.6 times of its original size.

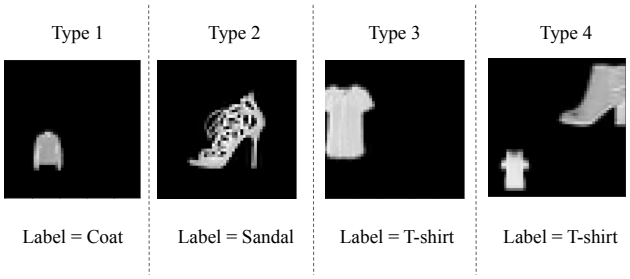


Figure 5: Example of samples in Multi-scale Fashion MNIST dataset. This dataset contains four different types of samples, with equal proportion.

For the Type 2 and 3, the objects are scaled to the 1.5 times of its original size. The Type 4 contains two objects, one is large, another is small, and its label is decided by the small objects. In the Type 1, 3, 4, the locations of object were generated from the 2D uniform distribution. For Type 2, the object is fixed at the center. We used the ratios of *1:1:1:1* for each type, respectively. The total numbers of training and test images were 60000 and 10000 respectively, the same as the original Fashion MNIST dataset.

• **CIFAR-10.** We used the original *CIFAR-10* dataset which consists of 50k training and 10k test images of 32×32 pixels, respectively. For training we adopted the data augmentation technique used in the [8]. Specifically, each image was zero-padded with 4 pixels on each side, then randomly cropped to produce 32×32 resolution. Further, training images were flipped horizontally with 0.5 probability and the pixel values were normalized by subtracting channel means and dividing by channel standard deviations.

5.2 Baseline Methods

We compared to two types of baselines: *(i)* a state-of-the-art image classification model; and *(ii)* early-exiting policies.

• **ResNet [8].** We not only compared with ResNet, but also use ResNet as the main branch network f_m of proposed EPNet. We made the following changes to the original ResNet structure to account for the image size difference between our chosen datasets and the ImageNet dataset of which ResNet was designed for. These changes are: *(i)* we removed the first conv layer and the first pooling layer; *(ii)* the first conv layer in the first residual block was performed by stride of 2.

• **BranchyNet [20].** We chose a representative entropy-based early-exiting policy as described in BranchyNet [20]. At a high level, BranchyNet works by comparing the entropy of logits of an exiting branch to a predefined threshold and halting the forward propagation in main branch if the logits entropy is lower. We used the recommended entropy thresholds of 0.2 and 0.3 for the baselines and denoted them as *BranchyNet-0.2* and *BranchyNet-0.3*, respectively. We also compared the BranchyNet with dynamic thresholds, denoted as *BranchyNet-oracle*, which was constructed as following: *(i)* the thresholds are manually tuned; *(ii)* by using the results of our EPNet as a guidance for searching the thresholds.

• **Softmax-gated policy.** This baseline leverages the maximum value of the softmax probability and compares the probability to a given threshold for early-exiting [10, 16]. We denoted the baseline as *Softmax-gate-*γ where γ is the threshold. Given the logits of a branch classifier, if the maximum value of the softmax probability

is larger than $1-\gamma$, the model halt the forward propagation. On each dataset, we pick two thresholds. By the smaller γ the network can achieve the high accuracy close to our EPNet. By the larger γ the network can maintain low computational cost close to EPNet. Similar with BranchyNet, we also compare with Softmax-gate with dynamic thresholds, denoted as *Softmax-gate-oracle*.

For fair comparisons, our EPNet uses the same structure as its ResNet for its main branch network f_m. Further, EPNet shares the same structure and parameters of f_m and f_a with the BranchyNet and Softmax-gated policy.

5.3 End-to-end Evaluation

5.3.1 Evaluation Methodology. We evaluated the effectiveness of EPNet on all three datasets with following two metrics.

The first metric we chose is the *benefit score* that was defined in the Equation (1). This metric allows us to compare our EPNet to baselines in a unified way. In each task, we firstly set the cost sensitivity to $\beta \times 0.01$, where β^{-1} is the order of magnitude of the average comutational cost (FLOPS) of a single convolutional layer in the EPNet used. Then we increase the cost sensitivity each time by adding $\beta \times 0.01$ to the previous cost sensitivity. So we can observe how the methods' performances change against the decreasing available resources. Under each setting of cost sensitivity, we retrain the controllers and keep the rest part of EPNet fixed.

We also used a metric, referred to as *budget-constrained accuracy*, for understanding the effectiveness of early-exiting [10, 16]. To calculate the budget-constrained accuracy, we first define a computational budget and then use it as a barrier for determining the accuracy. For example, in the case of CIFAR-10, we used our EPNet's total computational cost (FLOPS) over the test dataset as the computational budget, and evaluated all baseline models. For baseline models that did not finish all test images within the budget, we assigned labels in an uniformly random way to the remaining test images. We use acc^b to denote the resulting budget-constrained accuracy.

5.3.2 Performances on Max-Min MNIST dataset. We first describe the network structure and parameter settings we used in EPNet for training on the Max-Min MNIST dataset, followed by the performance comparison to its respective baselines.

Network structure setting. For the main branch network f_m, we used a ResNet with 12 convolutional layers. The first four layers each has 32 filters, followed by another four layers with 64 filters. The last two convolutional layers are of 128 filters. We downsampled by using a stride of 2 for convolution when the number of filters changed between layers. To construct the early exits, we used a single-layer classifier f_a^i that takes the input of the i-th convolutional layer of f_m. This resulted in a total of 12 potential exits. On i-th exit, The classifier f_a^i is a single fully-connected layer. The controller f_c^i consists of two fully-connected networks f_{in} and f_{cat}, where the f_{in} has 10, 10, 10 units in each layer, and the f_{cat} has 10, 10, 1 units in each layer.

Parameter setting. We adopted the Kaiming initialization [7] and BN [13] without dropout when training the main and branch classifiers f_m and f_a, respectively. We used a mini-batch size of 64 and momentum of 0.9. We set the initial learning rate to be 0.1. We trained the classifiers for a total of 60 epochs. Once the classifiers were trained, we fixed the classifiers and train the controllers f_c. The mini batch size is 64, and the initial learning rate is 0.01. We trained the controllers for a total of 60 epochs as well.

Result and discussion. Figure 6(a) compares the benefit score achieved by different baselines and our EPNet. We make the following three key observations. *First,* the EPNet greatly outperforms all baselines, and the gap of performance grows as the cost sensitivity grows. At beginning the cost sensitivity is small (2×10^{-7}), the EPNet outperforms the best baseline Softmax-gate-oracle by 3.11. When cost sensitivity reaches 1×10^{-6}, the gap between the scores of EPNet and Softmax-gate-oracle increases to 10.17. *Second,* the ability to learn the early-exiting policy from dataset is the reason of the superiority of EPNet. The performances of oracle baselines with dynamic thresholds and the branchyNet-0.2 / 0.3 are very close, indicating the thresholds 0.2 and 0.3 recommended in [20] are suitable for this dataset, while tuning the thresholds can't bring obvious benefit. In contrast, the gaps between the EPNet and the oracle baselines are much larger than the differences among the early-exiting baselines. This results may indicate the EPNet learns much better representation of the confidence of the classification than the rule-based methods. *Third,* The ResNet's benefit score is worst because it only focuses on accuracy while has the highest computational cost.

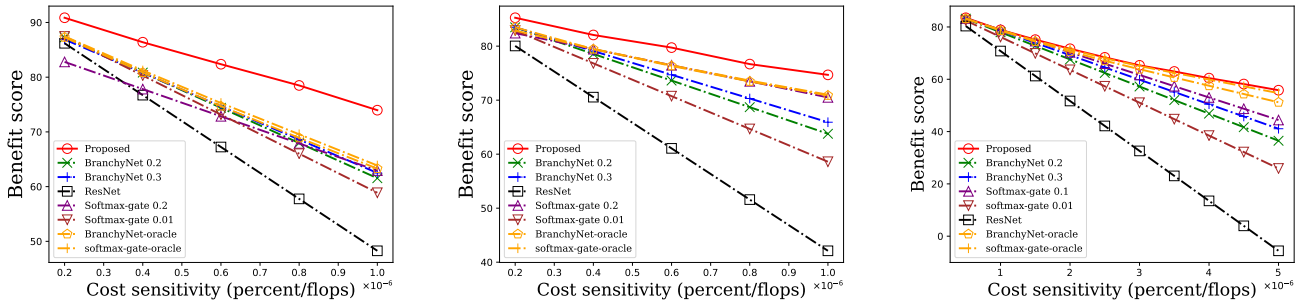
Table 1 shows the comparison on budgeted batch classification. The accuracy of EPNet is 95.51%, which is higher than the best baseline Softmax-gate-0.2 by 7.35%. The BranchyNet-0.2 and 0.3 can't finish within the budget. We have to rise the threshold to 0.95, then the BranchyNet's computational cost meets the limitation of budget. But its accuracy is only 84.13% which is 11% lower than EPNet. The ResNet with same structure of the EPNet's main branch network, can only achieve 46.86% under the limited budget, which is 48.65% lower than the EPNet. Compared with ResNet without budget limitation, the EPNet's accuracy is only 0.18% lower than it, while saves about 50% of computation. Even when the budget limitation is removed for the baselines except for ResNet, their accuracies are all lower than EPNet by at least 1%.

5.3.3 Performances on Multi-scale Fashion MNIST. Next we study the effectiveness of the EPNet on Multi-scale Fashion MNIST dataset.

Network structure setting. We used the same network structure as described in Section 5.3.2 for the Max-Min MNIST dataset.

Parameter setting. We adopted the Kaiming initialization [7] and BN [13] without dropout when training the classifiers (i.e,. f_m and f_a) and the controllers f_c, respectively. For the classifiers, we used a mini-batch size of 128 and momentum of 0.9. We set the initial learning rate to be 0.1 and divide the learning rate by 10 every 100 epochs. We trained the classifiers for a total of 300 epochs. Once the classifiers were trained, we fixed the classifiers and trained the controllers f_c. The mini batch size is 128, and the initial learning rate is 0.01. After 50 epochs the learning rate was reduced to 0.001, then we trained the controllers for another 50 epochs.

Result and discussion. Figure 6(b) shows the performances of each model according to the benefit score. Similar with Max-Min MNIST dataset, we make the following two key observations. *First,*



(a) Max-Min MNIST **(b) Multi-scale Fashion MNIST** **(c) CIFAR-10**

Figure 6: The benefit score comparison on all three datasets. We observe that our proposed EPNet outperformed two types of baselines.

Table 1: Results of budgeted classification. The budget of each task is the total computational cost of proposed method on the whole dataset. The results are reported as test accuracy constrained by budget (acc^b) (rank), test accuracy (acc*), and whether the classification finished before the budget ran out.

Dataset	Model	acc^b(%)	acc(%)	Completion
Max-Min MNIST	**Proposed**	95.51 (1)	95.51	✓
	BrachyNet-0.2	67.65 (5)	93.71	✗
	BrachyNet-0.3	70.82 (4)	93.03	✗
	BrachyNet-0.95	84.13 (3)	84.13	✓
	softmax-gated-0.2	87.76 (2)	87.76	✓
	softmax-gated-0.01	61.53 (6)	94.59	✗
	ResNet	46.86 (7)	95.69	✗
Multi-Scale Fashion MNIST	**Proposed**	88.68 (1)	88.68	✓
	BrachyNet-0.2	61.09 (5)	88.41	✗
	BrachyNet-0.3	68.11 (4)	87.88	✗
	BrachyNet-0.6	85.68 (2)	85.68	✓
	softmax-gated-0.2	85.38 (3)	85.38	✓
	softmax-gated-0.01	49.88 (6)	88.69	✗
	ResNet	32.14 (7)	89.92	✗
CIFAR-10	**Proposed**	88.61 (1)	88.61	✓
	BrachyNet-0.2	86.54 (4)	88.35	✗
	BrachyNet-0.3	87.83 (2)	87.83	✓
	softmax-gated-0.1	87.53 (3)	87.53	✓
	softmax-gated-0.01	71.72 (5)	88.76	✗
	ResNet	47.82 (6)	89.89	✗

the EPNet always outperforms all the baselines and the gaps between EPNet and the baselines are very clear. When the cost sensitivity is 2×10^{-7}, the benefit score of the EPNet is 85.25, and the score of best baseline BranchyNet-oracle is 83.48. When the cost sensitivity reaches 1×10^{-6}, the benefit score of EPNet is 74.68, and the best baseline, Softmax-gate-oracle is 71.01. *Second,* the performances of oracle baselines are almost the same with the Softmax-gate-0.2, indicating 0.2 is already a good threshold. Therefore manually tuning the threshold did not make a big difference. Similar to the discussion of the previous dataset, this observation reflects that the performance of the rule-based polices are limited by their confidence/uncertainty measure.

As show in the Table 1, on the budgeted batched classification, the accuracy of EPNet is 88.68%, which is higher than all the baselines by at least 3%. The ResNet with same structure of the EPNet's main branch network, can only achieve 32.14% under the limited budget, which is 56.54% lower than the EPNet. Both the branchyNet-0.2 and branchyNet-0.3 can't finish the whole test set

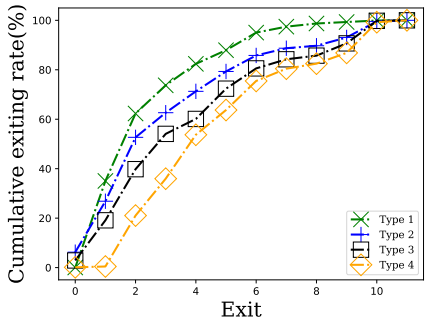


Figure 7: Cumulative exiting rate on the Multi-scale Fashion MNIST. The cost sensitivity was 2×10^{-7}.

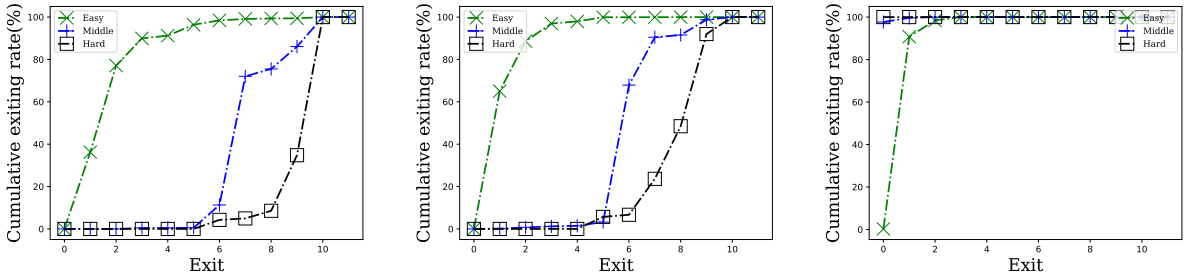
before the budget runs out. We have to increase the threshold to 0.6 to fit the budget, and the accuracy of branchyNet-0.2 is 85.68%. Even the budget limitation is removed, branchyNets' accuracies are still lower than EPNet, which may indicate their measurements of confidence, *i.e.*, the entropy of logits, may be not feasible in this task.

5.3.4 Performances on CIFAR-10. Lastly we study the effectiveness of the EPNet on CIFAR-10.

Network structure setting. For the main branch network f_m, we used a ResNet with 10 convolutional layers. The structure is similar with the previous task but only has two convolutional layers of 128 filters. For the controller of i-th branch, the f_{in} has 100, 10, 10 units in each layer, and the f_{cat} has 100, 100, 50, 1 units in each layer.

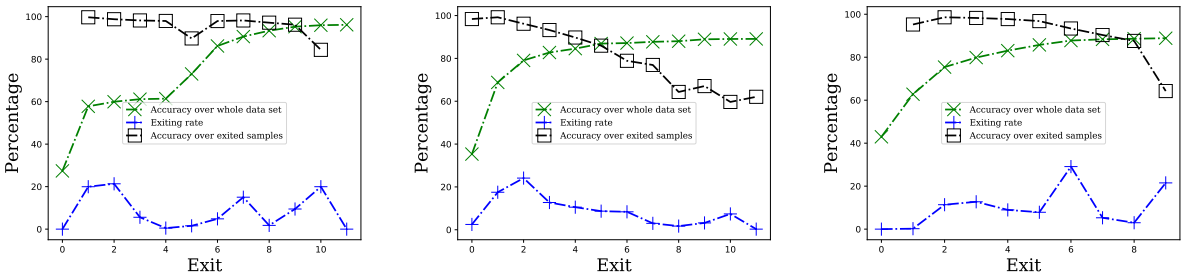
Parameter setting. The most of the parameters are same with the Multi-scale Fashion MNIST. Additionally we used the weight decay of 0.0001 to train the classifiers. Also, for training of controllers, we set the batch size = 64.

Result and discussion: Figure 6(c) shows the performance of each model according to the benefit score. The gaps between EPNet and baselines are not as large as on the other datasets. The reason may be the difficulties of the samples are not as obvious as the other two datasets. So we test the models under more settings of cost sensitivity. The key observations are: *First,* even though the sores of all models are very close to each other at beginning, the gap between the scores of EPNet and the baselines with fixed



(a) The cost sensitivity is 2×10^{-7} **(b)** The cost sensitivity is 1×10^{-6} **(c)** The cost sensitivity is 2×10^{-6}

Figure 8: Cumulative exiting rate of samples with varying difficulties on the Max-Min MNIST dataset.



(a) Max-Min MNIST **(b) Multi-scale Fashion MNIST** **(c) CIFAR-10**

Figure 9: The distribution of accuracy and exiting rate. The cost sensitivities for each dataset are 5×10^{-7}, 2×10^{-7}, 2×10^{-7}.

thresholds increases quickly. When cost sensitivity reaches 5×10^{-6}, the score gap is up to 10.17. *Second,* even though the oracle baselines are granted "unfaire" advantages, *i.e.*, their thresholds are tuned by hand and the searching is hinted by the results of EPNet, our EPNet still slightly outperforms them. When the cost sensitivity is small (5×10^{-7}), the EPNet outperforms the best baseline Softmax-gate-oracle by 0.2. Then their gap of benefit score grows to 1.2 as the the cost sensitivity grows to 5×10^{-6}.

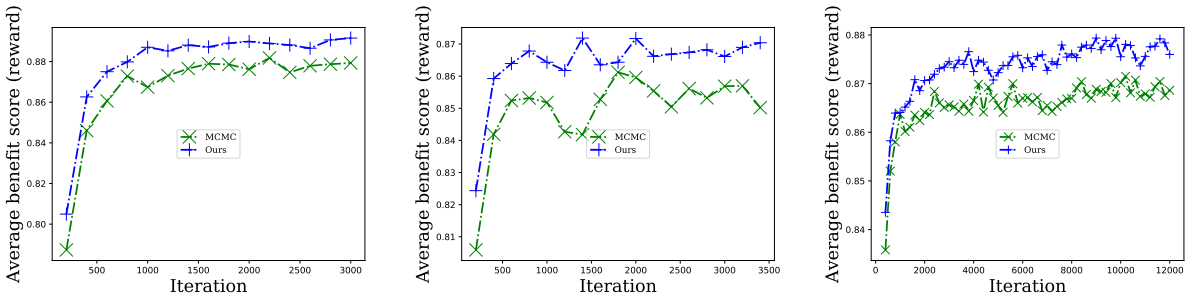
Table 1 compares the budget-constrained accuracy and contains the model test accuracy for reference. We observe that among all tested baseline models, only two were able to complete within the computational budget. Even so, *BranchyNet-0.3* and *Softmax-gate-0.1* had 0.78% and 1.08% lower accuracies than our EPNet, respectively. Compared to the *ResNet-11* that shared the same main branch structure, our EPNet achieved 40% higher budget-constrained accuracy. Our results demonstrate the effectiveness of our EPNet in operating with stringent resource.

5.4 Case Study: EPNet's Adaptivity to Classification Difficulty

The desired controller should be able to identify the samples which are easy to be classified at early stage. This property is well supported by the experiment on Max-Min MNIST and Multi-scale Fashion MNIST, which consist of samples of different difficulty levels in term of classification.

Multi-scale Fashion MNIST. In Figure 7 we show the cumulative exiting rate over all exits for the Multi-scale Fashion MNIST. The curves that are closer to top-left corner are more tend to exit at earlier branch exits. About 90% of Type 1 samples stop before exit-5. The reason may be the objects in Type 1 are very small, so the shallower layers are sufficient to model them. The 80%-stop point of Type 2 and Type 3 samples are exit-6 and exit-8 respectively. It makes sense because compared with Type 1, the Type 3 samples have larger objects that require down-sampling to capture the objects. Compared with Type 2, the locations of object of Type 3 is random, so it is also more difficult than the Type 2. The Type 4 is at most bottom-right, because the samples are harder than other types due to they have two objects and need to tell which one is larger.

Max-Min MNIST. The similar finding is more obvious on the Max-Min MNIST. Figure 8(a) shows when the cost sensitivity is 5×10^{-7}, the easy samples mainly stop at the second, third and fourth exit. The medium samples mainly stop at 8-th exit. The hard samples mainly stop at 10-th and 11-th exit. This result indicates the controller learns to predict the easy samples at shallow stage and leave the hard samples to the deep stage, thus the computational cost is saved. Figure 8(a) shows when the cost sensitivity increases to 2.5×10^{-6}, the controller tends to left-shift the exiting distribution of all samples, because the model is more sensitive to the computational cost. As show in the Figure 8(c), when the cost sensitivity reaches 5×10^{-6}, something interesting happened. The controller chose to output the hard and medium samples at the first exit, even before the easy cases! The reason is to correctly classify the hard samples, we have to use the deep layers, but the computation cost



(a) Max-Min MNIST (b) Multi-scale Fashion MNIST (c) CIFAR-10

Figure 10: Average benefit score during the training phase. The cost sensitivities for each dataset are 5×10^{-7}, 2×10^{-7}, 2×10^{-7}.

Table 2: Performance comparison of different controller layer configurations.

Configuration	Neurons in f_{in}	Neurons in f_{cat}	Benefit Score	Accuracy (%)	Total network Computation (FLOPS)	Controller Computation (FLOPS)
1	[10, 10, 10]	[10, 10, 1]	82.40	88.42	1.203×10^7	0.107×10^4
2	[50, 10, 10]	[50, 50, 1]	83.17	88.45	1.054×10^7	0.724×10^4
3	[100, 10, 10]	[100, 100, 50, 1]	83.51	88.61	1.019×10^7	2.419×10^4
4	None	[100, 100, 50, 1]	82.99	88.37	1.074×10^7	1.700×10^4

is much larger than the benefit of correct classification, as the cost sensitivity is large now. So in this case, outputting the hard cases at beginning is reasonable.

All datasets. For each exit, Figure 9 shows its accuracy on whole test set (denoted as acc_w), and on the exited samples (denoted as acc_e), as well as the exiting rate. On all the datasets, the acc_e is much higher than the acc_w at beginning. This reflects that the controller f_c^i can effectively find the samples which the classifier f_a^i can confidently predict, even without knowing the ground truth of sample difficulty. As the depth increases, the acc_w increases while the acc_e drops to below than acc_w. The reason is the easy cases have exited at previous branches, leaving the hard cases to the later branches.

5.5 Training discussion

Now we discuss the training step use different method. As we have statemented, the early exiting problem can be viewed as a reinforcement learning task, and could be solved by classic REINFORCE algorithm based on Markov chain Monte Carlo (MCMC) to estimate the gradients. Here we show the training process using our EPNet and MCMC in the Figure 10. As we can see, on three datasets, the training reward of both method increased very quickly at beginning, then slow down. The EPNet always keeps higher reward at each iteration step comparing with the MCMC REINFORCE algorithm. Especially on the Multi-scale Fashion dataset, the reward curve of MCMC REINFORCE algorithm has large fluctuation, even we already add the baseline to reduce the variance of policy gradient. On the contrary, our EPNet not only achieves higher reward, but also has much smaller fluctuation on the reward increasing curve. We think the reason may be our EPNet use the gradient of the exact expectation of reward instead of the estimation based on MCMC.

5.6 Parameter discussion

Lastly, we discuss the impact of the structure of controller on our EPNet performance. We train and test the EPNet on the CIFAR-10 under four configurations of controller, as show in the Table 2. Each configuration corresponds to different computational complexities. As we can see, increasing the depth and width of the controllers on each branch lead to higher computational cost of the controller (first three rows). However, the slightly higher computational cost on controller led to two orders of magnitude reduction in computational cost on whole network and better accuracy. This is because more complex controllers are better in learning the early-exiting policies.

Compared with Config 3, *i.e.*, the best configuration in Table 2, the Config 4 use the similar structure of f_{cat} but without f_{in}, which means the controllers in Config 4 only use the logits to decide to stop or continue, neglecting the original information in the feature vector after GAP layer. So its performance is worse than Config 3. Even though Config 4 uses a larger controller than Config 2, its performance on benefit score, classification accuracy and computational cost are worse than Config 2. This result support our idea that combining the pooled feature vector and logits can improve the controller's performance.

6 RELATED WORK

Rule-based Early-exiting. Teerapittayanon et al. proposed a multi-branch network named BranchyNet [20], which added several additional branches on CNNs including LeNet, AlexNet and ResNet. On each branch, the early exiting is controlled by the threshold of logit entropy. Huang et al. proposed a novel model called MSDNet, of which the structure is a stack of multiple DenseNet, for addressing the impacts of the multi-branch structure on the accuracy of branch classifier [10]. MSDNet is designed to provide coarse-level features to earlier branches and reduce the interference between branches. Li et al. further studied the problem of potential negative

impacts of gradients from multiple branches and proposed methods to collaboratively improve the training of branches [16]. Both [10] and [16] used the softmax probability for making early-exiting decisions. Our work propose a learning-based early-exiting approach for better adapting to inference environment.

Dynamic Inference on CNNs. Figurnov et al. proposed a spatial adaptive inference architecture called SACT [4] that can skip convolution within a residual block. Specifically, SACT calculates a halting score during every convolution in a residual block and decides whether to skip the next convolution in the same residual block. Veit et al. proposed a dynamic inference model called ConvNet-AIG [21] that aims to only execute the layers related to the category of input image. Concretely, ConvNet-AIG used a small network as a gated function to decide whether to execute a residual block or just jump over it through the shortcut link. Simiarly, Bengio et al. [2] proposed a method to dropout some units of a layer in neural network. Wang et al. proposed SkipNet [22] that leverages reinforcement learning to identify the suitable shallow networks per sample. Our work focuses on the co-design of a multi-branch network and its early-exiting policy for efficient dynamic inference.

Dynamic Inference on RNNs. Minh et al. proposed a recurrent attention model (RAM) [17] on visual learning tasks. RAM can learn to only attend to the important regions without scanning the entire image, similar to SACT [4]. On the task of time series classification, Hartvigsen et al. [6] proposed a novel model EARLIEST to jointly minimize the classification error and the execution time of the model. Both RAM and EARLIEST and the works mentioned above [2, 22] are trained by REINFORCE algorithm. Our work also leverages reinforcement learning to obtain the early-exiting policy. As our MDP has much smaller searching space, our proposed controller can be trained in an efficient non-sampling fashion.

7 CONCLUSION

In this work, we co-designed the multi-branch networks and the early-exiting policies in the context of dynamic inference. Our proposed solution, referred to as EPNet, addressed two key challenges, namely *(i)* designing the learning objective to balance both accuracy and efficiency; and *(ii)* explicitly considering the resource overhead associated with the early-exiting policies. Concretely, we designed a lightweight branch structure and cast the early-exiting problem as a Markov decision process. This enables EPNet to make exiting decisions *per convolutional layer* through the learned policy. Comparisons of EPNet on three datasets to two types of baselines demonstrate its efficacy in classification accuracy, adaptivity to sample difficulty, and resource budgets.

8 ACKNOWLEDGEMENT

We would like to thank all anonymous reviewers for their insightful comments, as well as Robert J. Walls for his discussion at the early stage of this project. This work was supported in part by NSF Grant CNS-1815619.

REFERENCES

[1] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. 2015. Multiple object recognition with visual attention. In *Proc. 3rd Int. Conf. Learning Representations (ICLR'15).*

[2] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16) Workshop.*

[3] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16).*

[4] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. 2017. Spatially adaptive computation time for residual networks. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17).* 1039–1048.

[5] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15).* 1135–1143.

[6] Thomas Hartvigsen, Cansu Sen, Xiangnan Kong, and Elke Rundensteiner. 2019. Adaptive-Halting Policy Network for Early Classification. In *Proc. 25th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'19).* 101–110.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. 2015 IEEE Int. Conf. on Computer Vision (ICCV'15).* 1026–1034.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16).* 770–778.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)Workshop.*

[10] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. 2018. Multi-Scale Dense Networks for Resource Efficient Image Classification. In *Proc. 6th Int. Conf. Learning Representations (ICLR'18).*

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17).* 4700–4708.

[12] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[13] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. 32nd Int. Conf. Machine Learning (ICML'15).* 448–456.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NeurIPS'12).* 1097–1105.

[15] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16).* 2554–2564.

[16] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. 2019. Improved Techniques for Training Adaptive Deep Networks. In *Proc. 2019 IEEE Int. Conf. on Computer Vision (ICCV'19).* 1891–1900.

[17] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27 (NeurIPS'14).* 2204–2212.

[18] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. 2013. Learning separable filters. In *Proc. 2013 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'13).* 2754–2761.

[19] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. 2016. Convolutional neural networks with low-rank regularization. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16).*

[20] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR).* IEEE, 2464–2469.

[21] Andreas Veit and Serge Belongie. 2018. Convolutional networks with adaptive inference graphs. In *Proc. 2018 the European Conf. on Computer Vision (ECCV).* 3–18.

[22] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning dynamic routing in convolutional networks. In *Proc. 2018 the European Conf. on Computer Vision (ECCV).* 409–424.

[23] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8 (1992), 229–256.

[24] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proc. 2018 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'18).* 6848–6856.