Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Energy-efficient Amortized Inference with Cascaded Deep Classifiers

Jiaqi Guan[1,2], **Yang Liu**[2], **Qiang Liu**[3], **Jian Peng**[2]
[1] Tsinghua University
[2] University of Illinois at Urbana-Champaign
[3] University of Texas at Austin
guanjq14@mails.tsinghua.edu.cn, liu301@illinois.edu, lqiang@cs.utexas.edu, jianpeng@illinois.edu

Abstract

Deep neural networks have been remarkable successful in various AI tasks but often cast high computation and energy cost for energy-constrained applications such as mobile sensing. We address this problem by proposing a novel framework that optimizes the prediction accuracy and energy cost simultaneously, thus enabling effective cost-accuracy trade-off at test time. In our framework, each data instance is pushed into a cascade of deep neural networks with increasing sizes, and a selection module is used to sequentially determine when a sufficiently accurate classifier can be used for this data instance. The cascade of neural networks and the selection module are jointly trained in an end-to-end fashion by the REINFORCE algorithm to optimize a trade-off between the computational cost and the predictive accuracy. Our method is able to simultaneously improve the accuracy and efficiency by learning to assign easy instances to fast yet sufficiently accurate classifiers to save computation and energy cost, while assigning harder instances to deeper and more powerful classifiers to ensure satisfiable accuracy. Moreover, we demonstrate our method's effectiveness with extensive experiments on CIFAR-10/100, ImageNet32x32 and original ImageNet dataset.

1 Introduction

The recent advances of deep learning techniques in computer vision, speech recognition and natural language processing have tremendously improved the performance on challenging AI tasks, including image classification [Krizhevsky *et al.*, 2012], speech-based translation and language modeling. Since the first success of deep convolutional neural network in the ImageNet challenge, more complex architectures [Simonyan and Zisserman, 2014; He *et al.*, 2016a; Szegedy *et al.*, 2017] have been proposed to further improve performance, but often at the cost of more expensive computation. However, in many real-world scenarios, such as vision-based robotics and mobile vision applications, we encounter a significant constraint of energy or computational cost for real-time inference. For example, mobile applications cast a high demand on fast, energy-efficient inference; it is desired to ensure that the majority (e.g., 90%) of the users do not feel the latency of the computation, given that most images are easy to analyze. This requires new learning methods that are *both accurate and fast*.

In this paper, we focus on test-time energy-efficient inference of image classification. Traditional approaches are usually based on directly scarifying accuracy for speed, e.g., by reducing or compressing well-trained complex neural networks at a cost of loss of accuracy. A key observation, however, is that accuracy and cost can be simultaneously improved, and do not necessarily need to scarify for each other; this is because although deeper or more complex networks usually come with higher overall accuracy, a large portion of images can still be correctly classified using smaller or simpler networks, and the larger networks are necessarily only for the remaining difficult images. Thus, the approach of our work is to jointly train an ensemble of neural networks with different complexity, together with a selection module that adaptively assigns each image to the smallest neural network that is sufficient to generate high-quality label. Unlike traditional learning approaches that learns with constant computation cost, our method learns to *predict both accurately and fast*. By training and using the policy module, our framework yields an efficient amortization strategy, which greatly reduce the computational or energy cost in the testing phase with even boosted predictive performance.

Technically, we frame the training of the neural classifiers and the selection module into a joint optimization of the training accuracy with a constraint on the expected computational cost (in terms of FLOPs cost). We design the policy module to be a optimal stopping process, which sequentially exams the the cascade of neural classifiers with increasing sizes (and hence predictive accuracies), and stops at the classifier that optimally trade-off the accuracy and complexity for each given image.

Our joint training is performed in an end-to-end fashion by the REINFORCE algorithm [Williams, 1992] to optimize a trade-off between the computational cost (in terms of FLOPs cost) and the predictive accuracy as reward signal. We per-

form experiments on the CIFAR and ImageNet classification datasets using a cascade of deep classifiers with varying sizes and architectures. As expected, on the CIFAR datasets, most images are assigned to the smaller networks which are already sufficiently predictive for them, while the remaining difficult images are assigned to larger and more powerful networks. And nearly half of the images are assigned to smaller networks on the ImageNet dataset. Our proposed model outperforms a well-trained accurate deep ResNet classifier in terms of accuracy but only requires less than 20% and 50% FLOPs cost on the CIFAR-10/100 datasets, 66% and 53% on the downsampled/original ImageNet dataset, respectively.

2 Related Work

There have been a number of existing methods on improving energy efficiency of deep neural networks. Most such techniques focus on simplifying network structure and/or improving basic convolution operations numerically. MobileNet [Howard *et al.*, 2017] uses depthwise separable convolutions to build light weight neural networks. ShuffleNet [Zhang *et al.*, 2017] uses pointwise group convolutions and channel shuffle operation to build an efficient architecture. Other techniques include pruning of connections [Han *et al.*, 2015] and bottleneck structure [Iandola *et al.*, 2016]. In addition to static techniques, Dynamic Capacity Network [Almahairi *et al.*, 2016] adaptively assigns its capacity across different portions of the input data by using a low and a high capacity sub-network. Spatially Adaptive Computation Time Networks [Figurnov *et al.*, 2017] dynamically adjust the number of executed layers for the regions of the image. Anytime Neural Networks [Hu *et al.*, 2017] can generate anytime predictions by minimizing a carefully constructed weighted sum of losses. Others [Li *et al.*, 2015; Yang *et al.*, 2016] consider cascaded classifiers in object detection to quickly reject proposals that are easy to judge. Conditional computation [Bengio *et al.*, 2015] and adaptive computation [Graves, 2016] propose to adjust the amount of computational cost by using a policy to select data. Many of these static and dynamic techniques are used in standard deep architectures such as ResNet [He *et al.*, 2016a] and Inception [Szegedy *et al.*, 2017], usually with a loss of accuracy. Different from these static and dynamic techniques, our method explicitly formulates the test-time efficiency as an amortized constrained sequential decision problem such that the expected computational cost, in terms of FLOPs cost, can be greatly reduced with even improved accuracy by adaptively assigning training examples with various difficulty to their best classifiers.

Adaptive Neural Network [Bolukbasi *et al.*, 2017](ANN) is the most relevant approach to ours. In their model, cascaded classifiers are trained and fixed, and only the termination policies are optimized to select the classifiers in a sequential manner. However, in our work, the classifiers in the cascade are also considered as a part of the entire policy so that they can be jointly optimized with the stopping policy module. Our experiments with the same baseline as ANN based on ImageNet dataset show that our method provides a more powerful tool to further improve the efficiency.

3 Method

In this section, we first formulate the energy-efficient inference problem as an optimization with amortized constraint. Then we reduce it to a sequential decision process and propose a solution based on REINFORCE algorithm. Finally, we introduce the details of implementation like classifier structure, policy module structure used in the experiments.

3.1 Energy-constrained Inference of Cascaded Classifiers

Classifiers, such as neural networks, are often more accurate with deeper or more complex architectures. However, the high computational or energy cost of complex networks are prohibitive for fast, real-time inference in applications deployed on mobile devices. If we have a cascade of classifiers with different sizes, it is possible to select the smallest, yet sufficiently powerful classifier for each input data to achieve both efficiency and accuracy simultaneously. This introduces our main problem: *Given a cascade of neural classifiers with different accuracies and cost, how to train them jointly together with an efficient selection mechanism to assign each data instance to the classifier that optimally trade off accuracy and cost?*

Specifically, suppose we have K classifiers $\{C_k\}_{k=1}^K$ with different energy cost $\{\mathcal{F}_k\}_{k=1}^K$. The energy cost $\mathcal{F}_k$ is assumed to correlate with the predictive capacity of classifiers, and can be, for example, the value of FLOPs or the number of layers in neural network classifiers. Given an input x, we denote by y its true label and $\hat{y} \sim C_k(\cdot|x)$ the label predicted by classifier C_k. In addition, we denote by $\Pi(k|x)$ a randomized policy that decides the probability of assigning input x to classifier C_k. Our target is to jointly train all classifiers $\{C_k\}$ and the policy $\Pi(k|x)$ to minimize the expected loss function under the constraint that the expected energy cost should be no larger than a desired budget $\mathcal{B}$, that is,

$$\max_{\Pi,\{C_t\}_{t=1}^K} \mathbb{E}_{(x,y)\sim\mathcal{D},k_x\sim\Pi(\cdot|x),\hat{y}\sim C_{k_x}(\cdot|x)}\left[-\mathcal{L}\left(\hat{y},y\right)\right]$$
$$s.t \quad \mathbb{E}_{(x,y)\sim\mathcal{D},k_x\sim\Pi(\cdot|x)}\left[\mathcal{F}_{k_x}\right] < \mathcal{B},$$

where k_x denotes the (random) classifier ID assigned to x. Further, we can reform the constrained optimization into an unconstrained optimization of a penalized cost function:

$$\max_{\Pi,\{C_t\}_{t=1}^K} \mathbb{E}_{(x,y)\sim\mathcal{D},k_x\sim\Pi(\cdot|x),y'\sim C_{k_x}(\cdot|x)}\left[-\mathcal{L}\left(y',y\right) - \alpha\mathcal{F}_{k_x}\right]$$

where α controls the trade-off between the predictive loss function and the energy cost. There is an (implicit) one-to-one map between the budget constraint $\mathcal{B}$ and the penalty coefficient α under which these two forms are equivalent in duality. We will use the penalized form in our experiments for its simplicity.

3.2 Energy Efficient Inference via Optimal Stopping

The design of the selection module Π plays an critical role in our framework. It should (i) get access to and efficiently leverage the information of the classifiers $\{C_k\}$ to make reasonable decisions, and (ii) be computationally efficient, e.g.,

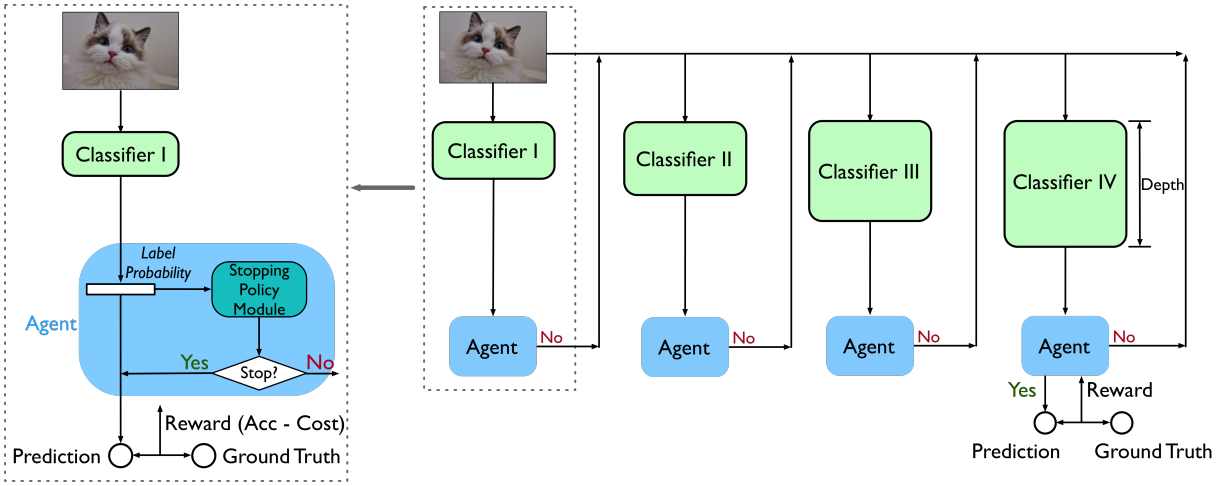

Figure 1: Our proposed model: Given an image in the dataset, starting from smallest model, our agent will decide whether to move to the next deeper model. If we decide to stop at a classifier, we predict the label based on the classifier. Finally, the agent will receive a reward as we described in the method section. Inside our agent, a stopping policy module takes label probability of a classifier's top layer as input and decides whether to stop or continue.

at least avoiding brute-forcely eliminating all the K classifiers and selects one the with largest confidence.

We propose to resolve this challenge by framing Π into a K-step optimal stopping process. At each time step t, we introduce a stopping policy module, which takes some feature $s_t(x)$ related to classifier C_t, and output a stopping probability $\pi_t(s_t(x))$ with which we decide to stop at the t-th classifier and take it as the final predictor for input x. Otherwise, we will move to a deeper classifier and repeat the same process until it reaches the deepest one. In this way, the overall probability of selecting at the k-th classifier is

$$\Pi(k|x) = \pi_k(s_k(x)) \prod_{t=1}^{k-1} (1 - \pi_t(s_t(x))).$$

Suppose we finally stop at the k-th classifier, our agent receives a reward consisting of two parts: the loss function for prediction using the selected classifier , i.e., $\mathcal{L}(\hat{y}, y)$ where $\hat{y} \sim C_k(\cdot|x)$, and the energy cost accumulated from the first classifier till current one, i.e., $\sum_{t=1}^{k} \mathcal{F}_t$. In practice, we also incorporate the accumulated computational cost of the stopping policy π_t in each $\mathcal{F}_t$. Importantly, once we stop at the k-th classifier, we no longer run the classifiers that are more expensive than k, which significantly saves the computational cost. Overall, this defines the following the reward signal:

$$R(k, x, y, \hat{y}) = -\mathcal{L}(\hat{y}, y) - \alpha \sum_{t=1}^{k-1} \mathcal{F}_t \qquad (1)$$

To recap, our decision module is framed as a Markov decision process consisting of the following components:

- **Observation:** The stopping probability $\pi_t(s_t(x))$ at the t-th step depends on a feature $s_t(x)$ which should represent the confidence level of the t-th classifier C_t. In this work, we simply use the output probability as the observation at each step, that is, $s_t(x) = C_t(\cdot|x)$.
- **Action:** Based on the output probability of the current classifier, our stopping policy module decides to stop at the current step with probability $\pi_t(s_t(x))$. If it finally stops at the k-th step, we use the current model C_k to predict the label, that is, $\hat{y} \sim C_k(\cdot|x)$.
- **Reward:** After finally stopping at one classifier, the agent receives a reward signal shown in Eq (1) consisting of both the negative loss function for prediction and the accumulated energy cost from the first step. In this paper, we use a normalized FLOPs count as the cost.

Assume the stopping probabilities $\{\pi_t\}$ and classifiers $\{C_t\}$ are parameterized by $\theta = \{\theta^{\pi_t}, \theta^{C_t}\}_{t=1}^{K}$. Our final goal is to find the optimal θ to maximize the expected return, by unrolling the conditional distributions defined by the entire policy:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\mathbb{E}_{k\sim\Pi(\cdot|x),\hat{y}\sim C_k(\cdot|x)} R(k, x, y, \hat{y})\big] \\ &= \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\sum_{k=1}^{K} \prod_{t=1}^{k-1} (1 - \pi_t(s_t(x); \theta)) \\ &\quad \cdot \pi_k(s_k(x); \theta) \cdot \sum_{\hat{y}} C_k(\hat{y}|x; \theta) \cdot R(k, x, y, \hat{y})\big]. \end{aligned}$$

3.3 Solving by REINFORCE

To solve this optimal stopping problem, we apply the well-known REINFORCE algorithm [Williams, 1992] by rolling out each individual sample (x, y) according to the current parameter and derive the policy gradient in following form:

$$\begin{aligned} \widehat{\nabla_\theta J} = \nabla_\theta \big(&\sum_{t=1}^{k-1} \log(1 - \pi_t(s_t(x); \theta)) + \log(\pi_k(s_k(x); \theta)) \\ &+ \log(C_k(\hat{y}|x; \theta))\big) \cdot (R(k, x, y, \hat{y}) - b) \end{aligned}$$

It is worthy to note that here we introduce a *baseline* b to reduce the variance in the estimated policy gradient. The *baseline* b is chosen by minimizing the variance of the gradient estimator on a mini-batch of the training data.

3.4 Cascaded Classifiers Using ResNet

In this paper, we use image classification for benchmarking our method. Deep residual network [He *et al.*, 2016a] has been widely used in image classification field since it was proposed. We choose ResNet as our model's baseline because we can easily build a sequence of networks from shallow to deep by adjusting the number of units in each block. Generally, the deeper network has the better prediction performance, though having more computational cost. The ResNet architecture we use is the same as the original version. Each block has different numbers of units and each residual unit performs the residual learning, which has a form $y = x + F(x, W_i)$, where x is called shortcut connection and F(x) is called residual function. The residual function we use is basic residual, which consists of two 3×3 convolution layers that both have equal input and output channels.Our ResNet is implemented in pre-activation [He *et al.*, 2016b] version, in which each convolution layer is preceded by a batch normalization layer [Ioffe and Szegedy, 2015] and a ReLU non-linear unit. In addition, after each block, the feature map size is halved and the number of filters is doubled, which follows the Very Deep Networks design [Simonyan and Zisserman, 2014] and ensures all units have equal computational cost.

We attach the policy network to this sequence of networks to achieve our algorithm. About the final output, drawing the technique of re-weighting all weak classifiers' results to give an ensemble and more accurate result in boosting, we use integrate outcomes from all previous classifiers instead of only the outcome from the stop classifier. The stop probability given by the policy module, on the other hand, can be seen as the confidence of current classifier's result. Thus, at the test stage, for those samples stopping at K-th classifier, we re-weight the outcomes of first K classifiers according to stopping probabilities that the policy module has ever given, and then output an ensemble result. After using this technique, the accuracy can be increased by about 0.1%.

4 Implementation and Experimental Setting

4.1 Datasets

As a proof of concept, we implement a cascade of deep neural network classifiers on four image classification datasets, including CIFAR-10, CIFAR-100 [Krizhevsky and Hinton, 2009], ImageNet32x32 [Chrabaszcz *et al.*, 2017] and original ImageNet [Russakovsky *et al.*, 2015] dataset. The first three datasets consist of 32x32 RGB colored images. The CIFAR-10 and CIFAR-100 datasets both have $50,000$ training images and $10,000$ test images, with 10 classes and 100 classes respectively. The ImageNet32x32 dataset is a down-sampled variant of original ImageNet dataset, which contains the same classes ($1,000$ classes) and the same number of images (1.28 million training images and 50k test images) with a reduced resolution of 32x32 pixels.

4.2 Neural Classifier Specification

To construct a cascade of neural network classifiers, we take the standard design of the ResNet architecture [He *et al.*, 2016a] to build a sequence of ResNets with nearly exponentially increasing depths. In this cascade, each ResNet classifier starts with a 3×3 convolution layer with 16 filters, followed by three blocks of residual units. Each unit consists of two 3×3 convolution layers. In the second and third blocks, the number of filters is doubled and the size of feature map is halved at the first unit. The numbers of units in each block are set to 1, 3, 5, 9, 18 to build this sequence of ResNets, with 8, 20, 32, 56, 110 layers respectively. For the ImageNet32x32 dataset, we adopt a width multiplier k following Wide-ResNet [Zagoruyko and Komodakis, 2016] to increase the capacity of individual classifiers by changing the number of filters. We set the width multipliers to 1, 1.5, 2, 3, 4 and the number of convolution layers to 40, so that the capacity or FLOPs of the ResNet classifiers are approximately exponentially increasing. We notice that this adoption works better than the original ResNet setting, due to the larger volume and higher diversity of the ImageNet dataset.

CIFAR			ImageNet32x32	
Layer	Unit Number	FLOPs(M)	Width Multiplier	FLOPs(M)
8	[1, 1, 1]	14.86	1	85.64
20	[3, 3, 3]	43.17	1.5	192.36
32	[5, 5, 5]	71.48	2	341.68
56	[9, 9, 9]	128.11	3	768.13
110	[18, 18, 18]	255.51	4	1364.97

Table 1: The ResNet structure used and their FLOPs.For the CIFAR datasets, our classifier cascade consists of ResNets with different layers; For the ImageNet dataset, it is ResNet with 40 layers of different widths.

Here our design ensures that the depth (and hence the computational complexity) of the cascade of neural network increases exponentially. This ensures that we do not waste significant computation resource in examining the smaller networks. To be more specific, assume the computational cost of the network classifiers are b^k, $k = 1, \ldots, K$, where $b > 1$, then the cost when we stop at the k-th classifier is $\sum_{\ell=1}^{k} b^l \leq \frac{b}{b-1} b^k$, which is at most $\frac{b}{b-1}$ times of b^k, the cost incurred when we select the k-th classifier by oracle, without examining any of the weaker classifiers.

In addition, to fairly compare to the ANN paper on ImageNet, we use pre-trained AlexNet [Krizhevsky *et al.*, 2012], GoogleNet [Szegedy *et al.*, 2014] and ResNet-50 from Caffe Model Zoo as baseline, which is same as the ANN paper. Their FLOPs are 7×10^9, 15×10^9 and 38×10^9 respectively.

4.3 Policy Module Specification

The stopping policy module is constructed using three fully-connected layers, with 64 hidden neurons each for CIFAR datasets, and with 256 hidden neurons for the ImageNet datasets. Each fully-connected layer is followed by a ReLU non-linear unit except the last layer. Finally, the fully-connected layers are followed by a softmax function which outputs the stopping probability. The input of this module is the label probability output from individual ResNets. This stopping policy module has nearly negligible computational cost ($\sim$0.01M FLOPs on the CIFAR datasets and $\sim$0.1M FLOPs on the ImageNet dataset), compared to that of the ResNets (see Table 1). We notice that other features, such as top-layer convolutional filters, would greatly increase the computational cost of this stopping policy module and have a lower classification accuracy.

CIFAR-10		
Model	Error	FLOPs
ResNet-8	12.33%	5.82%
ResNet-20	9.00%	16.90%
ResNet-32	8.40%	27.98%
ResNet-56	7.70%	50.14%
ResNet-110	7.38%	100.00%
Ours	**7.26%**	**18.16%**

CIFAR-100		
Model	Error	FLOPs
ResNet-8	39.98%	5.82%
ResNet-20	33.13%	16.90%
ResNet-32	31.56%	27.98%
ResNet-56	30.38%	50.14%
ResNet-110	28.63%	100.00%
Ours	**27.76%**	**48.98%**

ImageNet32x32 (Top-5 Error)		
Model	Error	FLOPs
ResNet40-1	39.72%	6.27%
ResNet40-1.5	32.76%	14.09%
ResNet40-2	29.64%	25.03%
ResNet40-3	24.67%	56.27%
ResNet40-4	22.22%	100.00%
Ours	**22.21%**	**66.22%**

ImageNet (Top-5 Error)		
Model	Error	FLOPs
AlexNet	20.08%	18.42%
GoogleNet	10.99%	39.47%
ResNet-50	7.80%	100.00%
ANN	8.14%	56.87%
Ours	**7.82%**	**53.47%**

Table 2: Comparing with Static ResNet Classifiers: We compare our model to well-trained, static ResNet classifiers with different sizes. In our method, the hyperparameter α is selected to match the accuracy of our method with that of the best static ResNet. Our model achieves not only lower error but also significantly less cost on all datasets.

4.4 Other Implementation Details

During the training phase, we adopt the standard data augmentation procedure [Lee *et al.*, 2015; He *et al.*, 2016a] on first three datasets: padding each side of the images by four zeros and randomly cropping a 32x32 image; randomly flipping left to right. For ImageNet dataset, we use the same data augmentation procedures used in original papers of baseline networks. For all the experiments, we use stochastic gradient descent with a momentum of 0.9 for the policy optimization. The learning rate schedule and the mini-batch size are set to be the same as in the original ResNet [He *et al.*, 2016a] for the gradients associated. The learning rate for the stopping policy module is set to be 0.1 and an exponential decay with a factor of 0.9 is applied every four epochs according to internal cross-validation within the training data.

To evaluate the classification performance, we use the top-1 accuracy for the CIFAR datasets and an additional top-5 accuracy for the ImageNet datasets, and they are also used in respective reward signals during training. We use the per-image number of floating-point operations (FLOPs) to measure the computational cost, with multiplications, additions and multiply-add operations considered. The network's size and FLOPs of individual ResNets are shown in Table 1.

5 Results

On CIFAR dataset, We compare our model to well-trained ResNet classifiers with 8, 20, 32, 56 and 110 layers, respectively, whose architectures are constructed in the same way as the classifiers in our cascaded model. We vary the hyper-parameter α, the coefficient of the energy cost in the reward signal, in the range of 10^{-4} to 10^{-2} to demonstrate the trade-off between the computational cost (FLOPs) and accuracy. The comparisons on CIFAR-10 and CIFAR-100 are shown in Figure 2 and Table 2. In Figure 2, the gray curve shows the performance of the static ResNet classifiers with difference numbers of layers, and the orange curve shows the performance of our model with different values of α. Clearly, our model achieves not only better classification accuracy but also higher cost effectiveness. Our model can achieve 0.12% higher accuracy with only 18.16% FLOPs on the CIFAR-10 dataset, compared to the best performing static ResNet110 classifier. On the CIFAR-100 set, our model obtains 0.87% higher accuracy with only 48.98% FLOPs, compared to the best static ResNet110 classifier. It is worth noting that the efficiency improvement is more prominent on the CIFAR-10 than the CIFAR-100, due to the fact that CIFAR-100 is a more challenging dataset so that deeper classifiers are more frequently required to distinguish similar labels. Similarly,

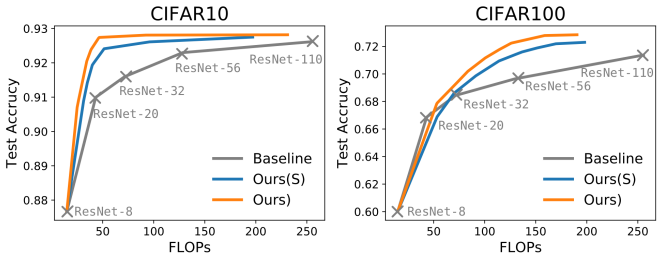

Figure 2: Results on CIFAR-10/100: The x-axis denotes the millions of FLOPs and y-axis denotes the corresponding accuracy obtained by the static ResNet (gray), our model (orange), and the simplified version of our model (blue), respectively. It is obvious that our models outperform the static ResNet under the same energy cost on a large spectrum.

the result on the ImageNet32x32 dataset (Table 2) shows that our model can achieve almost the same top-5 accuracy compared with the largest static ResNet classifier but only requires 66.22% computational cost. On ImageNet dataset, we construct a cascade of AlexNet, GoogleNet, and ResNet-50 as the baseline, which is same as the Adaptive Neural Network(ANN) paper to conduct a fair comparison. The result (Table 2) shows that our model can achieve almost the same top-5 accuracy compared to ResNet-50 but requires only 53.47% FLOPs, while the ANN requires 3.4% more FLOPs but the accuracy is 0.32% lower than ours.

As another baseline, we have implemented a simplified version of our model (denote as "Ours (S)" in Figure 2), in which we only train the stopping policy model to sequentially decide which the classifiers to use, and these classifiers within the cascade are pre-trained and fixed. The blue curve in Figure 2 indicates that this version can also outperform the best ResNet110 classifier as it dynamically decides the smallest classifier that is sufficient for a input image. Our jointly trained model shows further improved performance, especially when the amortized FLOPs required is small, as our model also determines which images to use for classifiers in the cascade during training, compared to this simplified version.

Further Analysis

We further investigate how our model works exactly by visualizing representative samples assigned to different classifiers. Figure 3 shows five images stopped at various classifiers and their label probability distributions given by the classifiers they visited before stopping. Taking the third samples as examples, we can see that the *turnstile* is first confused with *forklift* by the first two classifiers, but is correctly classified by the third classifier on which it stops. In the fifth image, *howler monkey* is identified by the fifth classifier, but is wrongly pre-

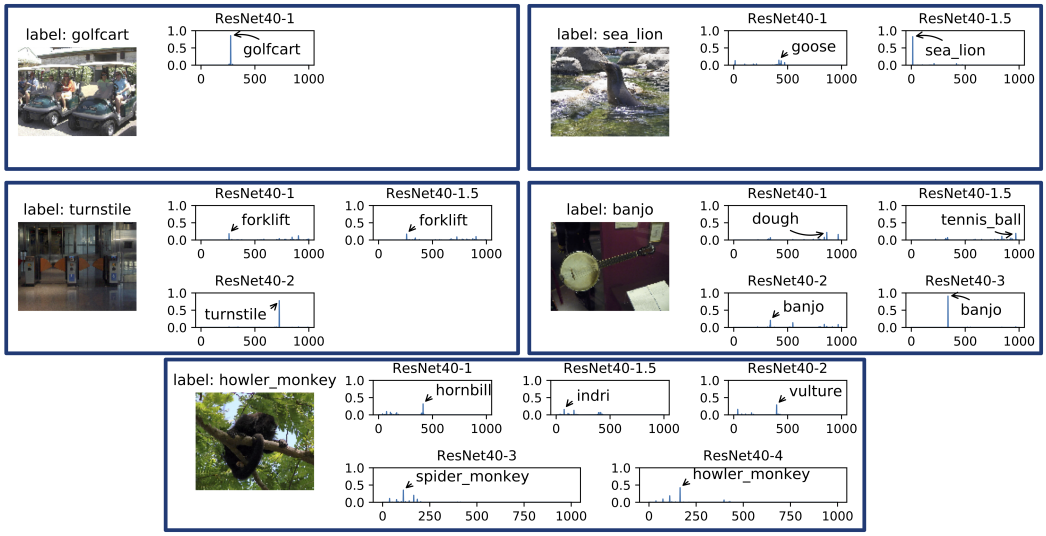

Figure 3: Example images and their predicted label probabilities: Five images from ImageNet assigned to different ResNet classifiers, with their label probability distributions given by the classifiers they visited.

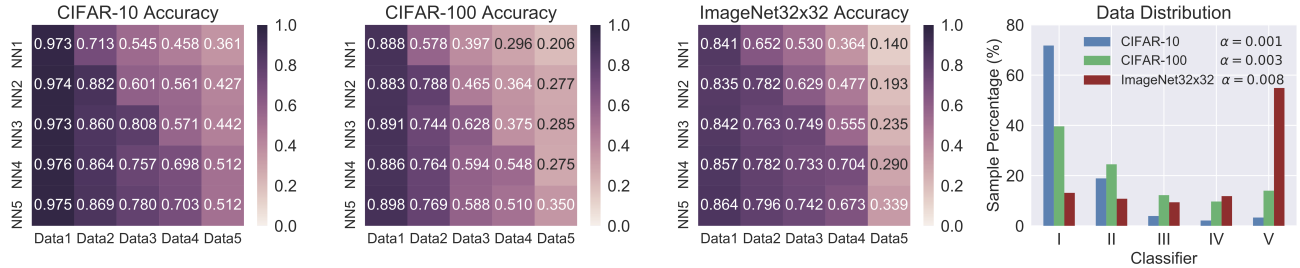

Figure 4: Accuracy distribution of classifiers in the cascade: The first three figures plot accuracy distributions on the CIFAR-10, CIFAR-100 and ImageNet32x32 datasets. The i-th row and j-th column denotes the average accuracy predicted by the i-th classifier of samples assigned to the j-th classifier. The fourth figure is the distribution of test images on individual classifiers, where x-axis indexes five classifiers and y-axis denotes the proportion of samples eventually assigned to the corresponding classifier.

dicted by all the first four classifiers as other classes, including *spider monkey* which is indeed easily confused with the true label.

We can divide the testing datasets into 5 subsets according to which classifier an image is assigned to by the selection module. This means that subset i contains all the images assigned to the i-th classifier in the cascade. Figure 4 shows the average accuracy of the different subsets on different networks. We can see that for the data assigned to classifier i, the accuracy at classifier i is consistently higher than the accuracies of classifier 1 to $i-1$ on all three datasets (i.e., the diagonals are larger than the upper triangular elements), suggesting that the selection modules successfully identify more accurate classifiers. Interestingly, we find that the accuracy does not always increase when ResNet becomes deeper. For example, on the relatively simple datasets, CIFAR-10 and CIFAR-100, the accuracies of subset i at classifier i (i=2,3,4,5) are even higher than accuracies at classifier $i+1$ to 5. This is because the REINFORCE algorithm distributes only harder images to the deeper ResNets, making them less accurate on the easier images.

The rightmost panel of Figure 4 shows the proportions of the 5 subsets in different datasets. We can see that in CIFAR-10 (blue) and CIFAR-100 (green), most images are easy to classify and are assigned to the smaller classifiers, while ImageNet is more difficult and a majority of it is assigned to the largest classifier. Even in ImageNet, our method successfully identifies a large portion of easier images, and hence obtain better average FLOPs than the biggest static ResNet.

6 Conclusion

In this work, we propose an energy-efficient model by cascading deep classifiers with a policy module. The policy module is trained by REINFORCE to choose the smallest classifier which is sufficient to make accurate prediction for each input instance. Tested on image classification, our model assigns a large portion of images to the smaller networks and remaining difficult images to the deeper models when necessary. In this way, our model is able to achieve both high accuracy and amortized efficiency during test time. We evaluate our energy-efficient model on the CIFAR-10, CIFAR-100 and downsampled and original ImageNet datasets. It obtains nearly the same as or higher accuracy than well-trained deep ResNet classifiers but only requires approximately 20%, 50% and 66% and 53% FLOPs cost respectively. With a spectrum of computational cost parameter α values, our model achieves different trade-offs between amortized computational cost and predictive accuracy.

References

[Almahairi *et al.*, 2016] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. Dynamic capacity networks. In *International Conference on Machine Learning*, pages 2549–2558, 2016.

[Bengio *et al.*, 2015] Emmanuel Bengio, Pierre Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *Computer Science*, 164(1):34–44, 2015.

[Bolukbasi *et al.*, 2017] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. 2017.

[Chrabaszcz *et al.*, 2017] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[Figurnov *et al.*, 2017] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1790–1799, 2017.

[Graves, 2016] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[Han *et al.*, 2015] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *Fiber*, 56(4):3–7, 2015.

[He *et al.*, 2016a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[He *et al.*, 2016b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[Howard *et al.*, 2017] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[Hu *et al.*, 2017] Hanzhang Hu, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Anytime neural networks via joint optimization of auxiliary losses. *arXiv preprint arXiv:1708.06832*, 2017.

[Iandola *et al.*, 2016] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Lee *et al.*, 2015] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*, pages 562–570, 2015.

[Li *et al.*, 2015] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Science*, 2014.

[Szegedy *et al.*, 2014] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. pages 1–9, 2014.

[Szegedy *et al.*, 2017] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[Yang *et al.*, 2016] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2129–2137, 2016.

[Zagoruyko and Komodakis, 2016] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[Zhang *et al.*, 2017] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.