Adaptive Multi-Exit Neural Networks using EM-Based Routing

Jason Kyauk
Stanford University
jkyauk@stanford.edu

Abstract

Real-time vision systems must operate under strict latency and compute constraints, particularly in safety-critical domains such as robotic surgery. Standard CNNs apply full-depth computation to all inputs, regardless of difficulty, causing unnecessary overhead on embedded hardware. In this work, we introduce a probabilistic routing mechanism for multi-exit CNNs based on Expectation–Maximization, treating exit identity as a latent variable and optimizing a likelihood–cost objective. Using a ResNet-18 backbone with four early exits on CIFAR-10, we show that EM-based routing achieves competitive accuracy–compute trade-offs compared to BranchyNet, reaching **89% accuracy at only 75% compute**. Our results demonstrate that probabilistic routing is a principled and effective alternative to heuristic confidence-based early exiting.

1 Introduction

Real-time computer vision systems, particularly in domains like robotic surgery, have many restrictions when it comes to latency and compute power, as surgeons benefit and more efficient when lag is minimal, and surgical robots mostly rely on compute from within itself, rather than a supercomputer. At inference time, these CV systems will have to process inputs from varying complexity, from routine tissues, to a messy scene with lots of smoke from a cauterizer and/or blood. However, a standard CNN is static in terms of when it exits, so it applies the same extensive computation to every input frame, despite easier frames not needing the final few layers to be accurately determined a lot of the time. This bottleneck can lead to hindered performance in embedded hardware, especially when it comes to real-time feedback. To address this issue, we enter the realm of adaptive computation for neural networks, in our particular case CNNs.

There already has been an extensive amount of work in adaptive-compute networks, as we will talk about some key examples in the Related Work section. However, a lot of these approaches are heuristic, and few attempt to try to model early-exit decisions based off the underlying probablistic structure of the data. Switching to the lens of probability in theory seems beneficial as it captures how likely each exit is to produce the correct label given observed features. This provides a more principled description of how a model's predictions relate to the true behavior of our data. Having this perspective allows for exit selection to be governed by likelihood rather than heuristics, which is one step closer to ground truth.

The work here proposes a probabilistic routing mechanism for Multi-Exit Neural Networks, more specifically focusing on a network consisting of exit classifiers connected by a ResNet-18 backbone that has been trained on CIFAR-10, and frozen. Therefore inputs to the whole network is the image, and each exit classifier's input is the pooled features from the immediate layer of the ResNet-18 backbone connected to the classifier. We then introduce another neural network, which represents a router, whose inputs are the standard image input from CIFAR-10, which is trained to predict give soft probabilities of which exit classifier the multi-exit neural network should exit at per input. More specifically, we treat each exit classifier as a latent variable z, and maximize the likelihood of the correct class label while penalizing by the cost of using a certain exit classifier, a slight modification to the EM approach we typically see in CS 229. This proposed pipeline allows us to learn a routing policy that isn't based off prediction confidence, but rather a more principled trade-off between likelihood of correctness and the compute resources it requires.

2 Related Work

Early-Exit Networks and Heuristic Routing:

Most adaptive-compute methods for CNNs rely on early exits trained alongside a deep backbone. **BranchyNet** [1] (in which the architecture for this work was partially inspired by) attaches side-classifier branches and determines exit using **confidence-based thresholds**, in particular softmax entropy. This simple idea showed promise as it proved these thresholds can substantially reduce inference cost, with only modest loss in accuracy. This is the biggest strength, as it showed great result with easy implementation. One of its weaknesses however is that shallow early exits, since they predict off shallow features, aren't the most accurate. **MSDNet** [2] extends this idea with a multi-scale architecture that improves the accuracy for the shallower early exit layers. However it is computationally heavy. On a higher level, the heuristic approach works, but comes with a major limitation, it is a heuristic. Confidence isn't a calibrated, or grounded estimate of correctness, and confidence scores may shift unpredictably across datasets, backbones, and during training. As a result, heuristic policies are sensitive to threshold choices, unreliable under distribution shifts, and unable to express uncertainty about exit selection. **EENets** [5] have attempt to address this limitation by learning an exit policy rather than relying on manually tuned thresholds. Instead of using entropy, they train a small network which outputs an efficiency score and halt when score exceeds a cost-informed threshold. This removes the hand-designed heuristics, and proves is strong thus is current SOTA, alongside BranchyNet, but the routing rule is deterministic. This doesn't allow the model to represent exit identity as a latent variable, and thus can not express uncertainty, nor reason about the expected accuracy-cost tradeoff.

Adaptive Computation and Probabilistic Halting:

The foundation of a probabilistic lens was validated by Graves through **Adaptive Computation Time (ACT)** [3], in which the model learns a sigmoidal halting distribution to determine how many recurrent updates a sequences model should perform. ACT formulates the halting in itself as a latent variable and optimizes an expectation of compute cost, eventually proving that probabilistic formulations can lead to more interpretable and flexible allocation of compute. Its main strength lies in providing a more grounded perspective, in that of a principled probabilistic framework, demonstrating that halting decisions can be learned than manually specified. **AdaTape** [4] extended this idea to Transformer architectures, showing promise that it can translate to different types of deep neural network architectures.

Our Approach:

Despite the works in adaptive computation, there is currently a large gap in research connecting these two concepts together. Thus, our work will draw inspiration from Grave's probabilistic perspective, and apply it to early-exit CNNs. Whereas early exiting is governed by deterministic confidence scores, or thresholding heuristics, we instead treat the exit identity as a latent variable, and learn a probabilistic routing distribution over all exits that optimizes an expected utility, balancing correctness and computation.

3 Dataset and Features

We evaluated our method on the CIFAR-10 dataset, a standard benchmark for image classification. Use of this dataset allows for direct comparison with minimal computational overhead.

Data Statistics

- **Classes:** 10 mutually exclusive classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
- **Training Set:** 50,000 labeled RGB images.
- **Test Set:** 10,000 labeled RGB images.
- **Image Size:** Each image is 32×32 with 3 channels.

The raw pixel values are normalized using standard channel-wise means and standard deviations for CIFAR-10. We apply standard data augmentation techniques (random cropping and horizontal flipping).

Intermediate feature maps from the ResNet-18 backbone are extracted after each residual block group and serve both classification and routing purposes.

Feature Dimensions

- Exit 1: $64 \times 32 \times 32$
- Exit 2: $128 \times 16 \times 16$
- Exit 3: $256 \times 8 \times 8$
- Exit 4: $512 \times 4 \times 4$

Before features are inputted into the exit classifier, a pooling function is applied on the features, to average out height and width into scalars, as we do not need spatial dimensions for the exit classifier, thus giving inputs to the exit classifier as just the channels.

4 Methods

4.1 Multi-Exit Architecture

For this project, a 4-Exit ResNet-18 was used as the main classification pipeline, and what we are trying to reduce compute on. First a ResNet-18 was loaded in, and retrained on the CIFAR-10 dataset to maintain consistency and to ensure high accuracy come inference time, as we are inferring based off the CIFAR-10 test set. Then at each ResNet Block (4 total), we attached a exit classifier head to it. The exit classifier is a 2-layer Multi-Layered Perception.

After retraining the ResNet-18, we then freeze the backbone, to simplify the training process of the exit classifiers and to avoid gradient conflicts, where if we didn't then layers to exit 1 would be training to make the features highly discriminant, which destroys features come the final exit layer. Therefore it is better to train them as if it were to be 4 different networks, just at varying depths, thus a frozen backbone.

Visual of Multi-Exit ResNet-18

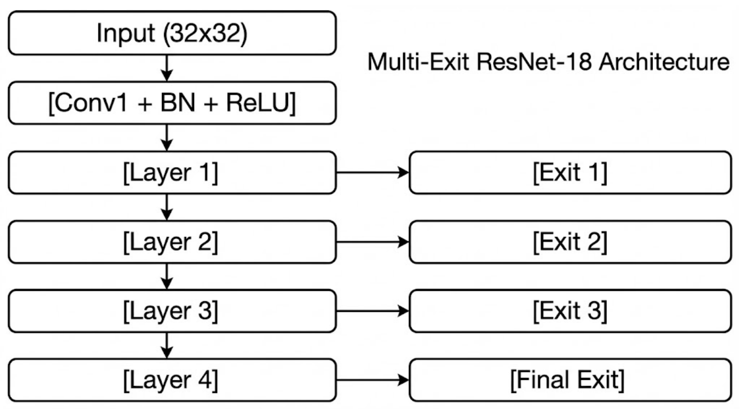

Figure 1: Multi-Exit ResNet-18 Architecture

4.2 Feature Caching

One of the algorithms we use then, to make the rest of the pipeline efficient and making use of our resources is the feature caching algorithm. The goal of this algorithm is to simply just save the features and separate them by which layer they were generated at. This feature caching algorithm is an important background algorithm, as our exit classifiers are trained on this data, and we don't want to have to retrain the ResNet backbone everything we want to train a new exit classifier that sits at a new depth.

4.3 Expectation–Maximization Routing

This is the main algorithm of this whole project, and the glue piece of the work. The goal of our project: how will the model decide when to exit early? In a probabilistic lens, this question then becomes: $P(z = k|x, y)$, or the probability that we should exit at exit head k, given our input and label. Since we don't exactly have that data we can use Baye's Rule to reform this question into:

$$P(z = k|x, y) = \frac{p(y|x, z = k) * p(z = k|x)}{\sum_j^M p(y|x, z = j) * p(z = j|x)},$$

where M = num of exit heads We then now see that we can find $p(y|x, z = k)$, but we don't know $p(z = k|x)$ just yet. However the the Expectation-Maximization algorithm can be applied until convergence (which is guaranteed by Jensen's Inequality), to solve for this. We initially set $p(z|x) = 0.25$ for all M heads, and repeat the Expectation and Maximization step.

For numerical stability, we want to use log values. Therefore:

$$\begin{aligned} P(z = k|x, y) &= \frac{e^{log(P(y|x,z=k)*p(z=k|x))}}{\sum_j^M e^{log(P(y|x,z=j)*p(z=j|x))}} \\ &= softmax(log(P(y|x, z = k)) + log(p(z = k|x))) \end{aligned} \tag{1}$$

This is the EM algorithm that only concerns itself of which is the best exit to take. Naturally, this will obviously skew towards the deepest exit as it has the most y labels right given that exit, as the deeper the network, the better the accuracy. So a cost regularizer was added to punish high-cost entries. This took in the form of $-\lambda(cost(k))$ where $cost(k)$ is the cost at exit k. Fitting this all into the Bayes' derivation in log-space it becomes:

$$
\begin{aligned}
P(z=k|x,y) &= \frac{e^{log(P(y|x,z=k)*P(z=k|x)*e^{-\lambda(cost(k))})}}{\sum_j^M e^{log(P(y|x,z=j)*P(z=j|x)*e^{-\lambda(cost(k))})}} \\
&= softmax(log(P(y|x,z=k)) + log(p(z=k|x)) - \lambda(cost(k)))
\end{aligned}
\tag{2}
$$

where λ is a tunable parameter, This is the key equation that'll allow us to predict the optimal exit head k that balances classification accuracy against computational cost.

4.4 Routers and Router Training

While the E-step provides an optimal routing distribution offline (since it requires access to the ground-truth label), we require a mechanism to perform routing at inference, when labels are unavailable. To accomplish this, we introduce a set of lightweight *routers*, one for each exit. Each router is trained to estimate the probability that its corresponding exit should be selected for a given input. Because routing should not give excesss overhead in compute, we employ a small MLP for each router.

Router k receives the feature representation $f_k(x)$ at its corresponding exit and outputs a scalar logit $s_k(x)$. During inference, this logit is passed through a sigmoid activation to produce a exit probability

$$p_k(x) = \sigma(s_k(x)),$$

which we interpret as the probability of exiting at exit k, conditioned on not exiting at any earlier exit. This induces a natural sequential exiting process: Router 1 is queried first; if the model chooses not to exit, Router 2 is evaluated, and so on until the final exit.

To train the routers, we use the soft posterior assignments produced by the EM E-step. These assignments represent the probability that exit k is the correct exit for input x. However, because our routers implement a sequential halting model, we convert the EM posteriors into the appropriate conditional probabilities. For example, the target for Router 2 becomes

$$t_2(x) = \frac{r_2(x)}{r_2(x) + r_3(x) + r_4(x)},$$

and the remaining routers are defined analogously.

Each router is trained using binary cross-entropy loss, which corresponds to minimizing the negative log-likelihood of a Bernoulli variable with soft targets. This choice is appropriate because routers produce a single logit (interpreted as the log-odds of halting), and the soft targets $t_k(x) \in [0,1]$ represent fractional halting probabilities derived from EM. After training, the routers provide a fast, label-free approximation of the EM routing distribution, enabling efficient early-exit decisions at inference time.

4.5 Inference

During inference, we will gather the first layer's features. Then the probability of exiting at that first exit layer will be calculated by doing the sigmoid of the 1st router's output on those features. If the probability is below threshold, we keep going until we hit an exit probability that is above threshold, or we reach the final exit.

5 Experiments and Results

Our central finding is that the proposed EM-based routing method achieves **89% accuracy at only 0.75 normalized compute**, demonstrating that a probabilistic latent-variable formulation can match the performance of heuristic early-exit strategies such as BranchyNet (0.876 accuracy at 0.645 compute). This provides strong evidence that balancing likelihood and computational cost through EM is a viable alternative to manually tuned confidence thresholds.

We present quantitative and qualitative evaluations of the system, compare against several baselines, and analyze why the EM objective produces meaningful exit-selection behavior.

5.1 Metrics

The primary metric is multi-class classification accuracy,

$$\mathrm{Acc} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i),$$

and normalized computational cost, defined as the expected FLOPs of the selected exit relative to the full ResNet-18 forward pass. Cost thus satisfies

$$\mathrm{Cost} = \mathbb{E}_x\big[C(z(x))\big], \qquad C(z) \in \{0.25,\ 0.50,\ 0.75,\ 1.00\}.$$

5.2 Baselines

We compare our method against several baselines to interpret accuracy–cost trade-offs:

1. **Standard ResNet-18**: Full-compute upper baseline.
2. **Fixed Multi-Exit ResNet**: Evaluates the raw accuracy of each exit in isolation.
3. **Random Routing**: A lower-bound that samples exits uniformly.
4. **Oracle Routing**: Upper bound obtained by selecting the lowest-cost exit that predicts the correct class.
5. **BranchyNet (Entropy Thresholding)**: A classical early-exit method serving as our primary benchmark.

The accuracies and costs of each baseline are given in Table 1.

Method	**Accuracy**	**Cost**
ResNet-18	0.9497	1.0000
Exit 1 Only	0.4359	0.2500
Exit 2 Only	0.6635	0.5000
Exit 3 Only	0.8984	0.7500
Exit 4 Only	0.9497	1.0000
Random Routing	0.7501	0.6384
BranchyNet	0.8756	0.6452
EM Routing (best)	0.8984	0.7500
Oracle Routing	0.9691	0.4855

Table 1: Accuracy–cost comparison across all baselines.

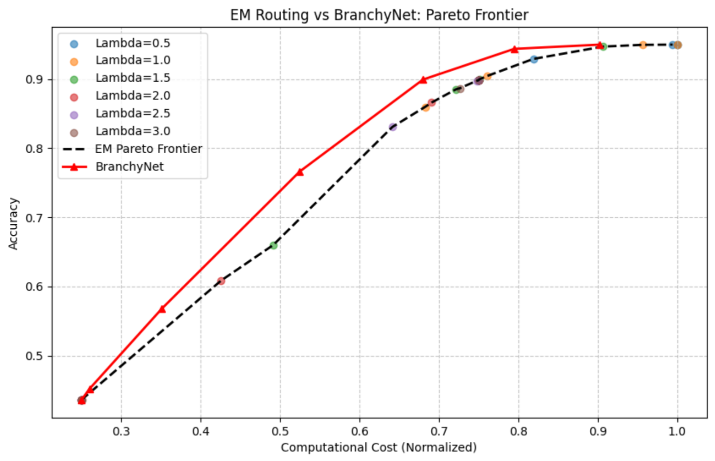

Figure 2: Pareto Plot vs Branchy Net

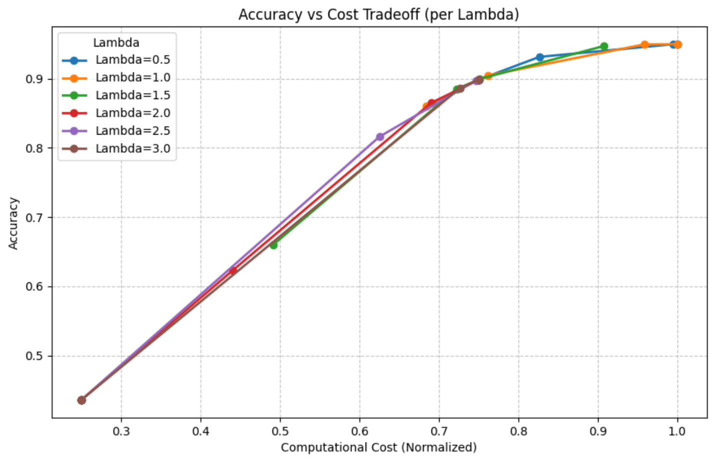

Figure 3: Lambda Curves Distributions

5.3 Accuracy–Cost Trade-Off (Pareto Frontier)

Figure 2 shows the accuracy–cost Pareto frontier for EM routing across a sweep of λ values, alongside the BranchyNet curve.

The key result is that EM achieves **0.89 accuracy at 0.75 cost**, and **0.865 Accuracy at 0.69 cost** (Figure 3) making it competitive with BranchyNet at similar computing budgets. Low values of λ favor deeper exits and higher accuracy, while large λ penalizes compute and pushes the model toward early exits. The smoothness and monotonicity of this curve demonstrate that the EM objective correctly modulates exit depth according to cost.

Although BranchyNet slightly outperforms EM at very high compute (reaching ≈ 0.94 accuracy at cost ≈ 0.90), EM closely tracks it across the critical mid-cost regime.

5.4 Baseline Comparison

Figure 4 visualizes accuracy and cost across all baselines. Several conclusions follow:

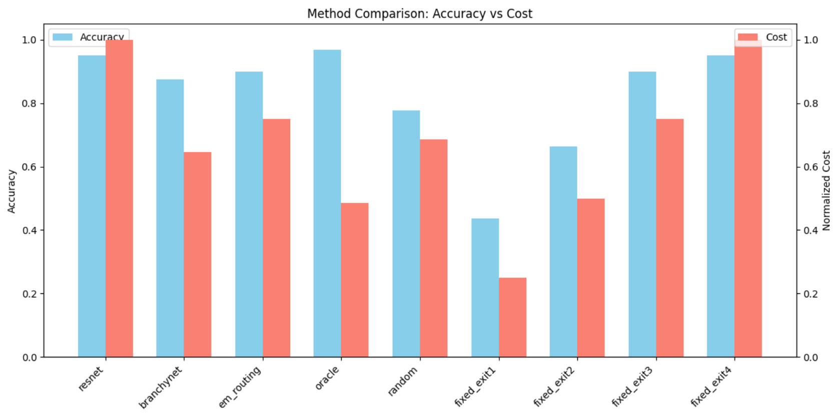

Figure 4: Bar chart comparing all baselines vs EM-based routing method

- The fixed exits show a steadily increasing accuracy ladder, validating that the architecture supports progressively refined features.
- Random routing performs significantly worse than both BranchyNet and EM, confirming that non-adaptive policies are insufficient.
- EM routing's best configuration aligns with Exit 3, demonstrating that the learned routing policy correctly identifies the depth that balances accuracy and compute under the cost constraint.
- The oracle baseline shows that as much as 0.97 accuracy is attainable at half the cost, highlighting headroom for future work.

5.5 Behavior at Extreme λ Values

For very large λ, EM routing reduces to always selecting Exit 1, yielding $(\mathrm{Acc}, \mathrm{Cost}) = (0.4359, 0.2500)$. This is not a failure mode but rather a confirmation that the cost term dominates the likelihood term under strong regularization, forcing the cheapest possible exit regardless of accuracy.

This behavior supports the validity of the probabilistic formulation: EM optimally shifts probability mass as the cost-pressure changes.

5.6 Summary

Across all baselines and trade-offs, the EM-based method demonstrates that a latent-variable perspective can perform comparably to heuristic early-exit methods. The result of **89% accuracy at 0.75 compute** shows that EM routing not only works in theory but also in practice on a real multi-exit CNN.

6 Conclusion and Future Work

In this project, we've successfully demonstrated that a EM-based approach, where we have exit layers as latent variables, can learn effective routing policies for multi-exit neural networks. By training the lightweight routers to mimic the optimal posterior distribution of exit choices, we've achieved a significant reduction in computational cost (25 percent) with minimal reduction in accuracy on the CIFAR-10 dataset. This work validates that difficulty is a learnable property of our input data, and that a probabilistic approach is a competitive alternative to the heuristic thresholds currently known in adaptive computation. One future path that may be explored after this is getting rid of a frozen backbone, and figure out a way to prevent gradient conflict, thereby allowing for the integration of routing and exit-head training into a fully joint EM-style procedure, rather than separate. Additionally, evaluating this method in larger scale datasets, and deeper models may show even more benefit in terms of compute saved, and is worth exploring.

References

[1] Teerapittayanon, S., McDanel, B., & Kung, H. (2016). BranchyNet: Fast inference via early exiting. *arXiv:1709.01686*.

[2] Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. Q. (2018). Multi-scale Dense Networks for Resource Efficient Image Classification. *arXiv:1703.09844*.

[3] Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983*.

[4] Sukhbaatar, S., Xu, Z., Vinyals, O., & Denoyer, L. (2023). AdaTape: Adaptive Computation for Multimodal Transformers. *arXiv:2301.13195*.

[5] Dimitriev, R., Johnson, J., & Giryes, R. (2024). EENets: Early Exit Networks via Learned Halting Policies. *arXiv:2409.05336*.