# HydraNets: Specialized Dynamic Architectures for Efficient Inference

Ravi Teja Mullapudi
CMU
rmullapu@cs.cmu.edu

William R.Mark
Google Inc.
billmark@google.com

Noam Shazeer
Google Inc.
noam@google.com

Kayvon Fatahalian
Stanford University
kayvonf@cs.stanford.edu

## Abstract

*There is growing interest in improving the design of deep network architectures to be both accurate and low cost. This paper explores semantic specialization as a mechanism for improving the computational efficiency (accuracy-per-unit-cost) of inference in the context of image classification. Specifically, we propose a network architecture template called HydraNet, which enables state-of-the-art architectures for image classification to be transformed into dynamic architectures which exploit execution for efficient inference. HydraNets are wide networks containing distinct components specialized to compute features for visually similar classes, but they retain efficiency by dynamically selecting only a small number of components to evaluate for any one input image. This design is made possible by a soft gating mechanism that encourages component specialization during training and accurately performs component selection during inference. We evaluate the HydraNet approach on both the CIFAR-100 and ImageNet classification tasks. On CIFAR, applying the HydraNet template to the ResNet and DenseNet family of models reduces inference cost by 2-4× while retaining the accuracy of the baseline architectures. On ImageNet, applying the HydraNet template improves accuracy up to 2.5% when compared to an efficient baseline architecture with similar inference cost.*

## 1. Introduction

Deep neural networks have emerged as state-of-the-art models for various tasks in computer vision. However, models that achieve top accuracy in competitions currently incur high computation cost. Deploying these expensive models for inference can consume a significant fraction of data center capacity [19] and is not practical on resource-constrained mobile devices or for real-time perception in the context of autonomous vehicles.

As a result, there is growing interest in improving the design of deep architectures to be both accurate and computationally efficient. In many cases, the solution has been to create new architectures that achieve a better accuracy/cost balance. Like most architectures, these new designs are *static architectures* that evaluate the entire network for each input, regardless of the input's characteristics. In contrast, *dynamic architectures* attempt to specialize the work performed during inference to properties of a particular input. For example, a cascade is a dynamic architecture that employs multiple models with different computational costs and uses low cost models to "early out" on easy-to-process inputs.

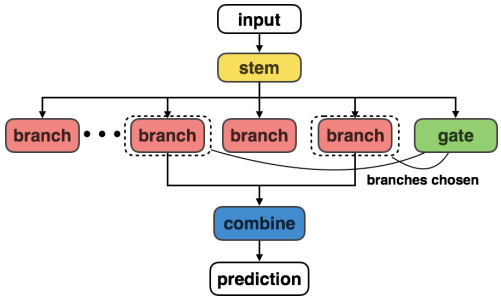



Figure 1: The HydraNet template architecture: contains multiple *branches* specialized for different inputs and a *gate* chooses which branches to run when performing inference on an input, and a *combiner* that aggregates branch outputs to make final predictions.

In this paper, we explore a dynamic architecture template, which we call *HydraNet*, which achieves efficiency gains by dynamically determining which subset of the architecture to run to best perform inference on a given input. In other words, a HydraNet maintains accuracy by having large capacity that is semantically specialized to aspects of the input domain. However, a HydraNet is computationally efficient because it only uses a small fraction of this capacity when performing inference on any one input.

Our primary contribution is the HydraNet architecture template, shown in Figure 1, which is a recipe for transforming state-of-the-art static network designs into dynamic architectures that offer better accuracy-per-unit-cost. We evaluate HydraNets in the context of image classification where we specialize network components for visually similar classes. The specialized components can also be viewed as performing different subtasks in the overall classification

task. The idea of dynamically routing inputs to specialized subnetworks is conceptually simple. However, training the components of a network to specialize for subtasks and successfully exploiting dynamic execution requires careful design of network components that balance the need to aggressively, but accurately, determine what subset of the architecture to run for a given input.

## 2. Related Work

**Sparsity.** The common approach to improving DNN efficiency is to enforce sparsity in network connectivity [21, 29, 4, 13]. This can be achieved via manual design of new DNN modules (Inception [27], SqueezeNet [16], MobileNet [13]) or via automated techniques that identify and remove the least important connections from a dense network [11, 10]. In either case, determination of the network topology is a static preprocessing step, and all connections are evaluated at the time of inference.

A complimentary optimization is to employ conditional execution at the time of inference to exploit sparsity in activations (skipping computation and memory accesses for model weights when activations are known to be zero). While attractive for reducing the energy consumption of DNN accelerators [9], fine-grained, per-element sparsity is difficult to exploit on CPUs and GPUs, which rely heavily on wide vector processing for performance. The subtask specialization we exploit in HydraNets can be viewed as a mechanism for designing and training a network architecture that, through coarse-grained conditional execution, is able to more effectively exploit dynamic sparsity.

**Cascades.** Cascades [28] are a common form of conditional model execution that reduces inference cost (on average) by quickly terminating model evaluation on inputs that are easy to process ("early out"). Region proposal [24] based models for object detection are canonical example of cascades in DNNs. Recent work has shown that integrating cascades into deep network architectures [8, 14] can improve the accuracy vs. cost of state-of-the-art architectures, where the later stages in a cascade specialize for difficult problem instances. The HydraNet approach of specializing network components for different subtasks is orthogonal and complementary to the benefits of cascades.

**Mixture of experts.** The idea of specializing components of a model for different subtasks is related to mixture-of-experts models where the experts are specialized for different inputs or tasks. Recent work on training very large DNNs for language modeling [26] has used conditional execution of experts for evaluating only a small fraction of experts for each training instance. One of the key aspects addressed in [26] is the design of the mechanism for choosing which experts to evaluate and trade-offs in network architecture to maintain computational efficiency. These design choices are tailored for recurrent models and cannot be directly applied to state-of-the-art image classification models which are feed-forward convolutional networks.

**Hierarchical classification.** Categories in ImageNet [6, 25] are organized into a semantic hierarchy using an external knowledge base. The hierarchy can be used to first predict the super class and only perform fine grained classification within the super class [5, 7]. HDCNN [30] is a hierarchical image classification architecture which is similar in spirit to our approach. HDCNN and Ioannou *et al*. [20, 17] improve accuracy with significant increase in cost relative to the baseline architecture. In contrast, HydraNets on ImageNet improve top-1 accuracy by 1.18-2.5% with the same inference cost as corresponding baseline architectures.

Both HDCNN and Ioannou *et al*. model the routing weights as continuous variables which are used to linearly combine outputs from multiple experts. Jointly learning the routing weights with the experts is similar to LEARN in Table 8, and performs poorly due to optimization difficulties (collapse and poor utilization of experts). HDCNN uses complex multi-stage training to mitigate optimization issues and provides robustness to routing error by overlapping the classes handled by each expert. HydraNets use binary weights for experts during both training and inference by dropping out all but the top-$k$ experts. This enables joint training of all the HydraNet components while allowing flexible usage of experts.

Architectural structures similar to HydraNet [1] have been used for learning the partition of categories into disjoint subsets. Our main contribution is a gating mechanism which reduces inference cost by dynamically choosing components of the network to evaluate at runtime. Recent work [22, 23] has explored directly incorporating inference cost in the optimization and explore training methods for jointly learning the routing and the network features. In contrast to the complex training regime required for joint learning, our approach enables a simple and effective training strategy which we comprehensively evaluate cost on both ImageNet and CIFAR-100 datasets.

## 3. HydraNet Architecture Template

The HydraNet template, shown in Figure 1, has four major components.

- *Branches* which are specialized for computing features on visually similar classes. We view computing features relevant to a subset of the network inputs as a *subtask* of the larger classification task.
- A *stem* that computes features used by all branches and in deciding which subtasks to perform for an input.
- The *gating* mechanism which decides what branches to execute at inference by using features from the stem.

- A *combiner* which aggregates features from multiple branches to make final predictions.

Realizing the HydraNet template requires partitioning the classes into visually similar groups that the branches specialize for, an accurate and cost-effective gating mechanism for choosing branches to execute given an input, and a method for training all the components. The following sections describe how we address these key questions.

### 3.1. Subtask Partitioning

To create a HydraNet with $n_b$ branches, we partition the classes into $n_b$ groups of equal size. Similar to hierarchical classification we group visually similar classes so that branches can specialize in discriminating among these classes. While it might be possible to manually create groups when the number of classes is small, for large classification problems we need a mechanism for automatically creating visually similar groups. We compute a feature representation for each class by averaging the features from the final fully connected layer of an image classification network for several training images of the same class. Clustering these average class features using k-means with $n_b$ cluster centers results in a partitioning of the feature space. Directly assigning each class to its nearest cluster center leads to an imbalanced class partitioning. Instead, each cluster center is assigned the class that is nearest to it and this process is repeated for $\frac{C}{n_b}$ steps, where $C$ is the total number of classes, resulting in a balanced partitioning. Each of the $n_b$ class partitions is assigned to one of the branches, which we refer to as a subtask.

### 3.2. Cost Effective Gating

Given a subtask partitioning, traditional hierarchical classification uses the gating function to classify among the subtasks and the branches for classifying among the classes within a subtask. For dynamic execution to be accurate, both the gating function and the chosen branch need to perform their respective classification tasks accurately. In practice, the accuracy of the gating function depends on the capacity (computational cost and parameters) of the gating component. Making the gating function highly accurate incurs significant computational cost negating the benefits of dynamic execution.

Our key insight in making the gating both computationally efficient and accurate is to change the function of both the subtasks and the gating function. In a HydraNet architecture, the branches do not perform final classification, instead they *only compute features relevant to the subtask assigned to the branch*. For example, a branch corresponding to a cluster of bird classes might compute features that distinguish between the different species of birds. Since the branches compute features instead of final predictions, executing more branches translates to computing more features that can be combined to make final predictions. Therefore, the job of the gating function is to choose which features (which branches) to compute. Since the gating function only needs to narrow down the final classification problem to determining which $k$ subtasks to compute (rather than a single precise subtask), a lower capacity gating function is sufficient. Note that in the HydraNet template the *stem* computation is shared across different *branches* making the execution of $k$ branches computationally efficient by design (Section 4.2).

We denote gating function score for an input $I$ by $g(I) \in [0,1]^{n_b}$, the output of branch $b$ by $branch(b)$, and indices of the top-$k$ branches by $topk(g(I))$. The combined output of the branches $comb$ is given by:

$$comb(I) = \sum_{b \in topk(g(I))} branch(b) \qquad (1)$$

We find that combining the features from the branches by concatenation works equally well in practice, but requires more memory and parameters than linear combination.

### 3.3. Training

Given a subtask partitioning, the stem, the branches, and the gating function of the HydraNet architecture are trained jointly. While direct supervision for branch outputs is not available, the labels for the gating function are given by the class to branch mapping determined by the subtask partitioning. The gating function and the downstream combiner indirectly provide supervision for the features that each branch needs to compute. Since the mapping of the class clusters to branch is fixed and determined by subtask partitioning, consistently using the features from the same branch for a subtask drives the features to specialize for the subtask. This approach is similar to the modular network approach for visual question answering [2, 18], where the questions define the set of modules to execute and a consistent mapping from questions to the modules encourages approximately learning the function designated to each module. The branches are indirectly supervised by the classification predictions after combining the features computed by the top-$k$ branches.

Both the gating function and the branches are supervised using ground truth labels for image classification. During training, the top-$k$ branches chosen for different inputs in the mini-batch can be different. We evaluate all branches on all inputs and mask out features from branches not picked by the gating function. In effect, the branch outputs from the branches other than the top-$k$ for each input are ignored and not seen by the combiner. Masking ensures that weights of the branches not chosen by the gating remain unchanged during back propagation. This can also be view as an adaptive form of drop out [3] where entire paths are dropped instead of units within a layer. We use the cross entropy loss for both the gating function and the final predictions.

| Model | Configuration $d$ | $w$ | Params ($\times 10^6$) | MADDs ($\times 10^6$) | Accuracy (Top-1) |
|---|---|---|---|---|---|
| ResSep-A | 2 | 0.50 | 1.96 | 181 | 61.88 |
| ResSep-B | 3 | 0.50 | 2.68 | 290 | 65.27 |
| ResSep-C | 2 | 0.75 | 3.98 | 380 | 67.16 |
| ResSep-D | 3 | 0.75 | 5.58 | 620 | 69.90 |
| ResSep-E | 3 | 1.00 | 9.53 | 1060 | 72.02 |
| ResNet-18 | - | | 11.69 | 1800 | 69.30 |
| MobileNet | - | | 4.2 | 569 | 70.60 |

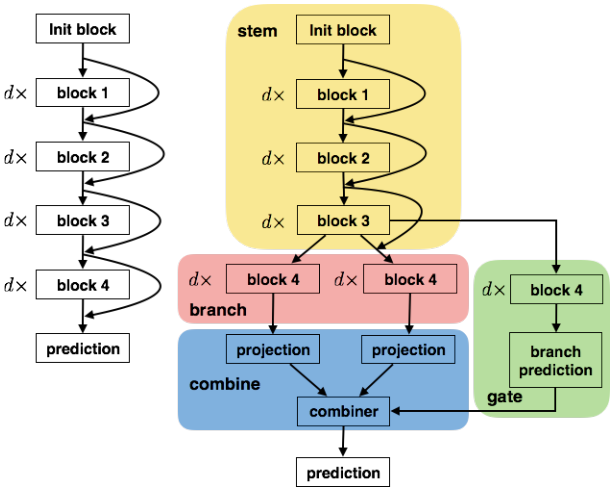
Table 1: Number of multiply-add operations and Top-1 ImageNet classification accuracy of variants of ResSep, ResNet-18, and MobileNet architectures for image classification. ResSep-D model is more accurate than Resnet-18, and 2.5× less expensive.

## 4. Architecture for ImageNet Classification

To analyze the computational cost and accuracy trade-offs enabled by HydraNet designs, we need a baseline architecture that is both computationally efficient at inference like MobileNet [13] and allows fast training like ResNet [12] in order to rapidly train multiple models in the design space. In response we created a static architecture, called ResSep which incorporates desirable architectural choices from both the MobileNet and the ResNet architectures, namely depth wise separable convolutions and residual connections.

### 4.1. Baseline Architecture

The high-level structure of the ResSep architecture (Figure 2) is similar to the ResNet architecture for ImageNet classification. There are four blocks (excluding the initial block) that can be stacked multiple times to create deep networks. Residual connections are added between the input and output of each block to accelerate training and provide regularization. Unlike ResNet, instead of using convolution layer for each of the blocks we use depth-wise separable convolutions, which have been shown to be effective at reducing computation costs with only minor reduction in accuracy [13]. The configuration of layers comprising each block in ResSep is shown in Figure 2. The amount of computation and number of parameters can be varied by scaling the number of filters in each block by the parameter $w$ and the depth of the network by the stacking factor $d$. Table 1 compares the computation cost, number of parameters and Top-1 ImageNet classification accuracy of several variants of the baseline ResSep architecture to popular image classification architectures. The ResSep-D model is more accurate than Resnet-18, and is 2.5× less expensive; it also compares favorably to the MobileNet architecture which is highly tuned for low cost inference on mobile devices.




(a) Transforming the baseline ResSep architecture for ImageNet classification into a HydraNet architecture. Left: ResSep with separable convolutions in each block. The hyper parameter $d$ is number of times each block is stacked. Residual connections are added from each block's input to its output. Right: the corresponding HydraNet architecture with branches, gate, and combiner components.

| Name | Layers | Stride | Channels |
|---|---|---|---|
| Init block | conv 3 × 3 | 2 | 64 |
| | max pool 3 × 3 | 2 | 64 |
| Block 1 | sep conv 3 × 3 | [1, 2] | 128 ×$w$ |
| | sep conv 3 × 3 | 1 | 128 ×$w$ |
| Block 2 | sep conv 3 × 3 | [1, 2] | 256 ×$w$ |
| | sep conv 3 × 3 | 1 | 256 ×$w$ |
| Block 3 | sep conv 3 × 3 | [1, 2] | 512 ×$w$ |
| | sep conv 3 × 3 | 1 | 512 ×$w$ |
| Block 4 | sep conv 3 × 3 | [1, 2] | 1024 ×$w$ |
| | sep conv 3 × 3 | 1 | 1024 ×$w$ |
| Projection | conv 1 × 1 | 1 | 1024 ×$w$ |
| Combiner | add | | 1024 ×$w$ |
| | sep conv 3 × 3 | 1 | 1024 ×$w$ |
| Prediction | avg pool 7 × 7 | 1 | 1024 ×$w$ |
| | fully connected | | 1000 |

(b) Layer configuration for each of the block in the ResSep architecture. The parameter $w$ controls the number of filters in each block. The first convolution layer in the first instance of each blocks uses stride 2 and subsequent instances use stride 1.

Figure 2: ResSep and the corresponding HydraNet Architecture for ImageNet.

### 4.2. Design Space for HydraNet Transformation

Transforming a baseline static architecture into a dynamic HydraNet architecture involves several design choices for each of the components. We describe these choices in the context of ResSep, but the design space trans-

forming ResNet [12] and DenseNet [15] architectures for CIFAR-100 classification is similar (Section 5.1).

The first of these choices is determining the architecture of the *stem* and the *branches*. Having a deeper *stem* allows sharing higher level features across the different branches or subtasks and allows features from deeper layers to be used to make more accurate gating decisions. On the other hand, making the gating decision after a deep *stem* diminishes the potential savings of dynamic execution. Therefore, deciding where to branch is crucial for retaining both computational efficiency and accuracy. We empirically observe that partitioning the first three ResSep blocks into the *stem* and replicating the fourth block in each of the *branches* gives better accuracy per unit computation cost.

In designing the branches, directly replicating ResSep Block 4 into branches results in an inordinate increase in the number of model parameters and the number of floating point operations (by 3-4$\times$ for $n_b = 10$) during training, even for a small number of branches. Since training on ImageNet is computationally expensive, we retain the structure of Block 4 but scale the number of filters in each convolution layer by $w_b$ to reduce training time. Balancing the computation in the *branches*, *stem*, and the number of branches is essential for improving computational efficiency of inference. The hyper parameters that govern this balance are listed in Table 2: $n_b$ controls the number of branches, $w_s$ and $w_b$ control the number of filters in the layers in the *stem* and *branches* respectively. When transforming architectures for smaller datasets (CIFAR-100) where training is relatively cheap we do not scale down the number of filters.

The architecture of the gating function is the same as that of a branch, but followed by a prediction block. The gating function predicts the top-$k$ branches relevant for the given input. Features from these branches are combined by first projecting them from a $w_b$-dimensional space into a $w_s$-dimensional space and adding them together. The *combiner* block uses the $w_s$ features to make final class predictions. The hyper parameters $w_g$ and $k$ (Table 2) control the cost of the gating function and the number of branches dynamically evaluated for each input. We further elaborate on how these hyper parameter choices impact gating accuracy in Section 5.2.

## 5. Experiments

In this section, we demonstrate the effectiveness of HydraNet architectures by evaluating the computational cost vs. accuracy for image classification using both ImageNet and CIFAR-100 datasets. We use the total number of multiply-add (MADD) operation performed by the network during inference as measure of cost. Although the number of MADDs does not directly translate to inference runtime, it is indicative of the runtime and allows for comparing network cost independent of hardware and associated software implementation details.

| Parameter | Description |
|---|---|
| $d$ | Number of times each block is stacked. Controls overall network depth. |
| $n_b$ | Total number of branches |
| $k$ | Number of branches selected by the gating mechanism for an input image |
| $w_s, w_b, w_g$ | Multipliers for controlling number of filters in layers of stem, branches, and gating components respectively |

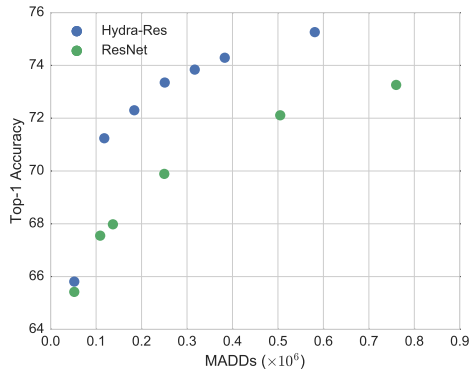
Table 2: Hyper parameters for exploring various trade-offs in the design of HydraNet architectures.

We first analyze the computational cost and accuracy of HydraNet architectures based on two state-of-the-art architectures on CIFAR-100. We then use the ResSep architecture, which is designed to be computationally efficient for both training and inference, on the ImageNet dataset for evaluating the design choices in each component of the HydraNet template.

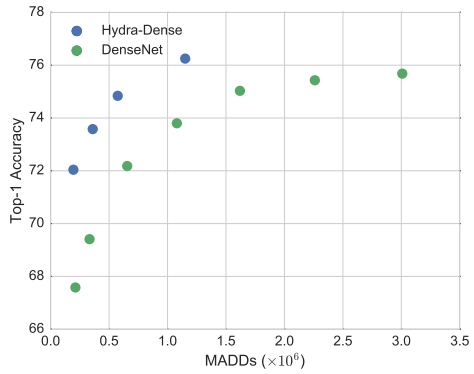
### 5.1. CIFAR

We use the HydraNet template to transform ResNet (plain residual units) and DenseNet architectures for CIFAR-100 classification into Hydra-Res and Hydra-Dense architectures. The overall structure of the Hydra-Res and Hydra-Dense models is similar to that of the Hydra-ResSep architecture for ImageNet, however both ResNet and DenseNet for CIFAR-100 classification architectures have three blocks instead of four. We replicate the third block in each of the $n_b = 20$ branches and dynamically run $k = 4$ branches for each input. Unlike the Hydra-ResSep model for ImageNet, the number of filters in the stem, branches and gating function are not scaled down ($w_b, w_s, w_g$=1) from the design of the static baseline architecture. We create multiple variants of both Hydra-ResNet and Hydra-DenseNet by varying the depth (which is encoded in the variant name in Table 3). The hyper parameters $k, n_b$ and the point to branch were chosen empirically by searching over $n_b = [5, 10, 20, 25]$, $k = [3, 4, 5]$ and branching at block 2 or block 3. The hyper parameter sweep was done using the smallest HydraNet architecture, Hydra-Res-d1 and Hydra-Dense-d2 in Table 3 and reused for the more expensive variants. We found that increasing the total number of branches ($n_b$) and dynamic branches ($k$) beyond 20 and 4 respectively had diminishing returns, while branching at block 2 delivered the better cost-accuracy trade-off.

The cost (in MADDs), number of model parameters and accuracy of variants of both Hydra-Dense and Hydra-Res are shown in Table 3. Both the Hydra-Res and Hydra-




(a) ResNet architectures.




(b) DenseNet architectures.

Figure 3: Inference cost vs accuracy of HydraNet architectures for CIFAR-100 classification. HydraNet architectures have the same accuracy as the baseline ResNet and DenseNet architecture at lower dynamic cost.

Dense architectures give the same level of accuracy as the static baseline architectures with significantly lower computational cost. For example, Hydra-Res-d4 yields the same accuracy as ResNet-164 while incurring one third the cost. As seen in Figure 3, similar trends hold for several variants of the HydraNet architectures. The improved accuracy of the HydraNet models is due to the overall increased capacity of the model, i.e., the total number of parameters and MADDs. However, at inference time the model dynamically picks a small subset of branches to execute; reducing the inference cost dramatically.

## 5.2. ImageNet

As with CIFAR, we trained multiple HydraNets to explore the accuracy vs. cost landscape. However, due to the high cost of training large models on ImageNet classification we focus on a single baseline family of architectures (ResSep from Section 4.2). The ResSep-D model takes $\sim$ 12 hours to train with distributed asynchronous training using 32 GPUs. Relative to ResSep-D, the Hydra-ResSep-D architecture shown in Table 4 takes $\sim 2.5\times$ longer to train. Training time and GPU memory make it impractical to explore large values for $n_b$ and $w_b$. For example, setting $w_b = 0.5$ instead of $w_b = 0.125$ in the Hydra-ResSep-D architecture makes the training time $\sim 4\times$ longer. The particular configurations in Table 4 were chosen to have similar inference cost as the baseline ResSep architectures (the hyper parameters $d, w_s$ directly correspond to the baseline architectures and $n_b, k, w_g$ are balanced such that the dynamic inference cost closely matches the corresponding baseline). We further elaborate on the choice of hyper parameters in the rest of the section with associated experiments.

| Model | Params ($\times 10^6$) | Inference MADDs ($\times 10^6$) | Accuracy (Top-1) |
|---|---|---|---|
| Hydra-Res-d1 | 1.28 | 52 | 65.81 |
| Hydra-Res-d2 | 2.86 | 118 | 71.24 |
| Hydra-Res-d3 | 4.43 | 184 | 72.30 |
| Hydra-Res-d4 | 6.01 | 251 | 73.35 |
| Hydra-Res-d5 | 7.59 | 317 | 73.84 |
| Hydra-Res-d6 | 9.17 | 383 | 74.29 |
| Hydra-Res-d7 | 10.74 | 449 | 74.71 |
| Hydra-Res-d9 | 13.90 | 581 | 75.26 |
| ResNet-14 | 0.17 | 52 | 65.42 |
| ResNet-20 | 0.27 | 81 | 66.18 |
| ResNet-26 | 0.37 | 109 | 67.55 |
| ResNet-32 | 0.47 | 137 | 67.98 |
| ResNet-56 | 0.85 | 250 | 69.89 |
| ResNet-110 | 1.73 | 505 | 72.11 |
| ResNet-164 | 2.60 | 760 | 73.26 |
| Hydra-Dense-d2 | 2.77 | 195 | 72.04 |
| Hydra-Dense-d3 | 5.43 | 360 | 73.58 |
| Hydra-Dense-d4 | 8.97 | 574 | 74.84 |
| Hydra-Dense-d6 | 18.72 | 1151 | 76.25 |
| DenseNet-d3 | 0.36 | 212 | 67.58 |
| DenseNet-d4 | 0.58 | 333 | 69.41 |
| DenseNet-d6 | 1.19 | 655 | 72.18 |
| DenseNet-d8 | 2.00 | 1080 | 73.80 |
| DenseNet-d10 | 3.03 | 1619 | 75.03 |
| DenseNet-d12 | 4.27 | 2260 | 75.43 |
| DenseNet-d14 | 5.72 | 3008 | 75.68 |

Table 3: Computation cost and CIFAR Top-1 accuracy of several variants of HydraNet architectures and baseline architectures. The HydraNet architectures have the same accuracy as the baseline networks with much lower dynamic cost. All the HydraNet architectures have $n_b = 20$ branches and use $k = 4$ branches during inference.

**HydraNet architecture improves accuracy while having similar computation costs as the baseline ResSep architecture.** Figure 4 shows the cost vs. accuracy of both Hydra-ResSep and the baseline ResSep architectures. As shown in Table 4, the HydraNet architectures are more ac-

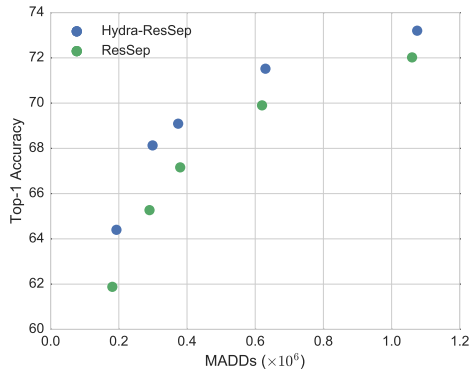



Figure 4: Cost vs. accuracy trend for Hydra-ResSep and the baseline ResSep architectures.

curate, and have more parameters compared to the baseline ResSep architectures with similar inference cost, especially in the low-cost regime. However, at inference time only $k = 5$ of the $n_b = 50$ branches are evaluated, making the inference cost comparable to the corresponding baseline architecture. The accuracy improvements of HydraNets in the low cost regime are more pronounced. We believe this is because of the increase in total capacity (parameters and total static MADDs) of the HydraNet architectures relative to the baseline is larger in this regime (HydraNet-ResSep-A has 5.6× more parameters and 3.3× more MADDs). In the high accuracy regime, the accuracy improves by 1.18% relative to the baseline (HydraNet-ResSep-D has 2.8× more parameters and 1.5× more multiply-adds).

**Increasing the total number of branches results in better accuracy and increases training costs.** In order to analyze the impact of the number of branches ($n_b$) we vary $n_b$ while keeping the other hyper parameters the same as HydraNet-ResSep-A (Figure 4). Increasing the number of branches increases the total capacity of the network and allows for more specialization, but makes learning the subtask gating function harder. However, since we rely on a soft top-k gating mechanism, increasing the total number of branches results in a net improvement in overall accuracy as shown in Table 5. Note that increasing the total number of branches increases the training cost, but not inference cost. Training cost and available GPU memory make it impractical to scale $n_b$ and $w_b$ to large values. We set $n_b = 50$ and $w_b = 0.125$ to keep the training time under $\sim$ 2 days (asynchronous training with 32 GPUs) for all the variants of HydraNet-ResSep.

**Executing a small number of branches dynamically improves accuracy significantly relative to executing a single branch.** Table 6 shows the importance of the soft-gating mechanism by varying $k$ while keeping other hyper parameters the same. Dynamically executing only a single branch results in lower accuracy than the ResSep-A model. This is due to inaccuracy in gating as well as the low capacity of a single branch. However, executing a small number of branches recovers accuracy and outperforms the ResSep-A model by 2.5%. There are diminishing improvements in accuracy with increasing $k$. We set $k = 5$ for all the HydraNet-ResSep architectures.

**Branching earlier in the network hurts accuracy with only a slight reduction in dynamic compute cost.** An important aspect of transforming a baseline architecture using the HydraNet template is deciding where the model is partitioned into branches. Table 7 shows the accuracy when the baseline architecture ResSep-A is transformed into a HydraNet architecture by branching at Block 2 (each branch and the gating function now has both Block 3 and 4 with the number of filters scaled by $w_b$ and $w_g$ respectively). Branching earlier in the network increases the overall training cost and results in significantly lower accuracy. We believe this is due to inaccurate decisions by the gating function when relying on low-level features. In principle, one could empirically search for the best split point at the layer granularity rather than block granularity. We restricted our experiments to block granularity to limit the time spent on this search.

**Clustering based subtask partitioning works well in practice.** We evaluate the effectiveness of clustering based subtask partitioning by comparing with several baselines. Table 8 shows different methods for partitioning and their corresponding accuracy. CLUSTER is the $k$-means based subtask partitioning described in Section 3.1. RANDOM partitions the classes into equally sized subsets randomly. NO GATING does not require subtask partitioning since all the branches are used both at training and inference, and serves as an upper bound on the accuracy that can be achieved since any gating function will utilize a subset of the branches per input. CLUSTER is 1.33% more accurate than RANDOM demonstrating that semantically partitioning the classes allows the branches to specialize more effectively. The accuracy gap between CLUSTER and NO GATE is 1.8%, which is narrow given that CLUSTER only uses five branches out of the total 50 per input.

We also explored alternative methods to CLUSTER which learn the subtask partitioning jointly with the training of the branches. Instead of supervising the gating function with a branch classification loss, LEARN uses the scores generated by the gating function as weights for linearly combining output features of the top-$k$ branches. The gating function learns to weigh the relevant branches higher for a given input based on the overall classification loss. Table 8 shows that LEARN gives lower accuracy than the baseline. We observe the gating function tends to favor branches which initially get more training signal. This results in a self-reinforcing loop: branches which are not picked do

| Model | Branch Point | Configuration | | | | | | Params ($\times 10^6$) | Inference MADDs ($\times 10^6$) | Accuracy (Top-1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $d$ | $w_s$ | $w_b$ | $w_g$ | $k$ | $n_b$ | | | |
| Hydra-ResSep-A | Block 3 | 2 | 0.50 | 0.125 | 0.125 | 5 | 50 | 10.89 | 193 | 64.40 |
| Hydra-ResSep-B | Block 3 | 3 | 0.50 | 0.125 | 0.250 | 5 | 50 | 13.30 | 299 | 68.13 |
| Hydra-ResSep-C | Block 3 | 2 | 0.75 | 0.125 | 0.250 | 5 | 50 | 15.70 | 374 | 69.09 |
| Hydra-ResSep-D | Block 3 | 3 | 0.75 | 0.125 | 0.500 | 5 | 50 | 19.51 | 630 | 71.52 |
| Hydra-ResSep-E | Block 3 | 3 | 1.00 | 0.125 | 0.500 | 5 | 50 | 26.83 | 1075 | 73.20 |

Table 4: Computation cost, hyper parameters, and Top-1 accuracy several variants of the Hydra-ResSep architecture for ImageNet classification. All the architectures are transformed from corresponding baseline ResSep architecture.

| Total branches | $n_b = 10$ | $n_b = 25$ | $n_b = 50$ |
|---|---|---|---|
| Accuracy (Top-1) | 61.76 | 63.21 | 64.40 |
| Params ($\times 10^6$) | 3.30 | 6.14 | 10.89 |
| Total MADDs ($\times 10^6$) | 239 | 376 | 605 |

Table 5: Increasing number of branches ($n_b$) gives better accuracy but increases training cost and the number of model parameters. Total MADDs is the cost of all branches which is proportional to training time. Other hyper parameters are the same as Hydra-ResSep-A in Table 4.

| Dynamic branches | $k = 1$ | $k = 3$ | $k = 5$ | $k = 10$ |
|---|---|---|---|---|
| Accuracy | 60.08 | 63.38 | 64.40 | 64.89 |
| Inference MADDs ($\times 10^6$) | 157 | 175 | 193 | 239 |

Table 6: Increasing the number of dynamically executed branches ($k$) of HydraNet-ResSep-A results in better accuracy with diminishing returns.

| Split point | Params | MADDs ($\times 10^6$) | | Accuracy |
|---|---|---|---|---|
| | | Total | Cond | |
| Block 2 | 9.66 | 695 | 174 | 61.35 |
| Block 3 | 9.66 | 605 | 193 | 64.40 |

Table 7: Branching at Block 2 of ResSep-A gives lower accuracy compared to branching at Block 3. All other hyper parameters are the same as Hydra-ResSep-A in Table 4.

| Method | Accuracy |
|---|---|
| RANDOM | 63.07 |
| LEARN | 61.50 |
| LEARN-BALANCE | 63.17 |
| CLUSTER | 64.40 |
| NO GATING | 66.21 |

Table 8: Accuracy of HydraNet-ResSep-A with different subtask partitioning and gating strategies. Other hyper parameters are the same as in Table 4 except for *no gating* which uses outputs of all the branches.

not learn to compute useful features and the gating function learns to ignore these branches. Prior work in language modeling has overcome the under utilization problem by adding a loss term encouraging equal utilization of branches for a batch of inputs [26]. Given a batch size of $N$ and the gating function $g(\text{batch\_id}, \text{branch\_id})$, the utilization of the branch $b$ across the batch is given by $\sum_{i=0}^{N} g(i, b)$. We encourage equal utilization of branches by adding an additional loss term $\lambda_{bal} \sum_{b=0}^{n_b} (\sum_{i=0}^{N} g(i, b))^2$, this corresponds to the LEARN-BALANCE row in the table. LEARN-BALANCE does significantly better than LEARN but still falls short of CLUSTER.

## 6. Discussion

The HydraNet architecture reduces computational cost by specializing components of a network for subtasks and exploiting this specialization at inference time. Although, the paper focuses on classification we believe that sparse dynamic execution will be increasingly important for building multi-task models which perform a wide range of tasks using a shared representation. Inference on high resolution images and video would need to exploit dynamic execution to keep up with the demand for computational efficiency.

Although HydraNet architectures enable more work-efficient inference than static architectures, this efficiency comes at the cost of longer training times due to the network's many branches. One approach to reducing the training time is to leverage dynamic execution in the training process. Unlike inference, training is sensitive to batch size and efficient dynamic execution in small batch settings is challenging since each item in the batch may need to execute a different set of branches. Current frameworks are not designed for efficient execution in these scenarios. However, dynamic training approaches have been used for training large scale language models [26] and adapting them to image models would be an interesting avenue to explore.

***Acknowledgements*** The authors would like to thank Deva Ramanan, Ashish Vaswani, Andrew Howard, Mark Sandler and Mohammad Norouzi for their valuable feedback. This research was supported by the National Science Foundation (awards IIS-1422767 and IIS-1539069), the Intel Science and Technology Center for Visual Cloud Computing, and a Google Faculty Fellowship.

## References


[1] K. Ahmed, M. H. Baig, and L. Torresani. Network of experts for large-scale image categorization. In *European Conference on Computer Vision*, pages 516–532. Springer, 2016. 2

[2] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 3

[3] J. Ba and B. Frey. Adaptive dropout for training deep neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3084–3092. Curran Associates, Inc., 2013. 3

[4] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. 2

[5] J. Deng, A. C. Berg, and L. Fei-Fei. Hierarchical semantic indexing for large scale image retrieval. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 785–792, Washington, DC, USA, 2011. IEEE Computer Society. 2

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 2

[7] J. Deng, S. Satheesh, A. C. Berg, and F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 567–575. Curran Associates, Inc., 2011. 2

[8] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. *CoRR*, abs/1612.02297, 2016. 2

[9] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44(3):243–254, June 2016. 2

[10] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally. DSD: regularizing deep neural networks with dense-sparse-dense training flow. *CoRR*, abs/1607.04381, 2016. 2

[11] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015. 2

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 4, 5

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 4

[14] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR*, abs/1703.09844, 2017. 2

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 5

[16] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. 2

[17] Y. Ioannou, D. P. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. *CoRR*, abs/1603.01250, 2016. 2

[18] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Inferring and executing programs for visual reasoning. *arXiv preprint arXiv:1705.03633*, 2017. 3

[19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM. 1

[20] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo. Deep neural decision forests. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015. 2

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

[22] L. Liu and J. Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *CoRR*, abs/1701.00299, 2017. 2

[23] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372, 2017. 2

[24] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015. 2

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual

Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2

[26] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017. 2, 8

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2

[28] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001. 2

[29] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2074–2082. Curran Associates, Inc., 2016. 2

[30] Z. Yan, V. Jagadeesh, D. DeCoste, W. Di, and R. Piramuthu. HD-CNN: hierarchical deep convolutional neural network for image classification. *CoRR*, abs/1410.0736, 2014. 2