Selectively Deep Neural Networks at Runtime

Anonymous CVPR submission

Paper ID 3873

Abstract

Deep networks are powerful tools for computer vision tasks, but they incur significant computational costs, even at test time. In this work, we present an approach for optimizing existing deep networks to enable faster running times with minimal loss of accuracy. In contrast with approaches such as model compression, we do not create a new network, but instead perform minimal modifications to the original network. Our approach is to learn “gatekeepers” at each layer of a deep network that determine whether to continue further in the network or to early exit, on a per-sample basis. The key advantage of our approach is that it is a simple modification that can be applied to any deep network and it does not require configuring and training a new network from the ground up as is the case with model compression approaches. Our approach allows for a very direct control of the trade-off between accuracy and running time, which is crucial for real deployments of deep networks in computationally constrained environments such as mobile devices. We demonstrate our approach on four deep network applications, which includes a state of the art predictor for measuring image interestingness.

1. Introduction

In recent years, deep learning has emerged as an incredibly powerful tool for a large number of computer vision and machine learning applications. It has revolutionized areas such as speech and image recognition and is being applied to an increasingly broad range of applications outside of these areas. While deep learning methods have led to large increases in accuracy, these advancements have come at high computation and resource costs, and these costs are only increasing as models get larger and deeper, e.g., one recent publication showed over one hundred layers [12]. As a result, it is quite challenging to use such systems in any sort of a resource-constrained environment. In this work, our goal is to address this problem by optimizing existing deep networks to enable faster running times with minimal loss in accuracy. Inspired by recent work in model compression [1], we seek to optimize an existing neural network; however, in contrast with model compression we do not create a new, smaller “mimic” network to simulate the original, instead we make minimal modifications to the original network. Our approach is based on the well-known programming concept of an “early-exit.” As illustrated in Figure 1, we add learned “gating” functions that decide on a per-image basis whether to continue further in the network or to early-exit and make a prediction without feeding through the full neural network. By propagating through fewer layers of the network, the average running time decreases. We call this modified network a “selectively deep” network, as it is conceptually analogous to selecting between parallel networks of different depths in a content and an application dependent way.

One of the main insights that drives this work is that the prediction performance on certain images may not benefit from additional network depth, as not all images are equally challenging. For example, it is not hard to predict the perceptual image quality for an extremely blurry or a constant colored image, thus the benefit of additional layers for these images is minimal. Our approach attempts to take advan-
tage of such inefficiencies in deep networks.

The key advantage of our approach is that we use much simpler modifications of a network than the current body of work on model compression, as we do not require configuring and training a new network from the ground up [1]. Instead we use lower complexity and well-understood tools such as Support-Vector Machines and Random Decision Forests for modifying the network. In addition, our approach allows for a very direct control of the trade-off between accuracy and running time, something that is difficult with existing approaches yet is crucial for deployments of deep networks in real-world applications.

We demonstrate our approach on four different vision tasks that leverage the power of deep networks to show the applicability and the performance benefits of the approach, which includes using our approach to create a state-of-the-art image interestingness measure [9, 10].

2. Related Work

Deep learning in Computer Vision: Deep learning has had a transformational affect on a number of computer vision applications, such as image recognition [15]. Recent work has shown that network depth and complexity can be of critical importance for achieving state of the art results [17, 20]. As a result, the trend has been to create deeper and more complex networks from eight [15], to sixteen [17], to hundreds [12] of layers. Due to this trend there has recently been an increased focus on optimizing networks to allow for faster training and testing. The approaches can be roughly broken into two categories: 1) model compression, where one creates and trains a new smaller “mimic” network to simulate the original, and 2) shortcut connections and budgeted algorithms, where it is allowed for part of a network or a set of feature detectors to be bypassed.

Model Compression: These approaches attempt to compress existing networks by applying data compression methods such as quantization, Huffman encoding, and hashing to replace parts of a deep network [11, 4], or by replacing the entire network with a smaller version that is trained to mimic the original network [2, 1]. Our work shares similar goals, as we seek to optimize existing networks; however our approach is simpler and less time consuming than these approaches, as we do not require training of a new neural network, which can be very time consuming.

Instead we use faster, lower complexity approaches such Support-Vector Machines and Random Decision Forests that require less training data and execution time. For example, the work of Bu et al. used one million images [1] and hours for training, while we use less than a thousand images and only minutes for training.

Shortcut Connections: Our work is most structurally similar to approaches that use shortcut connections and gating functions [18, 19, 12]. The primary difference with our work is that these approaches use gating functions to speed up training and/or to enable deeper networks to be trained, while our goal is speed up testing, and we do not perform any training of the original network.

Pipeline Sparsification and Bypassing: Our work is inspired by methods that consider computational budgets to ensure timely results for recognition [21, 13, 14]. We share similar goals of maximizing the accuracy vs timeliness trade-off. The difference in our work is that instead of bypassing distinct feature detectors in a static way, we bypass hidden layers of a neural network in an (input) image dependent way. Additional work has also looked at sparsifying network layer weights with the goal of using reducing memory cost during test and training time [5]. We focus on test time cost and do not address memory cost.

3. Selectively Deep Framework

Deep neural networks have computational costs that are dependent on the depth and complexity of the networks. Figure 2 illustrates this dependence, where we plot the average computation time for feeding an image through AlexNet [14]. These feature extraction times are the averages computed for 100 random images and are obtained from the Caffe framework, without using a GPU. We can observe that different layers have different computational complexities leading to the jagged nature of the curve, yet the general trend is clear; more hidden nodes per layer and more depth leads to an increased computation time. As mentioned earlier, the main insight that drives this work is that not all images are equally challenging for all applications. For example, consider the task of face detection on an image that consists only the sky in the background. This image should require very different computational processing time when compared to a different image capturing the clutter of the urban world in the background.
Our approach aims to exploit this observation that many images may not require the power of the full network depth, and consequently provide faster runtime for such images by selectively adjusting computation. Below we discuss our methodology for modifying existing neural networks so that we can easily consider the speed vs. accuracy trade-off at the test time.

Figure 1 provides a graphical illustration of our framework. Specifically, we add gating functions that decide on a per-image basis whether to continue in the network or to early-exit and make a prediction without feeding through the full network. We reiterate that our core assumption is that many of the test images do not need the propagation through the entire network, and by limiting the computation to the first few layers, we can hope to reduce the running time significantly.

Essentially, given a test image, the key idea is to determine at every layer if it is worth continuing with the propagation to the following layers. Such determination is based on the promise of further enhancing the prediction performance. In particular, the gate keepers at every layer determine if it is worth continuing to a deeper layer or whether an early exit will provide adequate performance. Once exited, we then have task-specific predictors whose job is to utilize the activations of the current layer and use those to make the predictions of interest. Thus, the two key additions to the neural network architecture are the gate keepers and the task-specific predictors. We now describe these in detail.

3.1. Training Task-Specific Predictors at Every Layer

One of the key requirements for an early exit is that the task-specific predictor can still be accurate based on computation that has happened so far. Formally, when handling a test instance \( x \), if we decide to halt the computation at layer \( l \), then we need to assure that predictions can still be made based on the features \( \phi_{l-1}(x) \) for \( j = \{1, 2, ..., l\} \), computed so far at the layers \( l \) and lower. We achieve this by training task-specific predictors, denoted as \( F_l(x) \), for every layer \( l \). Specifically, we train the task-specific predictor using the provided training data, albeit with the features \( \phi_l(x) \) as inputs (as opposed to \( x \) as inputs). These task-specific predictors are trained offline with the rest of the neural network and at inference time can quickly provide the required predictions when the gate keepers signal an early exit.

3.2. Training the Gate Keepers

The gate keepers are the crucial decision making entities, associated with layers in a deep neural network, that determine whether to terminate computation or keep going further, for a given test instance. Given that the gate keepers are introduced in this framework to reduce computation, it is fairly important that these decisions are made both efficiently and accurately.

We propose to model these gate keepers as binary classifiers implemented as linear support vector machines (SVM) or Random Decision Forests (RDF), thus the computational overhead by introducing such gate keepers at test time is minimal. The more challenging aspect, however, is training these gate keepers since there is no explicit training data labeled for such binary decisions. Consequently, we derive a bootstrapping procedure in order to generate the training data for the gate keepers.

Formally, let us consider a layer \( l \) for which we aim to train a gate keeper \( G_l \). The function of this gate keeper will be to determine that given an instance \( x \), whether the test time inference process should continue to higher layers (\( G_l(x) < 0 \)) or terminate at this layer itself (\( G_l(x) \geq 0 \)). First, we use \( \phi_l(x) \), the features extracted at the layer \( l \) itself as input to the gate keeper. Next, we derive the labeled training dataset via a \( k \)-fold training methodology. Specifically we follow the following steps:

1. Divide the training dataset into \( k \) folds.
2. Train the task specific learners on \( k - 1 \) folds and consider the remaining fold as the validation set.
3. For the layer \( l \), the label corresponding to the data point \( x \) in the validation fold is \(-1\) if any of the task specific classifiers, \( F_{l+j}(x) \) for \( j = \{1, 2, ..., l\} \), that operate on higher layers produce a prediction that is better than \( F_l(x) \). Otherwise the label is \(+1\).
4. Repeat steps 2 and 3 for every fold.

In step 3, we consider least square error as a measure of goodness for regression tasks, and classification accuracy for the labeling tasks. One of the key ideas, as described in step 3, is to favor lower layers and allow a data point to propagate forward to higher layers only if the lower layer does not yield good prediction performance. The above mentioned cross-validation procedure results in an unbiased training set that can be used to train the gate keepers.

Figure 3 shows the histogram of how the best layers that make the predictions are distributed amongst the three different layers for different tasks when using different neural network architectures. The key observation is that the top most layer does not always provide the best performance. In fact a majority of the mass of the histogram lies with the pool5 layer, which is the lowest layer in the set of the three layers considered in our experiments. Such observations across the tasks and different neural networks strongly indicate the advantages of using the proposed framework that would enable faster test time inference via the early exit.

We’d like to point out that the dimensionality of the activations from the hidden / intermediate layers in neural network can be large. A very large dimensional feature vector...
will indeed adversely affect the running time for $G_l$ and $F_l$. However, we alleviate this problem by resorting to compressing the feature space via random projections. Specifically, instead of using the activations of the entire hidden layer, we project the $d$-dimensional activations to a lower dimensional subspace via a random projection matrix $\Phi$ of size $k \times d$, where $k << d$. Thus, the input to the classifiers is the $k$-dimensional vector: $\Phi \cdot \phi_l(x)$. The details of the complete architecture, including the task specific classifiers $F_l$, gate keepers $G_l$ are provided below with the experiments.

4. Experiments

In this section, we illustrate our approach by applying it to several deep learning applications. We present the results of experiments on existing popular deep networks, AlexNet [15] and VGGNet-16 [17], and one experiment where we use the features extracted from the hidden layers of AlexNet to create a state-of-the-art image interestingness measure [9] [10].

4.1. Experimental framework

In each of our experiments, we consider the activations of the three layers that are closest to the output (softmax) layer for the two deep networks under consideration - AlexNet and VGGNet-16. These layers are denoted as ‘fc7,’ ‘fc6,’ and ‘pool5’ respectively by the creators of these networks [15, 17]. ‘fc7’ denotes the features extracted from the final hidden layer, just before propagating through the fully connected layer to produce the class predictions in AlexNet and VGGNet. ‘fc6’ features are the activations of the layer before fc7 and ‘pool5’ is the layer before fc6.

For our experiments we have found that using activations from even earlier layers in both the networks have poor accuracy and using them does not provide a significant benefit, thus we only present results using pool5, fc6, and fc7 as the layers we augment with our approach. However, it is important to note it is possible for other applications or networks that earlier layers may provide a benefit, and our approach can easily be used with any number of layers.

Because we are investigating the use of the networks’ hidden layer activations as features and our goal is purely to evaluate our approach and not the flexibility of the networks, all of its weights are frozen to those provided online [3] which were learned on ImageNet dataset [16], i.e., we did not fine-tune any network for any of our experiments or datasets. We report the running time in seconds over all the test images for each component (passive learners, gating functions, the task-specific ‘active’ learners, and the feature extraction in the hidden layers) along with an overall frame-per-second (FPS) measure. Furthermore, we report a theoretical upper-bound for our approach, which is the mean average precision (MAP) if the gating functions were perfect, i.e., the gating functions select the right layer for each test image with 100% accuracy. We also report the scenario where the test points were randomly assigned to each layer, i.e., where the gating functions behave randomly. For each experiment, we also report an ‘efficiency’ metric defined as MAP per unit time (for classification tasks) or median correlation per unit time (for regression tasks) that we achieved on the test set.

4.2. ImageNet Classification

For our first set of experiments, we use two network architectures AlexNet and VGGNet-16, and apply our approach for the problem of image classification on ImageNet validation data. In order to effectively train the gate keepers, we used images that belong to the first 50 classes from the ImageNet training dataset (about 62,000 images). Upon deriving the layer labels on these images (as described in Sec. 3.2), we sampled them such that at every layer $l$, a gate keeper has equal number of positive and negative samples. Specifically, while training the gate-keeper at pool5, samples with labels fc6 or fc7 were considered as negative samples. Similarly, while training the gate-keeper at fc6, samples with labels pool5 and fc6 were considered as positive samples and those with fc7 were considered as negative samples. The training samples with ‘no-preference’ labels were not used for training the gating functions. (The “no preference” category indicates the cases that no layer’s output is correct). We found that addressing the skewed layer label distribution (as illustrated in Fig. 3) by this simple sampling strategy helped us in improving the performance of the gate keepers. Furthermore, owing to the number of
From the ImageNet validation dataset which contains a total of 50,000 images [16], we randomly chose 2,500 images that belong to the first 50 object classes. We extracted features from the hidden layers of AlexNet and VGGNet-16 separately to train the passive as well as task-specific learners. As discussed in the previous section, we pre-process ‘pool5’ to a lower dimensionality (denote as ‘pool5-compressed.’) The dimensionality of pool5 features in AlexNet and VGGNet-16 is 9216 and 25088 respectively, and we projected them down to 4096. This number was found via cross-validation.

Tables 1 and 2 show the mean average precision of our approach compared to the results of using the passive learners at each layer alone in addition to an upper-bound for our method. From these results we observe that despite having very low MAP scores, owing to low feature extraction time in the hidden layers, pool5 (and pool5-compressed) features have higher FPS and efficiency values. However, our approach yields better performance for both network architectures with some overhead in the run-time. The efficiency obtained from our approach outperforms the passive fc7 and fc6 learners in Table 1 and competes with them favorably in Table 2. Given that the passive fc7 and fc6 learners are predominantly used as off-the-shelf feature extractors for many vision tasks [7, 6], these results illustrate the benefit of employing active learners and gate-keepers on a per-image basis in computationally constrained environments. For instance, it can be seen from Table 2 that, given a constrained time budget of about 700 seconds, employing the proposed approach over using pool-5 alone would yield better overall classification performance.

### 4.3. Fine-grained Classification

The fine-grained dataset, released as part of the ImageNet challenge in 2013 [8], includes many categories of aircraft, birds, cars, dogs, and shoes. We evaluated our approach using AlexNet for extracting features on this dataset. We randomly sampled 2215 images that belong to 10 different breeds of dogs and trained and tested our passive and active learners over 50/20 splits as before. As above, in order to train the ‘gate keepers,’ we augmented our training data by using over 26,000 images from this dataset that belong to different breeds of dogs. We also adopted the uniform sampling strategy in order to have equal number of positive and negative samples for training the gate-keepers at each layer. Random Decision Forests were used to train the ‘gate keepers’ and SVMs were used for passive and task-specific learners. We note that our test set does not overlap with the training set at any point.

The problem of visual fine-grained classification is an interesting test case as it can be extremely challenging when compared to basic visual recognition tasks due to the subtle differences in the appearance of certain parts across related categories; however, we show that the benefits our approach still hold for this application. Table 3 lists the mean average precision and the computational costs incurred on this dataset for 500 test images. As with the previous experi-

<table>
<thead>
<tr>
<th>Layer</th>
<th>MAP</th>
<th>Hidden Layers</th>
<th>Gating function</th>
<th>Task-learner</th>
<th>Total</th>
<th>FPS</th>
<th>MAP / Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc7 alone</td>
<td>0.723</td>
<td>71.0</td>
<td>-</td>
<td>0.011</td>
<td>71.01</td>
<td>7.04</td>
<td>0.0101</td>
</tr>
<tr>
<td>fc6 alone</td>
<td>0.713</td>
<td>58.6</td>
<td>-</td>
<td>0.009</td>
<td>58.60</td>
<td>8.53</td>
<td>0.0121</td>
</tr>
<tr>
<td>pool5 alone</td>
<td>0.530</td>
<td>38.75</td>
<td>-</td>
<td>0.042</td>
<td>38.79</td>
<td>12.89</td>
<td>0.0136</td>
</tr>
<tr>
<td>pool5 compressed</td>
<td>0.496</td>
<td>39.65</td>
<td>-</td>
<td>0.008</td>
<td>39.66</td>
<td>12.60</td>
<td>0.0125</td>
</tr>
<tr>
<td>Our approach</td>
<td>0.637</td>
<td>51.55</td>
<td>0.23</td>
<td>0.043</td>
<td>51.55</td>
<td>9.70</td>
<td>0.0123</td>
</tr>
<tr>
<td>Our approach (upper bound)</td>
<td>0.794</td>
<td>44.66</td>
<td>0</td>
<td>0.027</td>
<td>44.687</td>
<td>11.19</td>
<td>0.0177</td>
</tr>
<tr>
<td>Our approach (random)</td>
<td>0.60</td>
<td>51.23</td>
<td>0</td>
<td>0.06</td>
<td>51.30</td>
<td>9.74</td>
<td>0.0117</td>
</tr>
</tbody>
</table>

Table 1. Mean Average Precision, running times, and MAP per unit time for 50 Classes of ImageNet using AlexNet (for 500 test images)

<table>
<thead>
<tr>
<th>Layer</th>
<th>MAP</th>
<th>Hidden Layers</th>
<th>Gating function</th>
<th>Output layer</th>
<th>Total</th>
<th>FPS</th>
<th>MAP / Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc7 alone</td>
<td>0.856</td>
<td>770.00</td>
<td>-</td>
<td>0.01</td>
<td>770.01</td>
<td>0.65</td>
<td>0.00117</td>
</tr>
<tr>
<td>fc6 alone</td>
<td>0.833</td>
<td>760.00</td>
<td>-</td>
<td>0.01</td>
<td>760.01</td>
<td>0.66</td>
<td>0.00110</td>
</tr>
<tr>
<td>pool5 alone</td>
<td>0.717</td>
<td>695.00</td>
<td>-</td>
<td>0.05</td>
<td>695.05</td>
<td>0.72</td>
<td>0.00103</td>
</tr>
<tr>
<td>pool5 compressed</td>
<td>0.717</td>
<td>696.50</td>
<td>-</td>
<td>0.02</td>
<td>696.52</td>
<td>0.72</td>
<td>0.00103</td>
</tr>
<tr>
<td>Our approach</td>
<td>0.792</td>
<td>719.87</td>
<td>0.20</td>
<td>0.062</td>
<td>720.13</td>
<td>0.69</td>
<td>0.00110</td>
</tr>
<tr>
<td>Our approach (upper bound)</td>
<td>0.895</td>
<td>715.24</td>
<td>0</td>
<td>0.053</td>
<td>715.30</td>
<td>0.70</td>
<td>0.00125</td>
</tr>
<tr>
<td>Our approach (random)</td>
<td>0.782</td>
<td>729.67</td>
<td>0</td>
<td>0.033</td>
<td>729.703</td>
<td>0.68</td>
<td>0.00107</td>
</tr>
</tbody>
</table>

Table 2. Mean Average Precision, running times, and MAP per unit time for 50 Classes of ImageNet using VGG-16 (for 500 test images)
Table 3. Accuracy and running times for Fine-grained classification (10 class); domain: dogs (for 500 test images)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Mean Accuracy</th>
<th>Hidden Layers</th>
<th>Gating function</th>
<th>Output layer</th>
<th>Total</th>
<th>FPS</th>
<th>MAP / Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc7 alone</td>
<td>0.9138</td>
<td>71.00</td>
<td>-</td>
<td>0.0063</td>
<td>71.0063</td>
<td>7.04</td>
<td>0.0129</td>
</tr>
<tr>
<td>fc6 alone</td>
<td>0.8610</td>
<td>58.60</td>
<td>-</td>
<td>0.0062</td>
<td>58.6062</td>
<td>8.53</td>
<td>0.0147</td>
</tr>
<tr>
<td>pool5 alone</td>
<td>0.8251</td>
<td>38.75</td>
<td>-</td>
<td>0.0144</td>
<td>38.7644</td>
<td>12.89</td>
<td>0.0213</td>
</tr>
<tr>
<td>pool5 compressed</td>
<td>0.8190</td>
<td>39.65</td>
<td>-</td>
<td>0.0063</td>
<td>39.6566</td>
<td>12.61</td>
<td>0.0206</td>
</tr>
<tr>
<td>Our approach</td>
<td>0.8784</td>
<td>52.39</td>
<td>0.16</td>
<td>0.0124</td>
<td>52.567</td>
<td>9.51</td>
<td>0.0167</td>
</tr>
<tr>
<td>Our approach (upper bound)</td>
<td>0.9210</td>
<td>44.56</td>
<td>0</td>
<td>0.0167</td>
<td>44.5818</td>
<td>11.22</td>
<td>0.0206</td>
</tr>
<tr>
<td>Our approach (random)</td>
<td>0.8519</td>
<td>51.37</td>
<td>0</td>
<td>0.01</td>
<td>51.38</td>
<td>9.73</td>
<td>0.0165</td>
</tr>
</tbody>
</table>

4.4. Image Interestingness Measure

In our last experiment, we use our approach to develop a state-of-the-art “interestingness” metric. Gygli et al. [10] investigated the visual interestingness of pictures and proposed a bag of image features-driven predictor to computationally capture this visual property. Additionally, they conducted psychophysical experiments to gather human opinion scores on a dataset consisting of 2222 images to facilitate the evaluation of their interestingness predictor.

Since the entire image collection has only 2222 images with ground truth interestingness scores, we were not able to augment the data while training the gate keepers. Thus, as was done in the earlier experiments, the data was divided into non-overlapping train and test splits (in 80-20 ratio). Since the training dataset is small, an SVM with an RBF kernel was used to train the gate keepers. Furthermore, since predicting interestingness scores is a regression task, unlike the previous experiments, an SVR with an RBF kernel was used to train the passive as well as the task-specific learners.

In Table 4, we report the median Pearson’s correlation coefficient (PLCC) and Spearman’s rank ordered correlation coefficient (SROCC) between the predicted and the ground truth interestingness scores computed over three splits on 444 test images of the interestingness dataset [10]. As with previous experiments, we also report the passive learners’ results and the upper bound performance of our approach.

From Table 4, it can be observed that the deep features from all three layers outperform the current top-performing interestingness predictor [10], which already demonstrates a state-of-the-art interestingness measure. Furthermore, the proposed approach competes well with the performance obtained from pool5 and pool5-compressed and is computationally more efficient than passive fc6 and fc7 learners. The lack of sufficient data to efficiently train the gatekeepers negatively affects their performance, as can be seen in Table 4, where our approach performs very similar to the scenario where gate keepers are replaced by random layer label assignment or pool5-alone features. However, in the scenario where the test points are randomly assigned to each layer, about two-thirds of the total test points would be assigned to fc6 and fc7 layers thus increasing the overall feature extraction time in the hidden layers. Instead, using the proposed approach, a large fraction of the test points get filtered at pool5, thus bringing down the overall running time and thereby our approach is more efficient.

5. Discussion

We have illustrated the efficacy of our approach with four experiments using two different deep network architectures across classification and regression tasks. Our approach shows two primary benefits. The first is the ability to easily and systematically trade accuracy with running time, similar to “anytime” algorithms. Our approach is the first we are aware of to provide this for deep learning architectures. This is a significant benefit for these architectures, as they are computationally complex and many deployments of such networks need the flexibility to adjust running time.

In comparison to related work in model compression, our approach does not require configuring and training a new network, which can require many hours and a million im-
The property that earlier layers can work as well as “deep features” is not an entirely new observation, in that it is not uncommon for people to select different layers as features for different applications, e.g. with AlexNet some use fc6 [6, 7] while Zhou et al. [22] evaluated using a number of different hidden units and ultimately settled on fc7. However, this choice is generally made manually and not in a systematic way. Instead our approach presents a way to learn this selection optimally and in a content dependent way and uses this to improve both for accuracy and speed.

**Limitations:** Our approach does have some limitations and areas for future work. The benefit of our approach is dictated by the distribution of accurate predictions vs. layers (Figure 3) and the performance of the active classifiers we call “gating functions.” If earlier layers provide no image-dependent improvement in accuracy, this poses a limitation in that our active learners essentially have no function to learn and thus become random classifiers, which leads to no benefit in aggregate accuracy, though they still might be helpful in trading accuracy vs. running time.

Given a good distribution, where earlier layers do have higher accuracy for some images, the quality of the active binary classifiers becomes the limiting step. Here the precision and recall dictates the overall aggregate accuracy of the final setup and the running time as well. The gap between our results and our theoretical upper-bound shows that there is some room for future work. A challenge is improving these classifiers without increasing the test running time, as any increase, such as due to using a more complex classifier, impacts the overall run-time benefits of bypassing layers in the deep network. Thus our primary area of focus for future work is to improve the performance of the classifiers serving as gate-keepers.

**References**


Figure 4. The trade-off between decision boundary for gating functions vs. accuracy, running time, and accuracy/running time. Threshold less than 0 indicates that the gating function allowed more data through the gate, thus the bottom left corner is equivalent to using pool5 only and the upper right is fc7 only. Our approach allows one to pick a trade-off in a continuous way between the extremes of best accuracy and best running time.


