Multi-Scale Dense Convolutional Networks for Efficient Prediction

Gao Huang Cornell University
Danlu Chen Fudan University
Tianhong Li Tsinghua University
Felix Wu Cornell University
Laurens van der Maaten Facebook AI Research
Kilian Weinberger Cornell University

Abstract

We introduce a new convolutional neural network architecture with the ability to adapt dynamically to computational resource limits at test time. Our network architecture uses progressively growing multi-scale convolutions and dense connectivity, which allows for the training of multiple classifiers at intermediate layers of the network. We evaluate our approach in two settings: (1) *anytime classification*, where the network's prediction for a test example is progressively updated, facilitating the output of a prediction at any time; and (2) *budgeted batch classification*, where a fixed amount of computation is available to classify a set of examples that can be spent unevenly across "easier" and "harder" inputs. Experiments on three image-classification datasets demonstrate that our proposed framework substantially improves the state-of-the-art in both settings.

1 Introduction

Recent years have witnessed a surge in demand for applications of visual object recognition, for instance in self-driving cars [2] or content-based image search [39]. This demand has in part been fueled by the promise generated through the astonishing progress of convolutional networks (CNNs) on visual object recognition benchmark competition datasets, such as ILSVRC [6] and COCO [29], where state-of-the-art models may have even surpassed human-level performance [13, 14].

However, the requirements of such competitions differ from real-world applications, and they tend to incentivize resource-hungry models with high computational demands during inference time. For example, the COCO 2016 competition was won by a large ensemble of computationally intensive CNNs[1] — a model likely too expensive for any resource-aware application. After all, super-human accuracy in pedestrian detection is of no use if the prediction is delivered after the car has already hit the pedestrian. Indeed, in many real-world applications the preferred network architecture is typically not the one that achieves the highest classification accuracy, but the one that achieves the best accuracy within some *computational budget*. If the test-time budget is known during training, one can easily learn a network with an appropriate size that fits exactly within the computational budget. But what if the budget is unknown or is amortized across test cases?

In this paper, we focus on exactly these two settings. We propose a novel convolutional network architecture with an internal cascade of intermediate classifiers (Figure 1), which we refer to as *Multi-Scale DenseNet (MSDNet)*. Despite their limited depth, the intermediate classifiers are competitive because of two crucial components of MSDNets: *multi-scale feature maps* [35] and *dense connectivity* [16]. The multi-scale feature maps produce high-level feature representations that are amenable to classification early on. The dense connectivity pattern allows the network to reuse and bypass existing features from prior layers and ensures high accuracies in later layers.

[1] http://image-net.org/challenges/talks/2016/GRMI-COCO-slidedeck.pdf

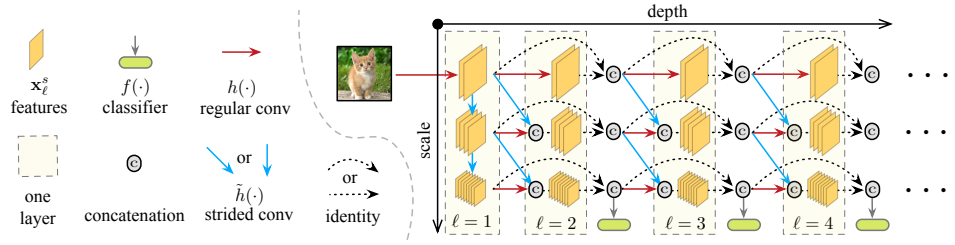


Figure 1: Illustration of the first four layers of an MSDNet with three scales. The horizontal direction corresponds to the layer direction (depth) of the network. The vertical direction corresponds to the scale of the feature maps. Horizontal arrows indicate a regular convolution operation, whereas diagonal and vertical arrows indicate a strided convolution operation. Classifiers only operate on feature maps at the coarsest scale. Connections across more than one layer are not drawn explicitly: these connections are implicit through recursive concatenations.

We study the performance of MSDNets in two settings with computational constraints at test-time: *anytime prediction*, where the network can be forced to output a prediction at any given point in time; and *budgeted batch classification*, where a fixed computational budget is shared across a large set of examples. The anytime setting is applicable to, for example, self-driving cars, which need to process high-resolution video streams on-the-fly and may need to output a prediction immediately when an unexpected event occurs. The budgeted batch classification setting is applicable to, for example, web services that may need to process a varying number of uploaded test examples per second using fixed resources [1]. In the budgeted batch classification setting, systems may increase their average accuracy by reducing the amount of computation spent on "easy" cases and spending the saved computation on "difficult" cases.

We evaluate MSDNets on three image-classification datasets. In the anytime classification setting, we show that it is possible to provide the ability to output a prediction at any time while maintain high accuracies throughout. In the batch classification setting, we show that MSDNets can be effectively used to adapt the amount of computation to the difficulty of the example to be classified, which allows us to reduce the computational requirements of our models drastically whilst performing on par with state-of-the-art CNNs in terms of overall classification accuracy.

2 Related Work

We briefly review some related prior work on computation-efficient networks, memory-efficient networks, and cost-sensitive machine learning. Our network architecture draws inspiration from several other network architectures.

Computation-efficient networks. Most prior work on (convolutional) networks that are computationally efficient at test time focuses on reducing model size after training. In particular, many studies propose to prune weights [12, 26, 28] or quantize weights [18, 33] during or after training. These approaches are generally effective because deep networks often have a substantial number of redundant weights that can be pruned or quantized without sacrificing (and sometimes even improving) performance. Prior work also studies approaches that directly learn *compact* models with less parameter redundancy. For example, the knowledge-distillation method [4, 15] trains small student networks to reproduce the output of a much larger teacher network or ensemble. Our work differs from those approaches in that we train a single model that can trade off computation for accuracy at prediction time without any re-training or finetuning. Indeed, weight pruning and knowledge distillation can be used in combination with our approach, and may lead to further computational improvements.

Resource-efficient machine learning. Various prior studies explore computationally efficient variants of traditional machine-learning models [10, 21, 31, 37, 38, 40–42]. Most of these studies focus on how to incorporate the computational requirements of computing particular features in the training of machine-learning models such as (gradient-boosted) decision trees. Whilst our study is certainly inspired by this prior work, the architecture we explore differs substantially from prior work on resource-efficient machine learning: most prior work exploits characteristics of machine-learning models (such as decision trees) that do not apply to deep networks. Our work is possibly most closely related to recent work on FractalNets [25], which can perform anytime prediction by progressively evaluating subnetworks of the full network. FractalNets differ from our work in that they are not explicitly optimized for computation efficiency: our experiments show that MSDNets substantially outperform FractalNets. Our dynamic evaluation strategy for reducing batch computational cost

is closely related to the the *adaptive computation time* approach [7, 8], and the recently proposed method of adaptively evaluating neural networks [3]. Different from these works, our method adopts a specially designed network with multiple classifiers, which are jointly optimized during training and can directly output confidence scores to control the evaluation process for each test example. The *adaptive computation time* method [8] and its extension [7] also perform adaptive evaluation on test examples to save batch computational cost, but focus on skipping units rather than layers. In [32], a "composer"model is trained to construct the evaluation network from a set of sub-modules for each test example. By contrast, our work uses a single CNN architecture with multiple intermediate classifiers that can be trained end-to-end.

Related network architectures. Our network architecture borrows elements from neural fabrics [35] and others [20, 22] to rapidly construct a low-resolution feature map that is amenable to classification, whilst also maintaining feature maps of higher resolution that are essential for obtaining high classification accuracy. We use the same feature-concatenation approach as DenseNets [16], which allows us to construct highly compact models. A similar way of reusing features is used in progressive networks [34]. Our architecture is related to deeply supervised networks [27] in that it incorporates classifiers at multiple layers throughout the network. In contrast to all these prior architectures, our network is designed to operate in settings with computational constraints at test time.

3 Problem Setup

We consider two settings that impose computational constraints at prediction time.

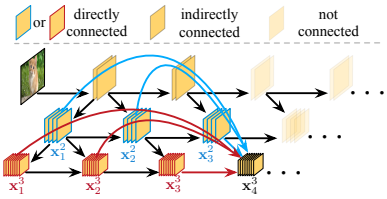
Anytime prediction. In the anytime prediction setting [10], there is a finite computational budget $B > 0$ available for each test example $\mathbf{x}$. The budget is nondeterministic, and varies per test instance. It is determined by the occurrence of an event that requires the model to output a prediction immediately. We assume that the budget is drawn from some joint distribution $P(\mathbf{x}, B)$. In some applications $P(B)$ may be independent of $P(\mathbf{x})$ and can be estimated. For example, if the event is governed by a Poisson process, $P(B)$ is an exponential distribution. We denote the loss of a model $f(\mathbf{x})$ that has to produce a prediction for instance $\mathbf{x}$ within budget B by $L(f(\mathbf{x}), B)$. The goal of an anytime learner is to minimize the expected loss under the budget distribution: $L(f) = \mathbb{E}\left[L(f(\mathbf{x}), B)\right]_{P(\mathbf{x},B)}$. Here $L(\cdot)$ denotes a suitable loss function. As is common in the empirical risk minimization framework, the expectation under $P(\mathbf{x}, B)$ may be estimated by an average over samples from $P(\mathbf{x}, B)$.

Budgeted batch classification. In the budgeted batch classification setting, the model needs to classify a set of examples $\mathcal{D}_{test} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ within a finite computational budget $B > 0$ that is known in advance. The learner aims to minimize the loss across all examples in $\mathcal{D}_{test}$ within a cumulative cost bounded by B, which we denote by $L(f(\mathcal{D}_{test}), B)$ for some suitable loss function $L(\cdot)$. It can potentially do so by spending less than $\frac{B}{M}$ computation on classifying an "easy" example whilst spending more than $\frac{B}{M}$ computation on classifying a "difficult" example.

4 Network Architecture

Like deeply supervised networks [27] and Inception models [36], our convolutional network has classifiers operating on feature maps at intermediate layers. Each of these classifiers can be used to output a prediction, which allows for retrieving preliminary predictions before a test image is propagated through the whole network. Such preliminary retrieval is essential in both the budgeted batch classification setting and the anytime prediction setting.

Challenges. There are two main challenges in designing network architectures that have classifiers at various intermediate layers: 1. obtaining acceptable prediction accuracies at an early stage in the network whilst training a deep architecture, and 2. ensuring that the computation of early classifiers is not wasted when later classifiers are evaluated. The first challenge arises because classifiers that operate on low-level features (*i.e.*, on features obtained before several levels of down-sampling) are generally inaccurate. The second challenge arises because feature maps in later layers generally capture different structure than features in early layers, which makes it difficult to construct classifiers that complement each other. Our convolutional networks address these two challenges via two main architectural changes: *multi-scale feature maps* and *dense connectivity*. We discuss both changes in detail below. An illustration of our architecture is shown in Figure 1.



x_ℓ^s	$\ell=1$	$\ell=2$	$\ell=3$	$\ell=4$
$s=1$	$h_1^1(\mathbf{x}_0^1)$	$h_2^1\left(\mathbf{x}_1^1\right)$	$h_3^1\left(\left[\mathbf{x}_1^1, \mathbf{x}_2^1\right]\right)$	$h_4^1\left(\left[\mathbf{x}_1^1, \mathbf{x}_2^1, \mathbf{x}_3^1\right]\right)$
$s=2$	$\tilde{h}_1^2(\mathbf{x}_1^1)$	$\begin{bmatrix} \tilde{h}_2^2\left(\mathbf{x}_1^1\right), \\ h_2^2\left(\mathbf{x}_1^2\right) \end{bmatrix}$	$\begin{bmatrix} \tilde{h}_3^2\left(\left[\mathbf{x}_1^1, \mathbf{x}_2^1\right]\right), \\ h_3^2\left(\left[\mathbf{x}_1^2, \mathbf{x}_2^2\right]\right) \end{bmatrix}$	$\begin{bmatrix} \tilde{h}_4^2\left(\left[\mathbf{x}_1^1, \mathbf{x}_2^1, \mathbf{x}_3^1\right]\right), \\ h_4^2\left(\left[\mathbf{x}_1^2, \mathbf{x}_2^2, \mathbf{x}_3^2\right]\right) \end{bmatrix}$
$s=3$	$\tilde{h}_1^3(\mathbf{x}_1^2)$	$\begin{bmatrix} \tilde{h}_2^3\left(\mathbf{x}_1^2\right), \\ h_2^3\left(\mathbf{x}_1^3\right) \end{bmatrix}$	$\begin{bmatrix} \tilde{h}_3^3\left(\left[\mathbf{x}_1^2, \mathbf{x}_2^2\right]\right), \\ h_3^3\left(\left[\mathbf{x}_1^3, \mathbf{x}_2^3\right]\right) \end{bmatrix}$	$\begin{bmatrix} \tilde{h}_4^3\left(\left[\mathbf{x}_1^2, \mathbf{x}_2^2, \mathbf{x}_3^2\right]\right), \\ h_4^3\left(\left[\mathbf{x}_1^3, \mathbf{x}_2^3, \mathbf{x}_3^3\right]\right) \end{bmatrix}$

Figure 2: The output $\mathbf{x}_\ell^s$ of layer ℓ at the s^{th} scale in a MSDNet. Herein, $[\dots]$ denotes the concatenation operator, $h_\ell^s(\cdot)$ a regular convolution transformation, and $\tilde{h}_\ell^s(\cdot)$ a strided convolutional. Note that the outputs of h_ℓ^s and $\tilde{h}_\ell^s$ have the same feature map size; their outputs are concatenated along the channel dimension.

Multi-scale feature maps. The first main change is that we maintain a feature representation at *multiple scales* in each layer[2] of the network, akin to convolutional neural fabrics [35]. The feature maps at a particular layer and scale are computed by concatenating the results of one or two convolutions: 1. the result of a regular convolution applied on the same-scale features from the previous layer (horizontal connections), and if possible 2. the result of a strided convolution applied on the finer-scale feature map from the previous layer (diagonal connections). The horizontal connections preserve and progress high-resolution information, which facilitates the construction of high-level features in later layers. The vertical connections produce low-resolution features in early layers that are amenable to classification.

It is noteworthy that although we adopt a similar multi-scale architecture as neural fabrics [35], our design differs substantially. MSDNets have a reduced number of scales and no sparse channel connectivity or up-sampling paths. They are at least one order of magnitude more efficient and typically more accurate — for example, an MSDNet with less than 1 million parameters obtains a test error below 7.0% on CIFAR-10 [23], whereas [35] report 7.43% with over 20 million parameters.

Dense connectivity. In order to allow each layer to receive inputs directly from all its previous layers we use *dense connections* [16] within each scale. Dense connections connect each layer with all subsequent layers directly (through feature concatenation). Such dense connections significantly reduce redundant computations by encouraging feature reuse and improve optimization by eliminating the gradient vanishing problem. Most importantly, dense connections make MSDNets robust towards the disruptive effects of intermediate classifiers. The feature maps that feed into an intermediate classifier are dominated by the loss of that particular classifier, which tends to be beneficial in the short term but sub-optimal for later classifiers (see section 4). Dense connections allow the model to bypass features that are only relevant to an intermediate classifier when computing features in later layers; they can do so in a way that is much harder to achieve with residual connections (as we show in section 6). We present the details of MSDNet below.

First layer. The first layer ($\ell=1$) is unique as it includes vertical connections in Figure 1. Its main purpose is to "seed" representations on all S scales. One could view its vertical layout as a miniature "S-layers" convolutional network (S=3 in Figure 1). Let us denote the output feature maps at layer ℓ and scale s as $\mathbf{x}_\ell^s$ and the original input image as $\mathbf{x}_0^1$. Feature maps at coarser scales are obtained via down-sampling. The output $\mathbf{x}_1^s$ of the first layer is formally given in the top row of Figure 2.

Subsequent layers. Following the dense connectivity pattern proposed by [16], the output feature maps $\mathbf{x}_\ell^s$ produced at subsequent layers, $\ell>1$, and scales, s, are a concatenation of transformed feature maps from all previous feature maps of scale s and $s-1$ (if $s>1$). Formally, the ℓ-th layer of our network outputs a set of features at S scales $\left\{\mathbf{x}_\ell^1, \dots, \mathbf{x}_\ell^S\right\}$, given in the last row of Figure 2.

Classifiers. The classifiers in MSDNets also follow the dense connectivity pattern within the coarsest scale, S, i.e., the classifier at layer ℓ uses all the features $\left[\mathbf{x}_1^S, \dots, \mathbf{x}_\ell^S\right]$. Each classifier consists of two convolutional layers, followed by one average pooling layer and one linear layer. In practice, we only attach classifiers to some of the intermediate layers, and we let $f_k(\cdot)$ denote the k^{th} classifier. During testing in the *anytime* setting we propagate the input through the network until the budget is exhausted and output the most recent prediction.

In the *batch budget* setting at test time, an example traverses the network and exits after classifier f_k if its prediction confidence (we use the maximum value of the softmax probability as a confidence measure) exceeds a pre-determined threshold θ_k. Before training, we compute the computational

[2]Here, we use the term "layer" to refer to a column in Figure 1.

cost, C_k, required to process the network up to the k^{th} classifier. We denote by $0 < q \leq 1$ a fixed *exit probability* that a sample that *reaches* a classifier will obtain a classification with sufficient confidence to exit. We assume that q is constant across all layers, which allows us to compute the probability that a sample exits at classifier k as: $q_k = z(1-q)^{k-1}q$, where z is a normalizing constant that ensures that $\sum_k p(q_k) = 1$. At test time, we need to ensure that the overall cost of classifying all samples in $\mathcal{D}_{test}$ does not exceed our budget B, which gives rise to the constraint $|\mathcal{D}_{test}| \sum_k q_k C_k \leq B$. We can solve this constraint for q and determine the thresholds θ_k on a validation set in such a way that approximately $|\mathcal{D}_{test}|q_k$ validation samples exit at the k^{th} classifier. In preliminary experiments, we also experimented with entropy-based confidence measures and class-specific thresholds, but we did not find these approaches to work better.

Loss functions. During training we use logistic loss functions $L(f_k)$ for all classifiers and minimize a weighted cumulative loss: $\frac{1}{|\mathcal{D}|}\sum_{(\mathbf{x},y)\in\mathcal{D}}\sum_k w_k L(f_k)$. Herein, $\mathcal{D}$ denotes the training set and $w_k \geq 0$ the weight of the k-th classifier. If the budget distribution $P(B)$ is known, we can use the weights w_k to incorporate our prior knowledge about the budget B in the learning. Empirically, we find that using the same weight for all loss functions (*i.e.*, setting $\forall k\colon w_k = 1$) works well in practice.

Network reduction and lazy evaluation. There are three straightforward ways to further reduce the computational requirements of MSDNets. First, it is inefficient to maintain all the finer scales until the last layer of the network. One simple strategy to reduce the size of the network is by splitting it into S blocks along the depth dimension, and only keeping the coarsest $(S-i+1)$ scales in the i^{th} block. This reduces computational cost for both training and testing. Every time a scale is removed from the network, we add a transition layer between the two blocks that merges the concatenated features using a 1×1 convolution and cuts the number of channels in half before feeding the fine-scale features into the coarser scale via a strided convolution (this is similar to the DenseNet-BC architecture of [16]). Third, since a classifier at layer ℓ only uses features from the coarsest scale, the finer feature maps in layer ℓ (and some of the finer feature maps in the previous $S-2$ layers) do not influence the prediction of that classifier. Therefore, we group the computation in "diagonal blocks" such that we only propagate the example along paths that are required for the evaluation of the next classifier. This minimizes the amount of unnecessary computations performed when we need to stop because the computational budget is exhausted. We call this strategy *lazy evaluation*.

5 Experiments

We evaluate the effectiveness of our approach on three image classification datasets, i.e., the CIFAR-10, CIFAR-100 [23] and ILSVRC 2012 (ImageNet) [6] datasets. We leave the detailed experimental setup and network configurations, as well as the results on CIFAR-10, to the supplementary materials. Code to reproduce all results is available at `https://github.com/gaohuang/MSDNet`.

5.1 Anytime Prediction

In the anytime prediction setting, the model maintains a progressively updated distribution over classes. During inference it can be forced to output its most up-to-date prediction at an arbitrary time.

Baselines. There exist two convolutional network architectures suitable for anytime prediction: FractalNets [25] and deeply supervised networks [27]. FractalNets allow for multiple evaluation paths during inference time, which vary in computation time. In the anytime setting, paths are evaluated in order of increasing computation. In our result figures, we show the FractalNet results reported in the original paper [25] for reference. Deeply supervised networks introduce multiple early-exit classifiers throughout a network, which are applied on the features of the particular layer they are attached to. Instead of using the original model proposed in [27], we use the more competitive ResNet and DenseNet architectures (referred to as *DenseNet-BC* in [16]) as the base networks in our experiments with deeply supervised networks. We refer to these as *ResNet^{MC}* and *DenseNet^{MC}*, where MC stands for *multiple classifiers*. Both networks require about 3×10^8 FLOPs when fully evaluated; the detailed network configurations are presented in the supplementary material. In addition, we include ensembles of ResNets and DenseNets of *varying* or *identical* sizes. At test time, the networks are evaluated sequentially (in ascending order of network size) to obtain predictions for the test data. All predictions are averaged over the evaluated classifiers. On ImageNet, we compare MSDNet against a highly competitive ensemble of ResNets with varying depths. The ResNets we use in the ensemble range from 10 layers to 50 layers. All the models are the same as those described in

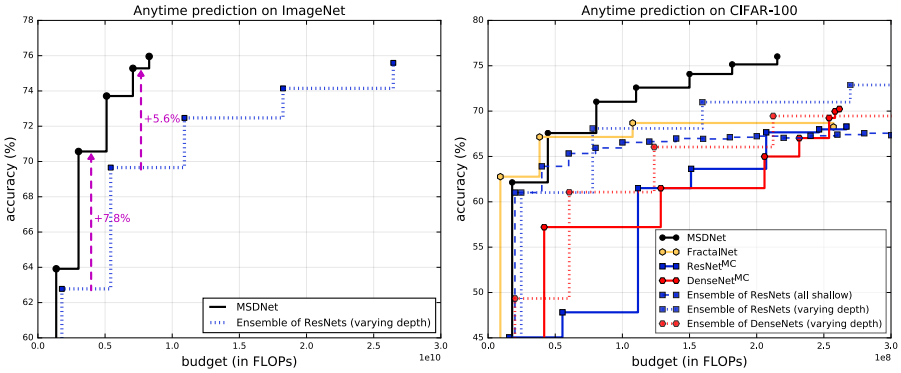


Figure 3: Accuracy (*top-1*) of *anytime prediction* models as a function of computational budget on the ImageNet (left) and CIFAR-100 (right) datasets. Higher is better.

[14], except the ResNet-10 and ResNet-26, which are variants of ResNet-18 that were modified by removing/adding one residual block from/to each of the last four spatial resolutions in the network.

Anytime prediction results are presented in Figure 3. The left plot shows the classification top-1 accuracy on the ImageNet validation set. Here, for all budgets in our evaluation, the accuracy of MSDNet substantially outperforms the ResNet ensemble. In particular, when the budget ranges from 0.5×10^{10} to 1.0×10^{10} FLOPs, MSDNet achieves $\sim 4\% - 8\%$ higher top-1 accuracy.

We evaluate more baselines on CIFAR-100 (and CIFAR-10; see supplementary materials). We observe that MSDNet substantially outperforms ResNets$^{\text{MC}}$ and DenseNets$^{\text{MC}}$ at any computational budget. This is due to the fact that after just a few layers, MSDNets have produced low-resolution feature maps that are much more suitable for classification than the high-resolution feature maps in the early layers of ResNets or DenseNets. MSDNet also outperforms the other baselines for nearly all computational budgets, although it performs on par with ensembles when the budget is very small. In the extremely low-budget regime, ensembles have an advantage because their predictions are performed by the first (small) network, which is optimized exclusively for the low budget. Having said that, the accuracy of ensembles does not increase nearly as fast when the budget is increased. MSDNet outperforms ensembles as soon as the ensemble needs to evaluate a second model: unlike MSDNets, this forces the ensemble to repeat the computation of similar low-level features multiple times. The accuracy of ensembles saturates rapidly when all networks in the ensemble are shallow.

5.2 Budgeted batch classification

In budgeted batch classification setting, the predictive model receives a batch of M instances and a computational budget B for classifying all M instances. In this setting, we use *dynamic evaluation*: we perform *early-exiting* of "easy" examples at early classifiers whilst propagating "hard" examples through the entire network, using the procedure described in Section 4.

Baselines. On ImageNet, we compare the dynamically evaluated MSDNet with five ResNets models, the 121-layer DenseNet of [16], AlexNet [24], and GoogleLeNet [36]; see the supplementary material for details. We also evaluate an ensemble of the five ResNets that uses exactly the same dynamic-evaluation procedure as MSDNets at test time: "easy" images are only propagated through the smallest ResNet(-10), whereas "hard" images are classified by all five ResNet models (predictions are averaged across all evaluated networks in the ensemble). We classify batches of $M = 128$ images.

On CIFAR-100, we compare MSDNet with several highly competitive baselines, including ResNets [14], DenseNets [16] of varying sizes, Stochastic Depth Networks [17], Wide ResNets [43] and FractalNets [25]. We also compare MSDNet to the ResNet$^{\text{MC}}$ and DenseNet$^{\text{MC}}$ models that were used in Section 5.1, using dynamic evaluation at test time. We denote these baselines as *ResNet^{MC} / DenseNet^{MC} with early-exits*. To prevent the result plots from becoming too cluttered, we present CIFAR-100 results with dynamically evaluated ensembles in the supplementary material. We classify batches of $M = 256$ images at test time.

Budgeted batch classification results on ImageNet are shown in the left panel of Figure 5. The plot shows that the predictions of MSDNets with dynamic evaluation are substantially more accurate than those of ResNets and DenseNets that use the same amount of computation. For instance, with an average budget of 0.5×10^{10} FLOPs, MSDNet achieves a top-1 accuracy of $\sim$76%, which is $\sim$5% higher than that achieved by a ResNet with the same number of FLOPs. Compared to the

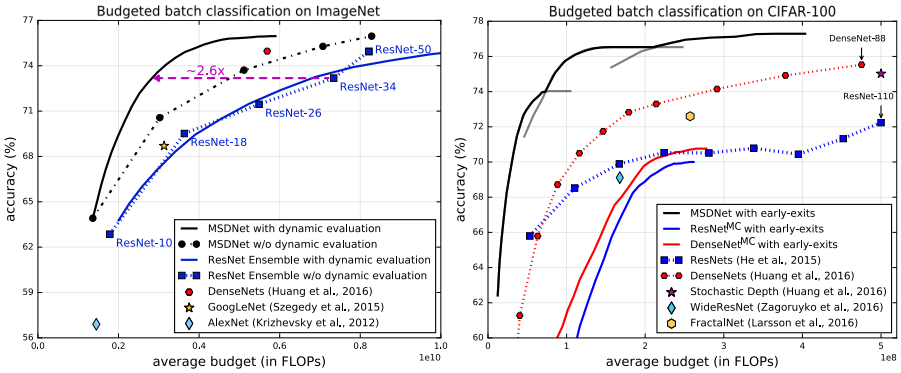


Figure 5: Accuracy (top-1) of *budgeted batch classification* models as a function of average computational budget per image the on ImageNet (left) and CIFAR-100 (right) datasets. Higher is better.

computationally efficient DenseNet-121, MSDNet uses $1.6\times$ fewer FLOPs to achieve the same classification accuracy. Moreover, MSDNet with dynamic evaluation allows for very precise tuning of the computational budget that is consumed, which is not possible with individual ResNet or DenseNet models. The ensemble of ResNets with dynamic evaluation performs roughly on par with the individual ResNet models (but it does allow for setting the computational budget very precisely).

The right panel of Figure 5 shows our results on CIFAR-100. We trained three MSDNets with different depths, each of which covers a different range of computational budgets. We plot the performance of each MSDNet as a gray curve; we select the best model for each budget based on its accuracy on the validation set, and plot the corresponding accuracy as a black curve. The results show that MSDNets consistently outperform all the baselines in terms of accuracy, given a particular budget. Notably, MSDNet performs on par with a 110-layer ResNet using only 1/10th of the computational budget. It is up to ~ 5 times more efficient than DenseNets, Stochastic Depth Networks, Wide ResNets, and FractalNets, whilst achieving higher accuracies than all these networks when there are no computational constraints. Similar to results in the anytime-prediction setting, MSDNet substantially outperform ResNets and DenseNets with multiple intermediate classifiers (*-MC*), which provides further evidence that the high-resolution features in the early layers of these models are not suitable for classification.

Visualization. To illustrate the ability of our approach to reduce the computational requirements for classifying "easy" examples, we show twelve randomly sampled test images from two ImageNet classes in Figure 4. The top row shows "easy" examples that were correctly classified and exited by the first classifier. The bottom row shows "hard" examples that were classified correctly by the final classifier because earlier classifiers refused to exit them, and that would have been incorrectly classified by the first classifier. The figure suggests that early classifiers recognize prototypical class examples, whereas the last classifier recognizes non-typical images.

(a) Red wine (b) Volcano

Figure 4: Sampled images from the ImageNet classes *Red wine* and *Volcano*. Top row: images exited from the first classifier of a MSDNet with correct prediction; Bottom row: images failed to be correctly classified at the first classifier but were correctly predicted and exited at the last layer.

6 Analysis and Conclusion

We perform additional experiments to shed light on the contributions of the three main components of MSDNet, *viz.*, multi-scale feature maps, dense connectivity, and intermediate classifiers.

Ablation study. We start from an MSDNet with six intermediate classifiers and remove the three main components one at a time. To make our comparisons fair, we keep the computational costs of the full networks similar, at around 3.0×10^8 FLOPs, by adapting the network width, i.e., number of output channels at each layer. After removing all the three components in an MSDNet, we obtain a regular VGG-like convolutional network. We show the classification accuracy of all classifiers in a model in the left panel of Figure 6. Several observations can be made: 1. the dense connectivity is crucial for the performance of MSDNet and removing it hurts the overall accuracy drastically (orange

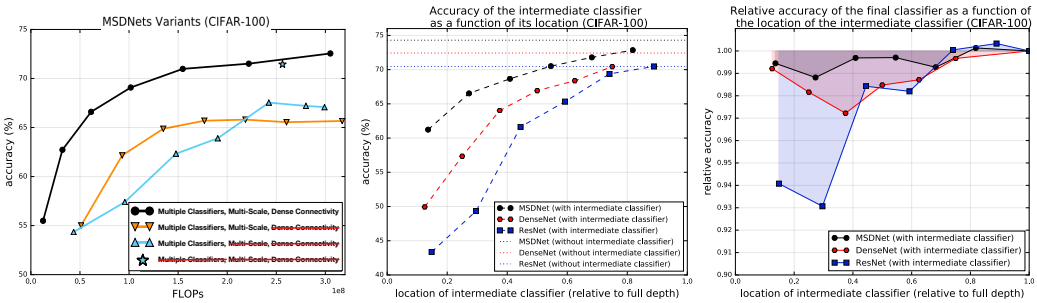


Figure 6: **Left:** Ablation study of MSDNets that shows the effect of dense connectivity, multi-scale feature maps, and intermediate classifiers **Middle:** Absolute accuracy of a single intermediate classifier compared to that of the final classifier for three network architectures. **Right:** Relative accuracy of the final classifier after introducing a single intermediate classifier at different layers in three network architectures. All experiments were performed on the CIFAR-100 dataset. Higher is better.

vs. black curve); 2. removing multi-scale convolution hurts the accuracy only in the lower budget regions, which is consistent with our motivation that the multi-scale design introduces discriminative features early on; 3. the final canonical CNN (star) performs similarly as MSDNet under the specific budget that matches its evaluation cost exactly, but it is unsuited for varying budget constraints. Interestingly, this final CNN performs substantially better at its particular budget region than the model without dense connectivity (orange line). This seems to suggest that dense connectivity is particularly important in combination with multiple classifiers.

Intermediate classifiers. Motivated by these results, we perform an additional experiment to investigate the importance of dense connectivity and multi-scale convolutions in the context of intermediate classifiers. In particular, we train MSDNets, DenseNets and ResNets with one intermediate classifier in varying (evenly spaced) locations and observe its accuracy and the effect it has on the *final* classifiers. For each architecture, the middle panel of Figure 6 shows the accuracy of the intermediate classifier as a function of its location (dashed lines). It also includes the accuracies of the respective final classifiers when no intermediate classifiers are introduced (dotted horizontal lines). Although all three architectures give rise to similar trends — the intermediate classifiers improve as their relative depth increase — there is a visible improvement obtained through the introduction of dense connectivity (from ResNet to DenseNet) and the multi-scale architecture (DenseNet to MSDNet).

The right plot in Figure 6 sheds light onto what degree an intermediate classifier influences the training of the final classifier. The graph shows the accuracies of the *final* classifiers as a function of the location of the *intermediate* classifiers, relative to the accuracy of a network without any intermediate classifier. The introduction of an intermediate classifier harms the final ResNet classifier, reducing its accuracy by up to 7%. DenseNets and MSDNet do not suffer as much from this effect. We postulate that this phenomenon may be caused by the intermediate classifier changing the type of features learned in its preceding layers, which improves its accuracy but is sub-optimal for classification in later layers. This effect becomes more pronounced when the first classifier is attached to an earlier layer. Dense connectivity allows later layers to bypass harmful early features and maintain the high accuracy of the final classifier irrespective of the location of the intermediate classifier.

Conclusion. We presented a study on training convolutional networks that are optimized to operate in settings with computational budgets at test time. In particular, we focus on two different computational budget settings, namely, testing with anytime prediction and budgeted batch classification. Both settings require individual test samples to take varying amounts of computation time in order to obtain competitive results. We introduce a new convolutional network architecture, which incorporates two main changes: 1. maintaining *multi-scale feature maps* in early layers of the convolutional network, and 2. using a *feature-map concatenation* approach that facilitates feature re-use in subsequent layers. The two changes allow us to attach intermediate classifiers throughout the network architecture and avoid wasted computation by re-using feature maps across classifiers. The results of our experiments demonstrate the effectiveness of these changes in settings with computational constraints.

In future work, we plan to investigate the effect of these architecture changes in experiments on tasks other than image classification, *e.g.*, image segmentation [30]. We also intend to explore approaches that combine MSDNets, for instance, with model compression [5, 11] and spatially adaptive computation [7] to further improve computational efficiency.

References

[1] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the International Conference on Machine Learning*, 2016.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for fast test-time prediction. *arXiv preprint arXiv:1702.07811*, 2017.

[4] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *ACM SIGKDD*, pages 535–541. ACM, 2006.

[5] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[7] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. *arXiv preprint arXiv:1612.02297*, 2016.

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

[9] S. Gross and M. Wilber. Training and investigating residual nets. 2016.

[10] A. Grubb and D. Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, volume 15, pages 458–466, 2012.

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[12] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IJCNN*, pages 293–299, 1993.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014.

[16] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017.

[17] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661. Springer, 2016.

[18] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NIPS*, pages 4107–4115, 2016.

[19] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 770–778, 2015.

[20] J.-H. Jacobsen, E. Oyallon, S. Mallat, and A. W. Smeulders. Multiscale hierarchical convolutional networks. *arXiv preprint arXiv:1703.04140*, 2017.

[21] S. Karayev, M. Fritz, and T. Darrell. Anytime recognition of objects and scenes. In *CVPR*, pages 572–579, 2014.

[22] T. Ke, M. Maire, and S. X. Yu. Neural multigrid. *CoRR*, abs/1611.07661, 2016.

[23] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[25] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

[26] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989.

[27] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, volume 2, page 5, 2015.

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[30] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[31] F. Nan, J. Wang, and V. Saligrama. Feature-budgeted random forest. In *ICML*, pages 1983–1991, 2015.

[32] A. Odena, D. Lawson, and C. Olah. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780*, 2017.

[33] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016.

[34] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[35] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *NIPS*, pages 4053–4061, 2016.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[37] K. Trapeznikov and V. Saligrama. Supervised sequential classification under budget constraints. In *AI-STATS*, pages 581–589, 2013.

[38] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4(34–47), 2001.

[39] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li. Deep learning for content-based image retrieval: A comprehensive study. In *ACM Multimedia*, pages 157–166, 2014.

[40] J. Wang, K. Trapeznikov, and V. Saligrama. Efficient learning by directed acyclic graph for resource constrained prediction. In *NIPS*, pages 2152–2160. 2015.

[41] Z. Xu, O. Chapelle, and K. Q. Weinberger. The greedy miser: Learning under test-time budgets. In *ICML*, pages 1175–1182, 2012.

[42] Z. Xu, M. Kusner, M. Chen, and K. Q. Weinberger. Cost-sensitive tree of classifiers. In *ICML*, volume 28, pages 133–141, 2013.

[43] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

7 Details of Experimental Setup

7.1 Datasets

The two CIFAR datasets contain $50,000$ training and $10,000$ test images of 32×32 pixels; we hold out $5,000$ training images as a validation set. The datasets comprise 10 and 100 classes, respectively. We follow [14] and apply standard data-augmentation techniques to the training images: images are zero-padded with 4 pixels on each side, and then randomly cropped to produce 32×32 images. Images are flipped horizontally with probability 0.5, and normalized by subtracting channel means and dividing by channel standard deviations. The ImageNet dataset comprises $1,000$ classes, with a total of 1.2 million training images and 50,000 validation images. We hold out 50,000 images from the training set to estimate the confidence threshold for classifiers in MSDNet. We adopt the data augmentation scheme of [14, 16] at training time; at test time, we classify a 224×224 center crop of images that were resized to 256×256 pixels.

7.2 Details of MSDNet Architecture and Baseline Networks

We use MSDNet with three scales on the CIFAR datasets. The convolutional layer functions in the first layer, h_1^s, denote a sequence of 3×3 convolutions (Conv), batch normalization (BN; [19]), and rectified linear unit (ReLU) activation. In the computation of $\tilde{h}_1^s$, down-sampling is performed by applying convolutions using strides that are powers of two. For subsequent feature layers, the transformations h_ℓ^s and $\tilde{h}_\ell^s$ are defined following the design in DenseNets [16]: Conv(1×1)-BN-ReLU-Conv(3×3)-BN-ReLU. We set the number of output channels of the three scales to 6, 12, and 24, respectively. Each classifier has two down-sampling convolutional layers with 128 dimensional 3×3 filters, followed by a 2×2 average pooling layer and a linear layer. The MSDNet used for ImageNet has four scales, respectively producing 16, 32, 64, and 64 feature maps at each layer. The original images are first transformed by a 7×7 convolution and a 3×3 max pooling (both with stride 2), before entering the first layer of MSDNets. The classifiers have the same structure as those used for the CIFAR datasets, except that the number of output channels of each convolutional layer is set to be equal to the number of its input channels.

Network architecture for anytime prediction. The MSDNet used in our anytime-prediction experiments has 24 layers (each layer corresponds to a column in Fig. 1 of the main paper), using the reduced network with transition layers as described in Section 4. The classifiers operate on the output of the $2\times(i+1)^{\text{th}}$ layers, with $i=1,\ldots,11$. On ImageNet, we use an MSDNet with 23 layers with four scales each, and the i^{th} classifier operates on the $(4i+3)^{\text{th}}$ layer (with $i=1,\ldots,5$). For simplicity, the losses of all the classifiers are weighted equally during training.

Network architecture for budgeted batch setting. The MSDNets used here for the two CIFAR datasets have depths ranging from 10 to 36 layers, using the reduced network with transition layers as described in Section 4. The k^{th} classifier is attached to the $(\sum_{i=1}^{k} i)^{\text{th}}$ layer. The network used for ImageNet is the same as that described for the anytime learning setting.

ResNet$^{\text{MC}}$ and DenseNet$^{\text{MC}}$. The *ResNet^{MC}* has 62 layers, with 10 residual blocks at each spatial resolution (for three resolutions): we train early-exit classifiers on the output of the 4^{th} and 8^{th} residual blocks at each resolution, producing a total of 6 intermediate classifiers (plus the final classification layer). The *DenseNet^{MC}* consists of 52 layers with three dense blocks and each of them has 16 layers. The six intermediate classifiers are attached to the 6^{th} and 12^{th} layer in each block, also with dense connections to all previous layers in that block.

7.3 Training Details

We train all models using the framework of [9]. On the two CIFAR datasets, all models (including all baselines) are trained using stochastic gradient descent (SGD) with mini-batch size 64. We use Nesterov momentum with a momentum weight of 0.9 without dampening, and a weight decay of 10^{-4}. All models are trained for 300 epochs, with an initial learning rate of 0.1, which is divided by a factor 10 after 150 and 225 epochs. We apply the same optimization scheme to the ImageNet dataset, except that we increase the mini-batch size to 256, and all the models are trained for 90 epochs, with two learning rate drops, after 30 and 60 epochs respectively.

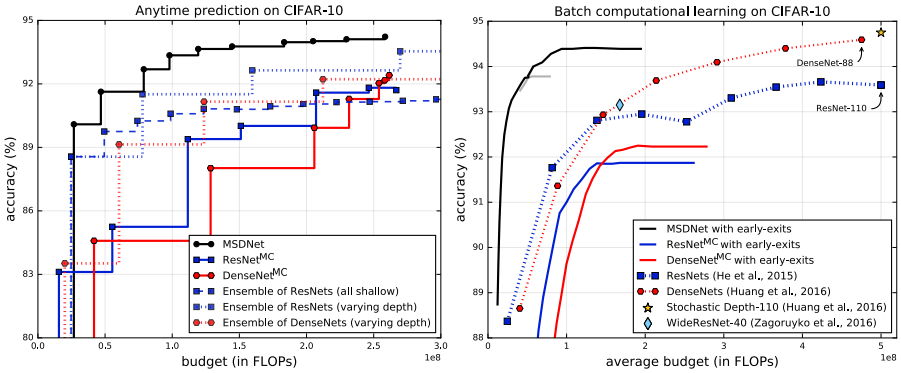


Figure 7: Classification accuracies on the CIFAR-10 dataset in the anytime-prediction setting (*left*) and the budgeted batch setting (*right*).

8 Additional Results

8.1 Results on CIFAR-10

For the CIFAR-10 dataset, we use the same MSDNets and baseline models as we used for CIFAR-100, except that the networks used here have a 10-way fully connected layer at the end. The results under the anytime learning setting and the batch computational budget setting are shown in the left and right panel of Figure 7, respectively. Similar to what we have observed from the results on CIFAR-100 and ImageNet, MSDNets outperform all the baselines by a significant margin in both settings. As in the experiments presented in the main paper, ResNet and DenseNet models with multiple intermediate classifiers perform relatively poorly.

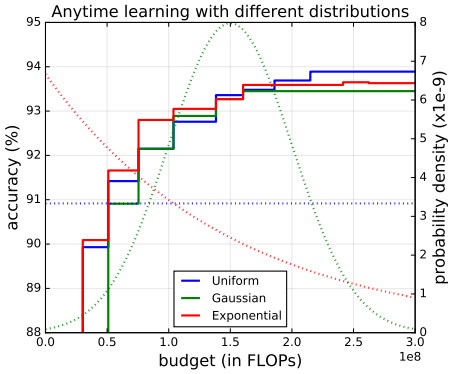


Figure 8: Classification accuracies of our anytime prediction models on the CIFAR-10 dataset using three different budget distributions $P(B)$.

8.2 Budget Distribution

To investigate to what extent the performance of MSDNets in the Anytime setting depends on the budget distribution $P(B)$, we perform experiments with three different budget distributions. The three distributions are illustrated in Figure 8 as dotted lines: a uniform distribution, an exponential distribution and a normal distribution. During training we assign weights w_k proportional to the probability that the forced exit will result in an exit after classifier f_k. The results in Figure 8 show that the accuracy distributions are somewhat shifted based on the weight assignments, however, no drastic changes occur. This result provides some reassurance that the uniform weighting is probably a good choice in many real-world applications, and that we do not need to worry too much about errors in our estimates of $p(B)$.

8.3 More Computationally Efficient DenseNets

Here, we discuss an interesting finding during our exploration of the MSDNet architecture. We found that following the DenseNet structure to design our network, *i.e.*, by keeping the number of output channels (or *growth rate*) the same at all scales, did not lead to optimal results in terms of the accuracy-speed trade-off. The main reason for this is that compared to network architectures like ResNets, the DenseNet structure tends to apply more filters on the high-resolution feature maps in the network. This helps to reduce the number of parameters in the model, but at the same time, it greatly increases the computational cost. We tried to modify DenseNets by doubling the growth rate after each transition layer, so that more filters are applied to low-resolution feature maps. It turns out that the resulting network, which we denote as DenseNet*, significantly outperform the original DenseNet in terms of computational efficiency.

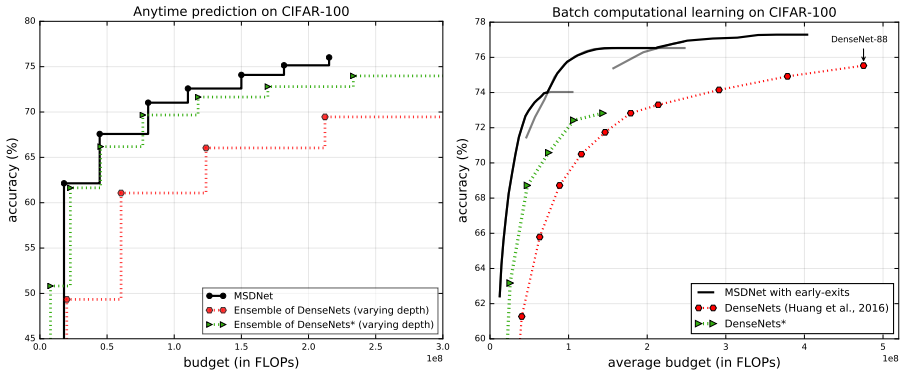


Figure 9: Test accuracy of DenseNet* on CIFAR-100 under the anytime learning setting (*left*) and the budgeted batch setting (*right*).

We experimented with DenseNet* in our two settings with test time budget constraints. The left panel of Figure 9 shows the anytime prediction performance of an ensemble of DenseNets* of varying depths. It outperforms the ensemble of original DenseNets of varying depth by a large margin, but is still slightly worse than MSDNets. In the budgeted batch budget setting, DenseNet* also leads to significantly higher accuracy over its counterpart under all budgets, but is still substantially outperformed by MSDNets.