Stochastic Downsampling for Cost-Adjustable Inference and Improved Regularization in Convolutional Networks

Jason Kuen[1], Xiangfei Kong[1], Zhe Lin[2], Gang Wang[3], Jianxiong Yin[4], Simon See[4], Yap-Peng Tan[1]
[1]Nanyang Technological University [2]Adobe Research [3]Alibaba AI Labs [4]NVIDIA
{jkuen001,xfkong,eyptan}@ntu.edu.sg, zlin@adobe.com,
gangwang6@gmail.com, {jianxiongy,ssee}@nvidia.com

Abstract

It is desirable to train convolutional networks (CNNs) to run more efficiently during inference. In many cases however, the computational budget that the system has for inference cannot be known beforehand during training, or the inference budget is dependent on the changing real-time resource availability. Thus, it is inadequate to train just inference-efficient CNNs, whose inference costs are not adjustable and cannot adapt to varied inference budgets. We propose a novel approach for cost-adjustable inference in CNNs - Stochastic Downsampling Point (SDPoint). During training, SDPoint applies feature map downsampling to a random point in the layer hierarchy, with a random downsampling ratio. The different stochastic downsampling configurations known as SDPoint instances (of the same model) have computational costs different from each other, while being trained to minimize the same prediction loss. Sharing network parameters across different instances provides significant regularization boost. During inference, one may handpick a SDPoint instance that best fits the inference budget. The effectiveness of SDPoint, as both a cost-adjustable inference approach and a regularizer, is validated through extensive experiments on image classification.

1. Introduction

Convolutional networks (CNNs) [7] have greatly accelerated the progress of many computer vision areas and applications in recent years. Despite their powerful visual representational capabilities, CNNs are bottlenecked by their immense computational demands. Recent CNN architectures such as Residual Networks (ResNets) [10, 11] and Inception [36] require billions of floating-point operations (FLOPs) to perform inference on just one single input image. Furthermore, as the amount of visual data grows, we need increasingly higher-capacity (thus higher complexity) CNNs which have shown to better utilize these large visual data compared to their lower-capacity counterparts [35].

There have been works which tackle the efficiency issues of deep CNNs, mainly by lowering numerical precisions (quantization) [16, 29, 43], pruning network weights [8, 23, 41, 12, 25], or adopting separable convolutions [18, 3, 40]. These methods result in more efficient models which have fixed inference costs (measured in floating-point operations or FLOPs). Models with fixed inference costs cannot work effectively in certain resource-constrained vision systems, where the computational budget that can be allocated to CNN inference depends on the real-time resource availability. When the system is lower in resources, it is preferable to allocate a lower budget for more efficient or cheaper inference, and vice versa. Moreover, in some cases, the exact inference budget cannot be known beforehand during training time.

As a simple solution to such a concern, one could train several CNN models such that each has a different inference cost, and then select the one that matches the given budget at inference time. However, it is extremely time-consuming to train many models, not to mention the computational storage required to store the weights of many models. In this work, we focus on CNNs whose computational costs are dynamically adjustable at inference time. A CNN with cost-adjustable inference only has to be trained once, and it allows users to control the trade-off of inference cost against network accuracy/performance. The different inference instances (each with different inference cost) are all derived from the same model parameters.

For cost-adjustable inference in CNNs, we propose a novel training method - Stochastic Downsampling Point (SDPoint). A SDPoint instance is a network configuration consisting of a unique downsampling point (layer index) in the network layer hierarchy as well as a unique downsampling ratio. As illustrated in Fig. 1, at every training iteration, a SDPoint instance is randomly selected (from a list of instances), and downsampling happens based on the downsampling point and ratio of that instance. The earlier the downsampling happens, the lower the total computational costs will be, given that spatially smaller feature maps are cheaper to process.

During inference, a SDPoint instance can be deterministically handpicked (among the SDPoint instances seen during training) to match the given inference budget. Existing approaches [22, 38, 20] to achieve cost-adjustable inference

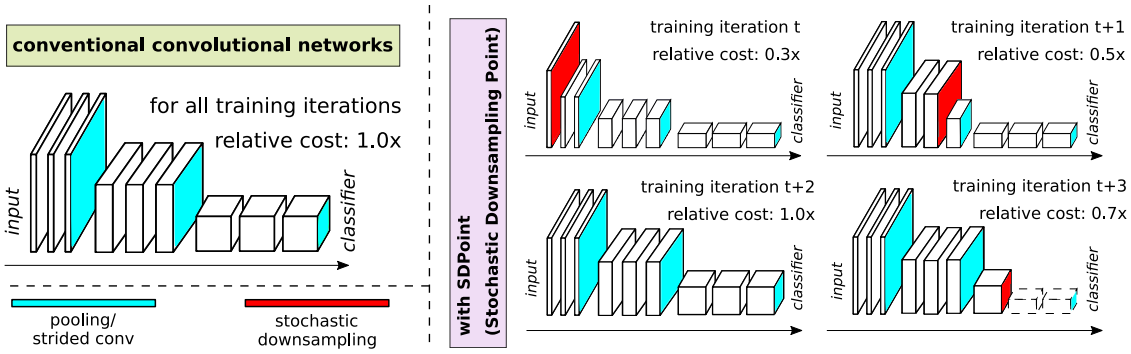

Figure 1: Progression of feature map spatial sizes during training of a (**Left**) conventional CNN, (**Right**) with SDPoint. The *costs* here refer to computational costs measured in numbers of floating-point operations (FLOPs).

in CNNs work by evaluating just subparts of the network (e.g., skipping layers or skipping subpaths), and therefore not all network parameters are utilized during cheaper inference. In contrast to existing approaches, SDPoint makes full use of all network parameters regardless of the inference costs, thus making better use of network representational capacity. Moreover, the (scale-related) parameter sharing across the SDPoint instances (each with a different downsampling and downsampling ratio) provides significant improvement in terms of model regularization. On top of these advantages, SDPoint is architecture-neutral, and it adds no parameter or training overheads. We carry out experiments on image classification with a variety of recent network architectures to validate the effectiveness of SDPoint in terms of cost-accuracy performances and regularization benefits. The code to reproduce experiments will be released.

2. Related Work

Cost-adjustable Inference: One representative method to achieve cost-adjustable inference is to train "intermediate" classifiers [22, 21, 38] which branch out of intermediate network layers. A lower inference cost can be attained by *early-exiting*, based on the intermediate classifiers' output *confidence* [22] or *entropy* [38] threshold. The lower the threshold is, the lower the inference cost will be, and vice versa. In [22], intermediate softmax classifiers are trained (second stage) after the base network has been completely trained (first stage). The downside of [22] is that the intermediate classifier losses are not backpropagated for fine-tuning the base network weights. To make the networks more aware of intermediate classifiers, BranchyNet [38] has intermediate classifiers (each with more layers per branch than [22]) and final classifier trained jointly, using a weighted sum of classification losses. Unlike these works, our SDPoint method relies on the same final classifier for different inference costs. FractalNets [20] which are CNNs designed to have many parallel subnetworks or "paths" which can be stochastically dropped for regularization during training. For cost-adjustable inference, some FractalNet's "paths" can be left out. But the path-dropping regularization gives inconsistent/marginal improvements if data augmentation is being used.

Stochastic Regularization: Our work is closely related to stochastic regularization methods which apply certain stochastic operations to network training for regularization. Dropout [34] drops network activations, while DropConnect [39] drops network weights. Stochastic Depth [15] allows nonlinear residual building blocks to be dropped during training. Swapout [33] allows elementwise Bernoulli random variables, for each of the residual network function separately. These 4 methods are similar in the way that during inference, all stochastically dropped elements (activations, weight, residual blocks) are to be present. For any of the methods, its different stochastic instances seen during training have rather comparable forward pass costs, making them unfit for cost-adjustable inference.

Multiscale parameter-sharing: Multiscale training of CNNs, first introduced by [9] is quite similar to SDPoint. In the training of [9], the network is trained with 224×224 and 180×180 images alternatively (one scale per epoch). The same idea has also been applied to CNN training for other tasks [2, 30]. While multiscale training downsamples the input images to different sizes, SDPoint only downsamples feature maps (at feature level). Downsampling at feature level encourages earlier network layers to learn to better preserve information, to compensate for loss of spatial information caused by stochastic downsampling later. This does not apply to multiscale training, where the input images are downsampled through interpolation operations which happen before network training takes place.

3. Preliminaries: Conventional CNNs with Fixed Downsampling Points

Conventionally, downsampling of feature maps happens in CNNs at several predefined fixed locations/points in the layer hierarchy, depending on the architectural designs. For example, in ResNet-50, spatial pooling (happens af-

ter the first ReLU layer, and after the last residual block) and strided convolutions (or convolution with strides > 1 which happens right after the 3rd, 7th, and 13th residual blocks) are used to achieve downsampling. Between these downsampling layers are *network stages*. Downsampling in CNNs trades low-level spatial information for richer high-level semantic information (needed for high-level visual tasks such as image classification) in a gradual fashion.

During network inference, these fixed downsampling points have to be followed exactly as how they are configured during training, for optimal accuracy performance. In this work, we go beyond fixed downsampling points - we develop a novel stochastic downsampling method named Stochastic Downsampling Point (SDPoint) which does not restrict downsampling to happen every time at same fixed points in the layer hierarchy. The proposed method is complementary to the fixed downsampling points in existing network architectures, and do not replace them. SDPoint can be simply plugged into existing network architectures, and no major architectural modifications are required.

4. Stochastic Downsampling Point

A Stochastic Downsampling Point (SDPoint) instance has a unique downsampling point $p \in \mathbb{Z}$ and a unique downsampling ratio $r \in \mathbb{R}$ which are stochastically/randomly selected during network training. A p and a r are stochastically selected at the beginning of each network training iteration, and downsampling occurs to the selected point (based on the selected ratio) for all samples in the current training mini-batch. The downsampling points and a downsampling ratios will be discussed more thoroughly in the upcoming sections. Downsampling is performed by a downsampling function $D(\cdot)$ which makes use of some downsampling operations. When the selected point falls at the lower layer in the layer hierarchy, the downsampling happens earlier (in the forward propagation), causing quicker loss of spatial information in the feature maps, but more computation savings. Conversely, spatial information can be better preserved at higher computational costs, if the stochastic downsampling happens later.

SDPoint can effectively turn the feature map spatial sizes right before prediction layers to be different from original sizes, and this could cause shape incompatibility between the prediction layer weights (as well as labels) and the convolutional outputs (before prediction layers). To prevent this, we preserve the feature map spatial size in the last network stage, regardless of stochastic downsampling taking place or not, by adjusting convolution strides and/or pooling sizes accordingly. For example, in image classification networks, we consider the global average pooling layer [24] and the final classification layer to be the last network stage. Therefore, regardless of the spatial size (variable due to SDPoint) of the incoming feature maps, we globally pool them to have spatial size of 1×1.

4.1. Downsampling Operation

As discussed in Sect. 3, the downsampling operation employed in $D(\cdot)$ can be either pooling [1] (*average* or *max* variations) or strided convolution. We opt for average pooling (the corresponding downsampling function is denoted as $D_{\text{avg}}(\cdot)$), rather than strided convolutions or max pooling for several reasons. Strided convolutions are the preferred way to do downsampling in recent network architectures, because they add extra parameters (convolution weights) and therefore improving the representational capability. In this work, we want to rule out the possible performance improvements from increase in parameter numbers (rather than the SDPoint itself). Moreover, strided convolutions with integer-valued strides cannot work well with arbitrary downsampling ratios (see Sect. 4.3). On the other hand, average pooling is preferred over max pooling in this paper due to the fact that max pooling itself is a form of non-linearity. Using max pooling as the downsampling operation could either push for a greater non-linearity in the network (positive outcome) which is unfair to the baselines, or could exacerbate the vanishing gradient problem [13] commonly associated with deep networks (negative outcome). Besides, the effectiveness of average pooling has been validated through its extensive roles in recent CNN architectures (e.g., global average pooling [24, 10], DenseNets' transition [14]).

4.2. Downsampling Points

At every training iteration, a downsampling point p for a SDPoint instance can be drawn from a discrete uniform distribution on a set of predefined downsampling point indices $P = \{0, 1, 2, ..., \mathcal{N}\text{-}1, \mathcal{N}\}$, with $\mathcal{N} + 1$ number of points. In this work, the downsampling point candidates are the points between two consecutive CNN "basic building blocks", mirroring the placements of *fixed downsampling* layers in conventional CNNs. We keep the original network (without stochastic downsampling) as an instance by assigning the index $p = 0$ to it, so that we can perform full-cost inference later. Let $F(\cdot)$ denote the function carried out by the i-th basic building block, $\mathbf{w}_i$ denote the network weights involved in the block. For a given input $\mathbf{x}_i$ and downsampling ratio r, the downsampling is carried out as following:

$$\mathbf{y}_i = D_{\text{avg}}(F(\mathbf{x}_i; \mathbf{w}_i); s_i, r) \tag{1}$$

to obtain the output $\mathbf{y}_i$. The downsampling switch denoted as $s_i \in \{\texttt{True}, \texttt{False}\}$ is turned on if $p = i$.

For non-residual CNNs (e.g., VGG-Net [32]), the basic building block comprises 3 consecutive *convolutional*, *Batch Normalization* (BN) [17], *non-linear activation* layers. On the other hand, for residual networks, residual

blocks are considered as the basic building blocks. the downsampling point p can be stochastically selected to be any point between any 2 basic building blocks in the network, where downsampling happens. Since a residual block involves two streams of information - (i.) the identity skip connection and (ii.) the non-linear function consisting of several network layers, we apply stochastic downsampling function $D_{\text{avg}}(\cdot)$ to the point right after the residual addition operation. We also experiment with Densely Connected Networks (DenseNets) [14] in this paper. For DenseNets, the SDPoint downsampling points are the points right behind each block concatenation operation, mirroring the *fixed downsampling* in DenseNets.

In principle, each mini-batch sample could have its unique downsampling point p_i (for stronger stochasticity), but due to practical reasons (e.g., training efficiency, ease of implementation), we resort to using the same p_i for all samples in a mini-batch. While it is possible to have more than one downsampling points in each training iteration, the number of possible combinations or SDPoint instances would become excessively large. Some of the instances would deviate too much from the original network, in terms of computational cost and accuracy performance. We opt for single **stochastic downsampling point** in this work.

4.3. Downsampling Ratios

We consider a set of downsampling ratios R, which the SDPoint instance can stochastically draw a downsampling ratio r from, for use at current training iteration. As with Sect. 4.2, downsampling ratios are drawn according to discrete uniform distributions. The ratios cannot be too low that they hamper the training convergence (due to parameter-sharing unfeasibility). And, we consider only a small number of downsampling ratios in R to prevent an excessive number of SDPoint instances, which would cause great difficulty in experimentally evaluating all SDPoint instances for cost-adjustable inference. A recent experimental study [26] on CNNs finds that it is sufficient to make qualitative conclusions about optimal network structure that hold for the full-sized (224×224 image resolution) ImageNet [31] classification task, by using just 128×128 (roughly half the original resolution) input images. Conceivably, the same network structure/architecture that works well with a certain image resolution is likely to work well with a resolution double/half of that. Motivated by the above-mentioned heuristics and experimental finding, we come up with the downsampling ratio set $R = \{0.5, 0.75\}$. The same ratios have also been used by [2] for "multiscale-input" semantic segmentation. The same hyperpameter R is used across all experiments in this paper.

Downsampling with such fractional downsampling ratios cannot be trivially achieved with integer-valued pooling hyperparameters. For example, pooling a 28×28 feature map to a 21×21 one (with r of 0.75 and minimal overlaps) cannot be easily done by tuning just the pooling size and stride. To this end, we adopt a spatial pooling strategy (which works along with the pooling choice in Sect. 4.1) akin to that of Spatial Pyramid Pooling [9] that generates fixed-length representation via adaptive calculations of pooling sizes and strides.

Algorithm 1 : Training with SDPoint

$P = \{0, 1, 2, ..., \mathcal{N}\text{-}1, \mathcal{N}\}$ ▷ Downsampling Points
$R = \{0.5, 0.75\}$ ▷ Downsampling Ratios
while given a training mini-batch $\mathbf{x}$ **do**
  Randomly draw p from P
  Randomly draw r from R
  $\mathbf{x}_1 = \mathbf{x}$
  for $i \in \{1, 2, ..., \mathcal{N}\text{-}1, \mathcal{N}\}$ **do** ▷ Forward pass
    if $i == p$ **then** $s_i = \texttt{True}$ **else** $s_i = \texttt{False}$
    $\mathbf{x}_{i+1} = D_{\text{avg}}(F(\mathbf{x}_i; \mathbf{w}_i); s_i, r)$
  end for
  Compute *loss* with $\mathbf{x}_{\mathcal{N}+1}$
  Backward pass
  Parameter updates
end while

4.4. Training with SDPoint

SDPoint gives rise to a new training algorithm for CNNs. The training algorithm consolidating all the previously introduced SDPoint concepts is given in Algorithm 1. $F(\cdot)$ denotes the generic nonlinear building network block in CNNs. For simplicity sake, we omit the other network layers which are not basic building blocks - typically the starting and ending layers. In a nutshell, Algorithm 1 shows that whenever a building block index i is equal to the downsampling point p, the downsampling switch s is turned on. Stochastic downsampling then happens to the output of i-th building block, with the stochastic downsampling ratio r. It is important to point out that the (stochastic) downsampling does not happen, if p is drawn to be 0, allowing the network to work in its original "unadulterated" form.

4.5. Regularization

SDPoint can be seen as a regularizer for CNNs. When stochastic downsampling takes place, the receptive field size becomes larger and it causes a sudden shrinkage of spatial information in the feature maps. The network has to learn to adapt to such variations during training, and perform **parameter-sharing** across the downsampled feature maps and the originally sized feature maps (when $p = 0$). In addition to robustness in terms of receptive field size and spatial shrinkage, SDPoint also necessitates the convolutional layers to accommodate for different "*padded pixel* to *non-padded pixel*" ratios. For example, applying a 3×3 convolutional filter (with zero-padding of 1) to a 8×8 fea-

ture map gives a padded-pixel ratio of 0.44, compared to 0.56 ratio resulted from applying the same filter to 6×6 feature map. Zero-padded pixels are quite similar to the *zero-ed out* activations caused by Dropout [34], in the sense that they both are missing values. Thus, a higher padded-pixel ratio is akin to having a higher number of *dropped-out* activations, vice versa. This form of variation provides further regularization boost. Experimentally, we find that even with the use of heavy data augmentation - such as "scale + aspect ratio" augmentation [37, 36], SDPoint can still help.

5. Cost-adjustable Inference

A network that can perform inference at different computational costs depending on the user requirements, is considered to be capable of *cost-adjustable inference*. Opting for a lower inference cost usually results in a lower prediction accuracy, and vice versa. SDPoint naturally supports cost-adjustable inference, given that SDPoint instances have varying computational costs, given the different downsampling point locations and downsampling ratios. More importantly, the instances have all been trained to minimize the same prediction loss, and this helps them to work relatively well for inference. During inference, one may handpick a SDPoint instance (with its downsampling point p and downsampling ratio r) to make the inference cost fit a particular inference budget.

5.1 Instance-Specific Batch Normalization As mentioned in Sect. 4, SDPoint instances are trained in such a way that every training mini-batch and iteration shares the same SDPoint instance. For a SDPoint instance, the prediction and loss minimization during training are based on the Batch Normalization (BN) statistics (means and standard deviations) of that particular instance. Therefore, using the BN statistics accumulated over many training iterations (and thus many different SDPoint instances) for inference causes inference-training "mismatch". A similar form of inference-training "mismatch" caused by BN statistics has also been observed by [33] in another context. The BN statistics required for one SDPoint instance should differ from that of another instance. When using the same (accumulated) BN statistics to perform cost-adjustable inference, the inference accuracies could be jeopardized.

To address the "mismatch" issue, we compute SDPoint instance-specific BN statistics, and use them for cost-adjustable inference. Disentangling the different SDPoint instances by unsharing BN statistics makes the inference more accurate. The computational storage overhead resulted from *instance-specific BN* statistics is relatively low, as BN statistics of some earlier layers can be shared among certain SDPoint instances that downsample at later layers.

6. Experiments

Experiments are carried out on image classification tasks to evaluate SDPoint. We consider image classification datasets with varying dataset scales in terms of numbers of categories/classes and sample counts: CIFAR-10 [19] (50k training images, 10k validation images, 10 classes), CIFAR-100 [19] (50k training images, 10k validation images, 100 classes), ImageNet [31] (1.2M training images, 50k validation images, 1000 classes). For inference cost comparison, we measure the model costs in terms of floating-point operation numbers (FLOPs) needed for forward propagation of single image. We treat *addition* and *multiplication* as 2 separate operations. Implementations are in PyTorch [27].

6.1. CIFAR

For CIFAR-10 and CIFAR-100, the baseline architectures are Wide-ResNet [42] (WRN-d28-w10 and WRN-d40-w4) and DenseNetBC-d40-g60 [14]. 'd', 'w', 'g' stand for the network depth, widen factor of WRN, and growth rate of DenseNetBC, respectively. The training hyperparameters (e.g., learning rates, schedules, batch sizes, augmentation) follow the ones in original papers, except for training epoch numbers which we fix to 400 for all. The original learning rate schedules still apply (e.g., learning rates are dropped at 50% and 75% of total number of training epochs). The numbers of SDPoint downsampling points ($\mathcal{N}$) for {WRN-d28-w10, WRN-d40-w4, and DenseNetBC-d40-g60} are {12, 18 ,12} respectively. As mentioned in Sect. 4.3, the downsampling ratios are drawn uniformly from $R = \{0.5, 0.75\}$.

6.1.1 Baseline Comparison: We compare SDPoint with some baseline methods related to ours, in terms of cost-adjustable inference performance. The classification error-cost performance plots on CIFAR-10 and CIFAR-100 are shown in Fig. 2. Note that for SDPoint and baseline methods, not all instances of the same model appear on the plots; if a higher-cost instance performs worse than any lower-cost instance, it is not shown. Each model (evaluated on a dataset) is trained only once to obtain its cost-error plot.

(i) Early-Exits (EE): We train models based on the WRN with intermediate classifiers (branches) which allow early-exits (EE), following the design of BranchyNet [38]. Each *network stage* in the main network has two evenly spaced branches, and the branches each have single-repetition of building block per *branch network stage*. The blocks in the branches follow the same hyperparameters (e.g., #channels) as the blocks in the original network. For cost-adjustable inference, we evaluate every branch, and make all samples "exit" at the same branch. The early-exit models have considerably more parameters than both the baseline models and SDPoint-based models. We conjecture that the rela-

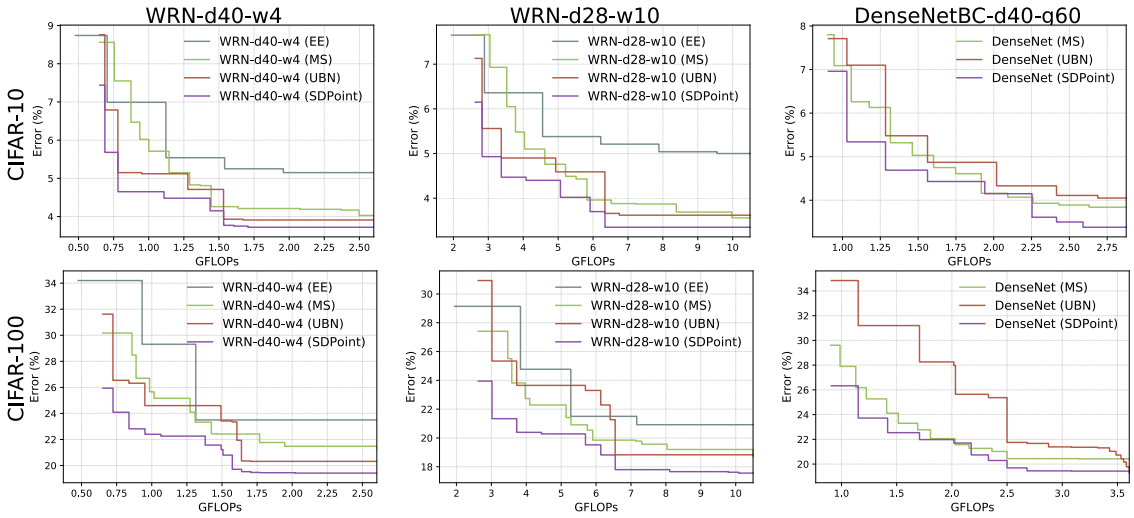

Figure 2: WRNs' and DenseNetBC's cost-error plots on CIFAR-10 (**Top**) and CIFAR-100 (**Bottom**). It is observed that models trained with SDPoint consistently outperform their non-SDPoint counterparts, given the same computational budgets.

tively worse performance of EE is due to lack of full network parameter ultilization. Also, EE forces CNN features to be classification-ready in early stage, thus causing higher layers to rely heavily on the classification-ready features, instead of learning better features on their own.

(ii) Multiscale Training (MS): Multiscale (MS) training is a baseline method inspired by [9, 2, 30]. The input images are downsampled using bilinear interpolations, to an integer-valued size randomly chosen from sizes ranging from half (16×16) to full size (32×32), with step size of 1 pixel. This is done for every training iteration, similar to SDPoint. The number of "instances" (16) resulted from multiscale training is close to the downsampling point numbers of applying SDPoint to WRNs and DenseNetBC(s). Also, the ranges of cost-adjustable inference costs among them are comparable. Instance-specific BN statistics are applied. The cost-adjustable performance of MS consistently trails behind that of SDPoint, as input downsampling causes more drastic information loss than feature map downsampling (see Sect. 2).

(iii) Uniform Batch Normalization (UBN): To validate the effectiveness of SDPoint instance-specific BN, we show the results of a SDPoint baseline whose BN statistics are averaged from many training iterations, and are uniform for all of its instances. There are consistent classification performance gaps between using UBN statistics and instance-specific BN statistics, suggesting that it is preferable to keep instance-specific statistics for inference.

6.1.2 State-of-the-art Comparison: Table 1 reports the CIFAR validation results of state-of-the-art (SOTA) ResNeXt [40] and DenseNetBC [14] models, for comparison with ours. For each SDPoint-enabled model, we show the results (giga-FLOPs, classification errors) from the best-performing SDPoint instance among its instances. Notably, WRN-d28-w10 with SDPoint is competitive to SOTA models on CIFAR-100, and it outperforms them on CIFAR-10. Overall, SDPoint considerably improves classification performance without bringing in additional parameters and computational costs, unlike the SOTA models which require about $2\times$ model complexity to attain slight improvements. In fact, the best SDPoint-enabled models on CIFAR-10 have reduced inference costs (FLOPs). We reckon that a prolonged preservation of spatial details (i.e., no early downsampling) in CNN feature maps is not crucial to a dataset with relatively low label complexity such as CIFAR-10. This reveals a drawback

Model	# Params	GFLOPs	CIFAR-10	CIFAR-100
ResNeXt-d29-c08 [40]	34.4M	10.8	3.65	17.77
ResNeXt-d29-c16 [40]	68.1M	21.4	3.58	17.31
DenseNetBC-d250-g24 [14]	15.3M	10.1	3.62	17.60
DenseNetBC-d190-g40 [14]	25.6M	18.7	3.46	**17.18**
WRN-d40-w4 [42]	8.9M	2.6	4.29	20.78
WRN-d40-w4 [42]	8.9M	2.5/2.6	3.73	19.55
with **SDPoint**			(↓ 0.56)	(↓ 1.23)
WRN-d28-w10 [42]	36.5M	10.5	3.84	18.51
WRN-d28-w10 [42]	36.5M	6.5/10.1	**3.35**	17.53
with **SDPoint**			(↓ 0.49)	(↓ 0.98)
DenseNetBC-d40-g60 [14]	4.3M	3.6	3.99	20.00
DenseNetBC-d40-g60 [14]	4.3M	2.7/3.6	3.39	19.25
with **SDPoint**			(↓ 0.60)	(↓ 0.75)

Table 1: CIFAR-10 and CIFAR-100 validation errors (%). The GFLOPs with 2 values separated by "/" are for CIFAR-10 and CIFAR-100 respectively.

Model	**Error(%)/Deterministic**	**Error(%)/Stochastic**
WRN-d28-w10	18.51	-
*with Dropout [34]	18.05	17.98
*with Swapout [33], *Linear(1,0.5)*	20.55	18.68
*with Swapout [33], *Linear(1,0.8)*	19.21	18.65
*with SDPoint	17.53	**17.20**

Table 2: *Deterministic* and *stochastic* inference results of training WRN-d28-w10 [42] with different stochastic training methods on CIFAR-100.

Model	# Params	GFLOPs	Top-1	Top-5
ResNeXt-d101-c64 [40]	∼89M	∼32	**20.4**	**5.3**
DenseNetBC-d264 [28]	∼73M	∼26	**20.4**	-
ResNeXt-d101-c32 [40]	44.3M	16.0	21.2	5.6
ResNeXt-d101-c32 [40] with Stochastic Depth [15]	44.7M	15.7	22.2 (↑ 1.0)	5.9 (↑ 0.3)
ResNeXt-d101-c32 [40] with **SDPoint**	44.3M	16.0	**20.4** (↓ 0.8)	**5.3** (↓ 0.3)
PreResNet-d101 [11]	44.7M	15.7	22.0	6.1
PreResNet-d101 [11] with Stochastic Depth [15]	44.7M	15.7	22.8 (↑ 0.8)	6.4 (↑ 0.3)
PreResNet-d101 [11] with **SDPoint**	44.7M	15.7	21.4 (↓ 0.6)	5.6 (↓ 0.5)
PreResNet-d101 [11] with SACT [4]	45.0M	11.1	24.4	7.2
PreResNet-d101 [11] with **SDPoint**	44.7M	7.7	24.3	7.2

Table 3: ImageNet top-1 and top-5 validation errors (%), with model parameter numbers and giga-FLOPs (GFLOPs).

of current practice of using CNNs in "one-size-fits-all" fashion.

6.1.3 Stochastic Training and Inference: Since our method is related to the stochastic training family, we experimentally compare SDPoint with Dropout [34] and Swapout [33] (using 2 linear decay rules suggested), with WRN-d28-w10 as the baseline and CIFAR-100 dataset. The results are given in Table 2. Stochastic inference was proposed by [33] as a way to tackle the poor interaction of Swapout/Dropout with BatchNorm. For fair comparison, we report also the results of performing stochastic inference (50 trials). In both *deterministic* and *stochastic* inference settings, we find that SDPoint outperforms the rest.

6.2. ImageNet

We consider ResNeXt-d101-c32 [40] and PreResNet-d101 [11] as baseline architectures. 'c' stands for ResNeXt's cardinality. With SDPoint, there are 33 downsampling points ($\mathcal{N}$) per model. We train the models on ImageNet-1k [31] training set, and evaluate them on the validation set (224×224 center crops). All models are trained using training hyperparameters and "scale + aspect ratio" augmentation [37] similar to [40]. Note that we do not allocate more training epochs to models with SDPoint. The cost-error plots are given in Fig. 3 and 4, for PreResNet-d101 and ResNeXt-d101-c32 respectively, along with some **fixed-cost** & **carefully designed**[1] baseline models from the same architecture families. Overall, models trained with SDPoint can roughly match the performance of baseline models in the lower-cost range, and surpass them in the upper-cost range. Notably, to obtain cost-error plots, SDPoint-enabled models only have to be trained once. The baseline models are **trained separately**,

[1] model hyperparameters are carefully chosen by the authors [11, 40] to optimize accuracy performances under some budget constraints.

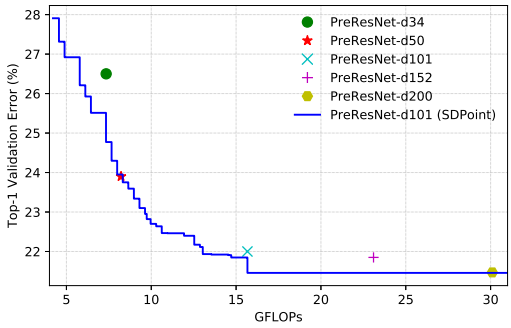

Figure 3: PreResNets' [11] cost-error plots on ImageNet. PreResNet-d101 (SDPoint) only has to be trained once (as a single model), while the baseline models (without SDPoint) has to be trained separately with huge training and storage costs.

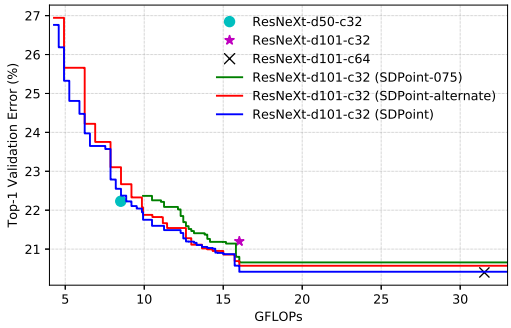

Figure 4: ResNeXts' [40] cost-error plots on ImageNet. Like Fig. 3, any of ResNeXt-d101-c32 (SDPoint..) only has to be trained once (as a single model).

resulting in a huge total number of epochs (#models × #epochs per model) and storage cost.

6.2.1 Ablation Study: We study the effects of choice of SDPoint downsampling points and downsampling ratios on cost-adjustable inference performance. For this, we train a ResNeXt-d101-c32 with default SDPoint hyperparameters (downsampling points at the end of every residual block, downsampling ratios of {0.5,0.75}), as well as 2 baseline models with (i) downsampling points at the end of every other residual block dubbed ***alternate*** (ii) downsampling ratio of just {0.75} dubbed ***075***. They are shown on Fig. 4. Either removing the 0.5 downsampling ratio or alternating blocks for downsampling gives worse results, due to reduced stochasticity (and regularization strengths).

6.2.2 State-of-the-art Comparison: We compare our models with SOTA ResNeXt-d101-c64 [40] and DenseNetBC-d264-g48 [28] models in Table 3. SDPoint pushes the top-1 and top-5 validation errors of ResNeXt-d101-c32 down to 20.4% and 5.3% respectively, which are (previously) only attainable by SOTA models with roughly 2× inference costs and parameter counts. We also display

Figure 5: Some Imagenet validation examples grouped according to the **minimum** inference costs (FLOPs) required by ResNeXt-d101-c32 (with SDPoint) to classify them correctly, in terms of top-5 accuracy. The ground-truth label names are shown below their corresponding images.

the results (and mean FLOPs) of Spatially Adaptive Computation Time (SACT) [4] paired with PreResNet-d101, and compare it to a SDPoint instance of our PreResNet-d101 that achieves similar classification errors. SDPoint merely needs 69% of FLOPs needed by SACT to achieve similar results. SACT saves computation by skipping layers (and network parameters) for certain locations in feature maps according to learned policy and inputs, while SDPoint downsamples feature maps to save computation (but makes full use of network parameters & capacity during inference). We contend that in cost-accuracy trade-off for inference, reducing feature map spatial sizes is less harmful to accuracy than skipping network parameters/layers. As a regularization study, we train the 2 baseline models with Stochastic Depth [15] (using suggested decay rule), and find that they considerably degrade the classification performance unlike SDPoint.

6.2.3 Analysis: We provide some analyses of ResNeXt-d101-c32 (trained with SDPoint on ImageNet) with regards to certain aspects of downsampling and SDPoint.

Cost-dependent misclassifications: We group ImageNet validation images (which are correctly classified with full inference cost) according to the minimum inference costs required to classify them correctly, and present some examples on Fig. 5. More difficult examples that require higher inference costs (9.9, 16.0 GFLOPs) to be classified correctly, generally have size-dominant interfering objects/scenes (e.g., `hair dryer`, `cab`, `caldron`, `cock`, `tench`), in contrast to the easier examples (4.3 GFLOPs). Intuitively, pooling-based downsampling causes more information loss to smaller objects than to larger (size-dominant) objects, especially when it occurs at some early layer, where the semantic/context information is still relatively weak to distinguish objects of interest from interfering objects. So, for those difficult examples, it makes sense to preserve spatially informative object details longer in the CNN layer hierarchy, and downsample the feature maps only after they are semantically rich enough.

Scale sensitivity: Training CNNs with SDPoint involves stochastic downsampling of intermediate feature maps, which we hypothesize to be beneficial for scale sensitivity/invariance, as mentioned in Sect. 4.5. To validate this hypothesis, we vary the *pre-cropping*[2] sizes of ImageNet validation images in the range of $256, ..., 352$ with step size of 16, resulting in 7 *pre-cropping* sizes. For every *pre-cropping* size, 224×224 center image regions are cropped out for evaluation. The models involved are SDPoint-enabled ResNeXt-d101-c32, and the baseline without SDPoint. We compute the mean of all pairwise cosine similarities (a total of 21 pairs) resulted from the different *pre-cropping* sizes, in terms of **ImageNet 1k-class probability scores**. This is done for entire ImageNet validation set. The pairwise cosine-similarity mean obtained for baseline model is **0.944**, while for the SDPoint-enabled model, it is **0.961**. A higher cosine similarity is a strong indicator of the model being less sensitive to scales. This demonstrates that SDPoint can indeed benefit CNNs, in terms of scale sensitivity.

7. Conclusion

We propose Stochastic Downsampling Point (SDPoint), a novel approach to train CNNs by downsampling intermediate feature maps. At no extra parameter and training costs, SDPoint facilitates effective cost-adjustable inference and greatly improves network regularization (thus accuracy performance). Through experiments, we additionally find out that SDPoint can help to identify more optimal (yet less costly) sub-networks (Sect. 6.1.2), sort input examples by various levels of classification difficulties (Fig. 5), and making CNNs less scale-sensitive (Sect. 6.2.3).

Acknowledgement: This research was carried out at the Rapid-Rich Object Search (ROSE) Lab, Nanyang Technological University, Singapore. The ROSE Lab is supported by the National Research Foundation, Singapore, and the Infocomm Media Development Authority, Singapore.

[2]It is a standard practice [10, 11, 40, 14] to resize images to a shorter side of 256 (***pre-cropping*** size) before doing 224×224 center-cropping.

References

[1] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *International Conference on Machine Learning (ICML)*, 2010. 3

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017. 2, 4, 6

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[4] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 7, 8

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

[6] J. Gu, G. Wang, J. Cai, and T. Chen. An empirical study of language cnn for image captioning. In *International Conference on Computer Vision (ICCV)*, 2017.

[7] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 2017. 1

[8] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Conference on Neural Information Processing Systems (NIPS)*, 2015. 1

[9] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision (ECCV)*, 2014. 2, 4, 6

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 3, 8

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 7, 8

[12] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, 2017. 1

[13] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991. 3

[14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 4, 5, 6, 8

[15] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision (ECCV*, pages 646–661. Springer, 2016. 2, 7, 8

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2016. 1

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015. 3

[18] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014. 1

[19] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 5

[20] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 2

[21] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics (AISTATS*, 2015. 2

[22] S. Leroux, S. Bohez, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt. Resource-constrained classification using a cascade of neural network layers. In *International Joint Conference on Neural Networks (IJCNN)*, 2015. 1, 2

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[24] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014. 3

[25] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *International Conference on Computer Vision (ICCV)*, 2017. 1

[26] D. Mishkin, N. Sergievskiy, and J. Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding (CVIU)*, 2017. 4

[27] A. Paszke, S. Gross, S. Chintala, and G. Chanan. Pytorch. http://pytorch.org/. 5

[28] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017. 7

[29] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016. 1

[30] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 6

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV*, 115(3):211–252, 2015. 4, 5, 7

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 3

[33] S. Singh, D. Hoiem, and D. Forsyth. Swapout: Learning an ensemble of deep architectures. In *Conference on Neural Information Processing Systems (NIPS)*, pages 28–36, 2016. 2, 5, 6, 7

[34] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014. 2, 5, 6, 7

[35] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision (ICCV)*, 2017. 1

[36] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI Conference on Artificial Intelligence*, 2017. 1, 5

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 5, 7

[38] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, 2016. 1, 2, 5

[39] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML)*, 2013. 2

[40] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 6, 7, 8

[41] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[42] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016. 5, 6

[43] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017. 1