NetAdaptV2: Efficient Neural Architecture Search with Fast Super-Network Training and Architecture Optimization

Tien-Ju Yang, Yi-Lun Liao, Vivienne Sze
Massachusetts Institute of Technology
{tjy,ylliao,sze}@mit.edu

Abstract

Neural architecture search (NAS) typically consists of three main steps: training a super-network, training and evaluating sampled deep neural networks (DNNs), and training the discovered DNN. Most of the existing efforts speed up some steps at the cost of a significant slowdown of other steps or sacrificing the support of non-differentiable search metrics. The unbalanced reduction in the time spent per step limits the total search time reduction, and the inability to support non-differentiable search metrics limits the performance of discovered DNNs.

In this paper, we present NetAdaptV2 with three innovations to better balance the time spent for each step while supporting non-differentiable search metrics. First, we propose channel-level bypass connections that merge network depth and layer width into a single search dimension to reduce the time for training and evaluating sampled DNNs. Second, ordered dropout is proposed to train multiple DNNs in a single forward-backward pass to decrease the time for training a super-network. Third, we propose the multi-layer coordinate descent optimizer that considers the interplay of multiple layers in each iteration of optimization to improve the performance of discovered DNNs while supporting non-differentiable search metrics. With these innovations, NetAdaptV2 reduces the total search time by up to $5.8\times$ *on ImageNet and* $2.4\times$ *on NYU Depth V2, respectively, and discovers DNNs with better accuracy-latency/accuracy-MAC trade-offs than state-of-the-art NAS works. Moreover, the discovered DNN outperforms NAS-discovered MobileNetV3 by 1.8% higher top-1 accuracy with the same latency.*[1]

1. Introduction

Neural architecture search (NAS) applies machine learning to automatically discover deep neural networks (DNNs) with better performance (e.g., better accuracy-latency trade-offs) by sampling the search space, which is the union of all discoverable DNNs. The search time is one key metric for NAS algorithms, which accounts for three steps: 1) training a *super-network*, whose weights are shared by all the DNNs in the search space and trained by minimizing the loss across them, 2) training and evaluating sampled DNNs (referred to as *samples*), and 3) training the discovered DNN. Another important metric for NAS is whether it supports non-differentiable search metrics such as hardware metrics (e.g., latency and energy). Incorporating hardware metrics into NAS is the key to improving the performance of the discovered DNNs [1–5].

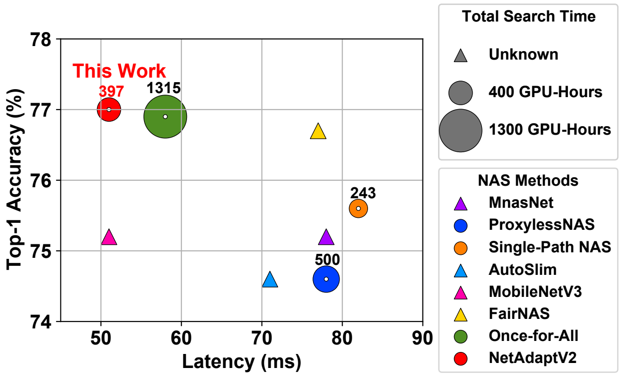


Figure 1: The comparison between NetAdaptV2 and related works. The number above a marker is the corresponding total search time measured on NVIDIA V100 GPUs.

There is usually a trade-off between the time spent for the three steps and the support of non-differentiable search metrics. For example, early reinforcement-learning-based NAS methods [2, 6, 7] suffer from the long time for training and evaluating samples. Using a super-network [8–16] solves this problem, but super-network training is typically time-consuming and becomes the new time bottleneck. The gradient-based methods [3, 17–24] reduce the time for training a super-network and training and evaluating samples at the cost of sacrificing the support of non-differentiable search metrics. In summary, many existing works either have an unbalanced reduction in the time spent per step (i.e., optimizing some steps at the cost of a significant increase in the time for other steps), which still leads to a long *total* search time,

[1] The project website: http://netadapt.mit.edu.

or are unable to support non-differentiable search metrics, which limits the performance of the discovered DNNs.

In this paper, we propose an efficient NAS algorithm, NetAdaptV2, to significantly reduce the *total* search time by introducing three innovations to *better balance* the reduction in the time spent per step while supporting non-differentiable search metrics:

Channel-level bypass connections (mainly reduce the time for training and evaluating samples, Sec. 2.2): Early NAS works only search for DNNs with different numbers of filters (referred to as *layer widths*). To improve the performance of the discovered DNN, more recent works search for DNNs with different numbers of layers (referred to as *network depths*) in addition to different layer widths at the cost of training and evaluating more samples because network depths and layer widths are usually considered independently. In NetAdaptV2, we propose *channel-level bypass connections* to merge network depth and layer width into a single search dimension, which requires only searching for layer width and hence reduces the number of samples.

Ordered dropout (mainly reduces the time for training a super-network, Sec. 2.3): We adopt the idea of super-network to reduce the time for training and evaluating samples. In previous works, *each* DNN in the search space requires one forward-backward pass to train. As a result, training multiple DNNs in the search space requires multiple forward-backward passes, which results in a long training time. To address the problem, we propose *ordered dropout* to jointly train multiple DNNs in a *single* forward-backward pass, which decreases the required number of forward-backward passes for a given number of DNNs and hence the time for training a super-network.

Multi-layer coordinate descent optimizer (mainly reduces the time for training and evaluating samples and supports non-differentiable search metrics, Sec. 2.4): NetAdaptV1 [1] and MobileNetV3 [25], which utilizes NetAdaptV1, have demonstrated the effectiveness of the single-layer coordinate descent (SCD) optimizer [26] in discovering high-performance DNN architectures. The SCD optimizer supports both differentiable and non-differentiable search metrics and has only a few interpretable hyper-parameters that need to be tuned, such as the per-iteration resource reduction. However, there are two shortcomings of the SCD optimizer. First, it only considers one layer per optimization iteration. Failing to consider the joint effect of multiple layers may lead to a worse decision and hence sub-optimal performance. Second, the per-iteration resource reduction (e.g., latency reduction) is limited by the layer with the smallest resource consumption (e.g., latency). It may take a large number of iterations to search for a very deep network because the per-iteration resource reduction is relatively small compared with the network resource consumption. To address these shortcomings, we propose the *multi-layer coordinate descent (MCD) optimizer* that considers multiple layers per optimization iteration to improve performance while reducing search time and preserving the support of non-differentiable search metrics.

Fig. 1 (and Table 1) compares NetAdaptV2 with related works. NetAdaptV2 can reduce the search time by up to $5.8\times$ and $2.4\times$ on ImageNet [27] and NYU Depth V2 [28] respectively and discover DNNs with better performance than state-of-the-art NAS works. Moreover, compared to NAS-discovered MobileNetV3 [25], the discovered DNN has 1.8% higher accuracy with the same latency.

2. Methodology: NetAdaptV2

2.1. Algorithm Overview

NetAdaptV2 searches for DNNs with different network depths, layer widths, and kernel sizes. The proposed **channel-level bypass connections (CBCs, Sec. 2.2)** enables NetAdaptV2 to discover DNNs with different network depths and layer widths by only searching layer widths because different network depths become the natural results of setting the widths of some layers to zero. To search kernel sizes, NetAdaptV2 uses the superkernel method [12, 21, 22].

Fig. 2 illustrates the algorithm flow of NetAdaptV2. It takes an initial network and uses its *sub-networks*, which can be obtained by shrinking some layers in the initial network, to construct the search space. In other words, a sample in NetAdaptV2 is a sub-network of the initial network. Because the optimizer needs the accuracy of samples for comparing their performance, the samples need to be trained. NetAdaptV2 adopts the concept of jointly training all sub-networks with shared weights by training a super-network, which has the same architecture as the initial network and contains these shared weights. We use CBCs, the proposed **ordered dropout (Sec. 2.3)**, and superkernel [12, 21, 22] to efficiently train the super-network that contains sub-networks with different layer widths, network depths, and kernel sizes. After training the super-network, the proposed **multi-layer coordinate descent optimizer (Sec. 2.4)** is used to discover the architectures of DNNs with optimal performance. The optimizer iteratively samples the search space to generate a bunch of samples and determines the next set of samples based on the performance of the current ones. This process continues until the given stop criteria are met (e.g., the latency is smaller than 30ms), and the discovered DNN is then trained until convergence. Because of the trained super-network, the accuracy of samples can be directly evaluated by using the shared weights without any further training.

2.2. Channel-Level Bypass Connections

Previous NAS algorithms generally treat network depth and layer width as two different search dimensions. The reason is evident in the following example. If we remove

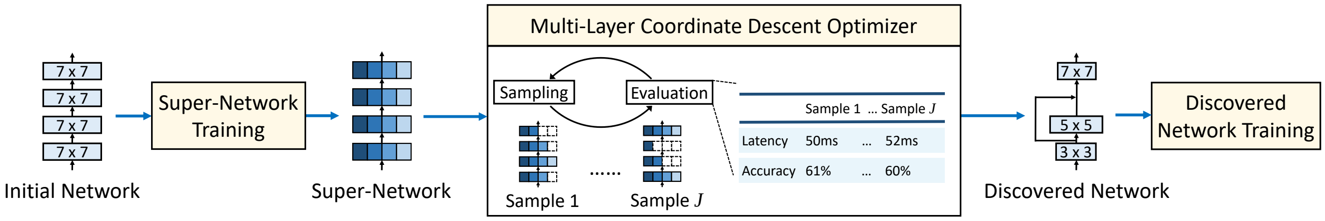


Figure 2: The algorithm flow of the proposed NetAdaptV2.

a filter from a layer, we reduce the number of output channels by one. As a result, if we remove all the filters, there are no output channels for the next layer, which breaks the DNN into two disconnected parts. Hence, reducing layer widths typically cannot be used to reduce network depths. To address this, we need an approach that keeps the network connectivity while removing filters; this is achieved by our proposed channel-level bypass connections (CBCs).

The high-level concept of CBCs is "when a filter is removed, an input channel is bypassed to maintain the same number of output channels". In this way, we can preserve the network connectivity when all filters are removed from a layer. Assuming the target layer in the initial network has C input channels, T filters, and Z output channels[2], we gradually remove filters from the layer, where there are M filters remaining. Fig. 3 illustrates how CBCs handle three cases in this process based on the relationship between the number of input channels (C) and the initial number of filters (T) (only M changes, and C and T are fixed):

- Case 1, $C = T$ (Fig. 3a): When the i-th filter is removed, we bypass the i-th input channel, so the number of output channels (Z) can be kept the same. When all the filters are removed ($M = 0$), all the input channels are bypassed, which is the same as removing the layer.
- Case 2, $C < T$ (Fig. 3b): We do not bypass input channels at the beginning of filter removal because we have more filters than input channels (i.e., $M > C$) and there are no corresponding input channels to bypass. The bypass process starts when there are fewer filters than input channels ($M < C$), which becomes case 1.
- Case 3, $C > T$ (Fig. 3c): When the i-th filter is removed, we bypass the i-th input channel. The extra ($C - T$) input channels are not used for the bypass.

These three cases can be summarized in a rule: when the i-th filter is removed, the corresponding i-th input channel is bypassed if that input channel exists. Therefore, the number of output channels (Z) when using CBCs can be computed by $Z = max(min(C, T), M)$. The proposed CBCs can be efficiently trained when combined with the proposed ordered dropout, as discussed in Sec. 2.3.

As a more advanced usage of T, we can treat T as a hyperparameter. Please note that we only change M, and C and T are fixed. From the formulation $Z = max(min(C, T), M)$, we can observe that the function of T is limiting the number of bypassed input channels and hence the minimum number of output channels (Z). If we set $T \geq C$ to allow all C input channels to be bypassed, the formulation becomes $Z = max(C, M)$, and the minimum number of output channels is C. If we set $T < C$ to only allow T input channels to be bypassed, the formulation becomes $Z = max(T, M)$, and the minimum number of output channels is T.

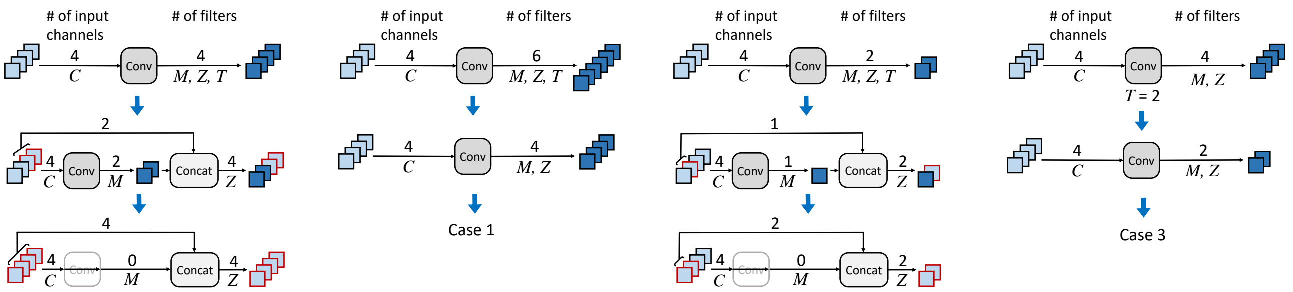
Setting $T < C$ enables generating the bottleneck, where we have fewer output channels than input channels ($Z < C$). The bottleneck has been shown to be effective in improving the accuracy-efficiency (e.g., accuracy-latency) trade-offs in MobileNetV2 [29]/V3 [25]. We take the case 1 as an example. In Fig. 3a, we can observe that the number of output channels is always 4, which is the same as the number of input channels ($Z = C = 4$) no matter how many filters are removed. Therefore, the bottleneck cannot be generated. In contrast, if we set T to 2 as the case 4 in Fig. 3d, no input channels are bypassed until we remove the first two filters because $Z = max(min(4, 2), 2) = 2$. After that, it becomes the case 3 in Fig. 3c, which forms a bottleneck.

2.3. Ordered Dropout

Training the super-network involves joint training of multiple sub-networks with shared weights. After the super-network is trained, comparing sub-networks of the super-network (i.e., samples) only requires their *relative* accuracy (e.g., sub-network A has higher accuracy than sub-network B). Generally speaking, the more sub-networks are trained, the better the relative accuracy of sub-networks will be. However, previous works usually require one forward-backward pass for training one sub-network. As a result, training more sub-networks requires more forward-backward passes and hence increases the training time.

To address this problem, we propose ordered dropout (OD) to enable training N sub-networks in a single forward-backward pass with a batch of N images. OD is inserted after each convolutional layer in the super-network and zeros

[2] If we do not use CBCs, Z is equal to T.



(a) Case 1: Same number of input channels and initial filters. $(C = T = 4)$

(b) Case 2: Fewer input channels than initial filters. $(C = 4 < T = 6)$

(c) Case 3: More input channels than initial filters. $(C = 4 > T = 2)$

(d) Case 4: Same number of input channels and initial filters but with a given T. $(C = 4, T = 2)$

Figure 3: An illustration of how CBCs handle different cases based on the relationship between the number of input channels (C) and the initial number of filters (T) (only the number of filters remaining (M) changes, and C and T are fixed). For each case, it shows how the architecture changes with more filters removed from top to bottom. The numbers above lines correspond to the letters below lines. Please note that the number of output channels (Z) will never become zero.

out *different output channels for different images in a batch*. As shown in Fig. 4, OD simulates different layer widths with a constant number of output channels. Unlike the standard dropout [30] that zeros out a random subset of channels regardless of their positions, OD always zeros out the last channels to simulate removing the last filters. As a result, while sampling the search space, we can simply drop the last filters from the super-network to evaluate samples without other operations like sorting and avoid a mismatch between training and evaluation.

When combined with the proposed CBCs, OD can train sub-networks with different network depths by zeroing out *all* output channels of some layers to simulate layer removal. As shown in Fig. 5, to simulate CBCs, there is another OD in the bypass path (upper) during training, which zeros out the *complement* set of the channels zeroed by the OD in the corresponding convolution path (lower).

Because NAS only requires the relative accuracy of samples, we can decrease the number of training iterations to further reduce the super-network training time. Moreover, for each layer, we sample each layer width almost the same number of times in a forward-backward pass to avoid biasing towards any specific layer widths.

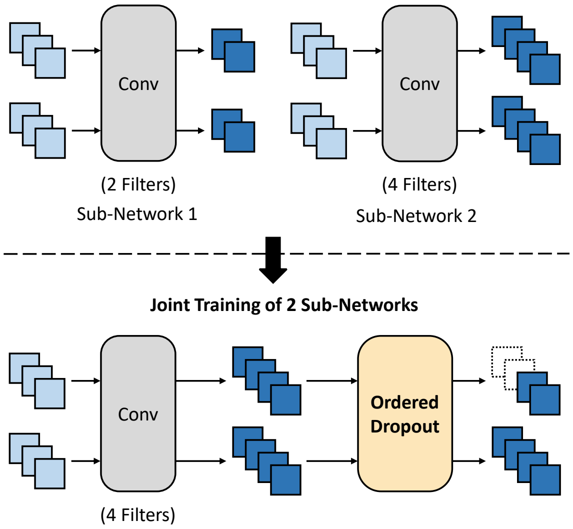


Figure 4: An illustration of how NetAdaptV2 uses the proposed ordered dropout to train two different sub-networks in a single forward-backward pass. The ordered dropout is inserted after each convolutional layer to simulate different layer widths by zeroing out some channels of activations. Note that all the sub-networks share the same set of weights.

2.4. Multi-Layer Coordinate Descent Optimizer

The single-layer coordinate descent (SCD) optimizer [26], used in NetAdaptV1 [1], is a simple-yet-effective optimizer with the advantages such as supporting both differentiable and non-differentiable search metrics and having only a few interpretable hyper-parameters that need to be tuned. The SCD optimizer runs an iterative optimization. It starts from the super-network and gradually reduces its latency (or other search metrics such as multiply-accumulate operations and energy). In each iteration, the SCD optimizer generates K samples if the super-network has K layers. The k-th sample is generated by shrinking (e.g., removing filters) the k-th layer in the best sample from the previous iteration to reduce its latency by a given amount. This amount is referred to as *per-iteration resource reduction* and may change from one iteration to another. Then, the sample with the best performance (e.g., accuracy-latency trade-off) will be chosen

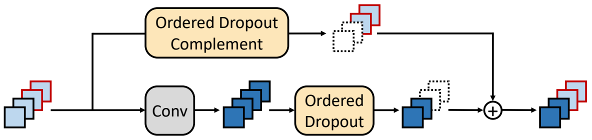


Figure 5: An illustration of how NetAdaptV2 uses the proposed channel-level bypass connections and ordered dropout to train a super-network that supports searching different layer widths and network depths.

and used for the next iteration. The optimization terminates when the target latency is met, and the sample with the best performance in the last iteration is the discovered DNN.

The shortcoming of the SCD optimizer is that it generates samples by shrinking only one layer per iteration. This property causes two problems. First, it does not consider the interplay of multiple layers when generating samples in an iteration, which may lead to sub-optimal performance of discovered DNNs. Second, it may take many iterations to search for very deep networks because the layer with the lowest latency limits the maximum value of the per-iteration resource reduction; the lowest latency of a layer becomes small when the super-network is deep. To address these problems, we propose the *multi-layer coordinate descent (MCD) optimizer*. It generates J samples per iteration, where each sample is obtained by randomly shrinking L layers from the previous best sample. In NetAdaptV2, shrinking a layer involves removing filters, reducing the kernel size, or both. Compared with the SCD optimizer, the MCD optimizer considers the interplay of L layers in each iteration so that the performance of the discovered DNN can be improved. Moreover, it enables using a larger per-iteration resource reduction (i.e., up to the total latency of L layers) to reduce the number of iterations and hence the search time.

3. Related Works

Reinforcement-learning-based methods [2, 4, 6, 7, 31] demonstrate the ability of neural architecture search for designing high-performance DNNs. However, its search time is longer than the following works due to the long time for training samples individually. Gradient-based methods [3, 17–19, 21–24] successfully discover high-performance DNNs with a much shorter search time, but they can only support differentiable search metrics. NetAdaptV1 [1] proposes a single-layer coordinate descent optimizer that can support both differentiable and non-differentiable search metrics and was used to design state-of-the-art MobileNetV3 [25]. However, shrinking only one layer for generating each sample and the long time for training samples individually become its bottleneck of search time. The idea of super-network training [10–12], which jointly trains all the sub-networks in the search space, is proposed to reduce the time for training and evaluating samples and training the discovered DNN at the cost of a significant increase in the time for training a super-network. Moreover, network depth and layer width are usually considered separately in related works. The proposed NetAdaptV2 addresses all these problems at the same time by reducing the time for training a super-network, training and evaluating samples, and training the discovered DNN in balance while supporting non-differentiable search metrics.

The algorithm flow of NetAdaptV2 is most similar to NetAdaptV1 [1], as shown in Fig. 2. Compared with NetAdaptV2, NetAdaptV1 does not train a super-network but train each sample individually. Moreover, NetAdaptV1 considers only one layer per optimization iteration and different layer widths, but NetAdaptV2 considers multiple layers per optimization iteration and different layer widths, network depths, and kernel sizes. Therefore, NetAdaptV2 is both faster and more effective than NetAdaptV1, as shown in Sec. 4.1.4 and 4.2.

For the methodology, the proposed ordered dropout is most similar to the partial channel connections [24]. However, they are different in the purpose and the ability to expand the search space. Partial channel connections aim to reduce memory consumption while training a DNN with multiple parallel paths by removing some channels. The number of channels removed is constant during training. Moreover, this number is manually chosen. As a result, partial channel connections do not expand the search space. In contrast, the proposed ordered dropout is designed for jointly training multiple sub-networks and expanding the search space. The number of channels removed (i.e., zeroed out) varies from image to image and from one training iteration to another during training to simulate different subnetworks. Moreover, the final number of channels removed (i.e., the discovered architecture) is searched. Therefore, the proposed ordered dropout expands the search space in terms of layer width as well as network depth when the proposed channel-level bypass connections are used.

4. Experiment Results

We apply NetAdaptV2 on two applications (image classification and depth estimation) and two search metrics (latency and multiply-accumulate operations (MACs)) to demonstrate the effectiveness and versatility of NetAdaptV2 across different operating conditions. We also perform an ablation study to show the impact of each of the proposed techniques and the associated hyper-parameters.

4.1. Image Classification

4.1.1 Experiment Setup

We use latency or MACs to guide NetAdaptV2. The latency is measured on a Google Pixel 1 CPU. The search time is reported in GPU-hours and measured on V100 GPUs.

The dataset is ImageNet [32]. We reserve 10K images in the training set for comparing the accuracy of samples and train the super-network with the rest of the training images. The accuracy of the discovered DNN is reported on the validation set, which was not seen during the search. The initial network is based on MobileNetV3 [25]. The maximum learning rate is 0.064 decayed by 0.963 every 3 epochs when the batch size is 1024. The learning rate scales linearly with the batch size [33]. The optimizer is RMSProp with an $\ell 2$ weight decay of 10^{-5}. The dropout rate is 0.2. The decay rate of the exponential moving average is 0.9998. The batch size is 1024 for training the super-network, 2048 for training the latency-guided discovered DNN, and 1536 for training the MAC-guided discovered DNN.

The multi-layer coordinate descent (MCD) optimizer generates 200 samples per iteration ($J = 200$). For the latency-guided experiment, each sample is obtained by randomly shrinking 10 layers ($L = 10$) from the best sample in the previous iteration. We reduce the latency by 3% in the first iteration (i.e., initial resource reduction) and decay the resource reduction by 0.98 every iteration. For the MAC-guided experiment, each sample is obtained by randomly shrinking 15 layers ($L = 15$) from the best sample in the previous iteration. We reduce the MACs by 2.5% in the first iteration and decay the resource reduction by 0.98 every iteration. More details are included in the appendix.

4.1.2 Latency-Guided Search Result

The results of NetAdaptV2 guided by latency and related works are summarized in Table 1. Compared with the state-of-the-art (SOTA) NAS algorithms [11, 12], NetAdaptV2 reduces the search time by up to 5.8× and discovers DNNs with better accuracy-latency/accuracy-MAC trade-offs. The reduced search time is the result of the much more balanced time spent per step. Compared with the NAS algorithms in the class of hundreds of GPU-hours, ProxylessNAS [3] and Single-Path NAS [22], NetAdaptV2 outperforms them without sacrificing the support of non-differentiable search metrics. NetAdaptV2 achieves either 2.4% higher accuracy with 1.5× lower latency or 1.4% higher accuracy with 1.6× lower latency. Compared with SOTA NAS-discovered MobileNetV3 [25], NetAdaptV2 achieves 1.8% higher accuracy with the same latency in around 50 hours on eight GPUs. We estimate the CO_2 emission of NetAdaptV2 based on [34]. NetAdaptV2 discovers DNNs with better accuracy-latency/accuracy-MAC trade-offs with low CO_2 emission.

4.1.3 MAC-Guided Search Result

We present the result of NetAdaptV2 guided by MACs and compare it with related works in Table 2. For a fair comparison, AutoAugment [36] and stochastic depth [37] with a survival probability of 0.8 are used for training the discovered network, which results in a longer time for training the discovered DNN. NetAdaptV2 achieves comparable accuracy-MAC trade-offs to NSGANetV2-m [38] while the search time is 2.6× lower. Moreover, the discovered DNN also outperforms EfficientNet-B0 [31] and MixNet-M [39] by up to 1.5% higher top-1 accuracy with fewer MACs.

4.1.4 Ablation Study

The ablation study employs the experiment setup outlined in Sec. 4.1.1 unless otherwise stated. To speed up training the discovered networks, the distillation model is smaller.

- **Impact of Ordered Dropout**

To study the impact of the proposed ordered dropout (OD), we do not use channel-level bypass connections (CBCs) and multi-layer coordinate descent (MCD) optimizer in this experiment. When we further remove the usage of OD, NetAdaptV2 becomes the same as NetAdaptV1 [1], where each sample needs to be trained for four epochs by following the setting of NetAdaptV1. To speed up the execution of NetAdaptV1, we use a shallower network, MobileNetV1 [40], in this experiment instead. Table 3 shows that using OD reduces the search time by 3.3× while achieving the same accuracy-latency trade-off. If we only consider the time for training a super-network and training and evaluating samples, which are affected by OD, the time reduction is 10.4×.

- **Impact of Channel-Level Bypass Connections**

The proposed channel-level bypass connections (CBCs) enable NetAdaptV2 to search for different network depths. Table 4 shows that CBCs can improve the accuracy by 0.3%. The difference is more significant when we target at lower latency, as shown in the ablation study on MobileNetV1 in the appendix, because the ability to remove layers becomes more critical for maintaining accuracy.

- **Impact of Multi-Layer Coordinate Descent Optimizer**

The proposed multi-layer coordinate descent (MCD) optimizer improves the performance of the discovered DNN by considering the joint effect of multiple layers per optimization iteration. Table 4 shows that using the MCD optimizer further improves the accuracy by 0.4%.

- **Impact of Resource Reduction and Number of Samples**

The two main hyper-parameters of the MCD optimizer are the per-iteration resource reduction, which is defined by an initial resource reduction and a decay rate, and the number of samples per iteration (J). They influence the accuracy of the discovered networks and the search time. Table 5 summarizes the accuracy of the 51ms discovered networks when using different initial latency reductions (with a fixed decay of 0.98 per iteration) and different numbers of samples.

The first experiment is fixing the number of samples per iteration and increasing the initial latency reduction from 1.5% to 6.0%, which gradually reduces the time for evaluating samples. The result shows that as long as the latency

Method	Top-1 Accuracy (%)	Latency (ms)	MAC (M)	Search Time (GPU-Hours)	Non-Diff. Metrics	CO_2 Emission (lbs)
MnasNet [2]	75.2	78	312	-	✓	-
ProxylessNAS [3]	74.6	78	320	500		142
Single-Path NAS [22]	75.6	82	-	243* (24 (TPU V3), 0, 219)		69
AutoSlim [10]	74.6	71	315	-	✓	-
FBNet [19]	74.9	-	375	-		-
MobileNetV3 [25]	75.2	51	219	-	✓	-
FairNAS [35]	76.7	77	325	-	✓	-
Once-for-All [11]	76.9	58	230	1315 (1200, 40, 75)	✓	374
BigNAS [12]	76.5	-	242	2304 (TPU V3) (2304, -, 0)	✓	655
NetAdaptV2 (Guided by Latency)	**77.0**	**51**	**225**	**397 (167, 24, 206)**	✓	**113**

* 1) We merge the time for training the super-network and that for training and evaluating samples into one. 2) We train the discovered network for 350 epochs as mentioned in [22].

Table 1: The comparison between NetAdaptV2 *guided by latency* and related works on ImageNet. The number of MACs is reported for completeness although NetAdaptV2 is not guided by MACs and achieves sub-optimal accuracy-MAC trade-offs. The numbers between parentheses show the breakdown of the search time in terms of training a super-network, training and evaluating samples, and training the discovered DNN from left to right. *Non-Diff. Metrics* denotes whether the method supports non-differentiable metrics. The last column CO_2 *Emission* shows the estimated CO_2 emission based on [34].

Method	Top-1 Accuracy (%)	MAC (M)	Search Time (GPU-Hours)
NSGANetV2-m [38]	78.3	312	1674 (1200, 24, 450)
EfficientNet-B0 [31]	77.3	390	-
MixNet-M [39]	77.0	360	-
NetAdaptV2 (Guided by MAC)	**78.5**	**314**	**656 (204, 35, 417)**

Table 2: The comparison between NetAdaptV2 *guided by MACs* and related works. The numbers between parentheses show the breakdown of the search time in terms of training a super-network, training and evaluating samples, and training the discovered DNN from left to right.

OD	Top-1 Accuracy (%)	Latency (ms)	Search Time (GPU-Hours)
	71.0 (+0)	43.9 (100%)	721 (100%) (0, 543, 178)
✓	71.1 (+0.1)	44.4 (101%)	221 (31%) (50, 2, 169)

Table 3: The ablation study of the proposed ordered dropout (OD) on MobileNetV1 [40] and ImageNet. The numbers between parentheses show the breakdown of the search time in terms of training a super-network, training and evaluating samples, and training the discovered DNN from left to right.

Methods		Top-1
CBC	MCD	Accuracy (%)
		75.9 (+0)
✓		76.2 (+0.3)
✓	✓	76.6 (+0.7)

Table 4: The ablation study of the channel-level bypass connections (CBCs) and the multi-layer coordinate descent optimizer (MCD) on ImageNet. The latency of the discovered networks is around 51ms, and ordered dropout is used.

reduction is small enough, specifically below 3% in this experiment, the accuracy of the discovered networks does not change with the latency reduction.

The second experiment is fixing the time for evaluating samples by scaling both the initial latency reduction and the number of samples per iteration at the same rate. As shown in Table 5, as long as the latency reduction is small enough, more samples will result in better discovered networks. However, if the initial latency reduction is too large, increasing the number of samples per iteration cannot prevent the accuracy from degrading.

• Accuracy Variation across Multiple Executions

To know the accuracy variation of each step in NetAdaptV2 [41], we execute different steps three times and summarize the resultant accuracy of the discovered networks in Table 6. The initial latency reduction is 1.5%, and the number of samples per iteration is 100 ($J = 100$). The la-

	Initial Latency Reduction	Number of Samples (J)	Top-1 Accuracy (%)
Fixed Number of Samples per Iteration	1.5%	100	76.4
	3.0%	100	76.4
	6.0%	100	75.9
Fixed Time for Evaluating Samples	1.5%	100	76.4
	3.0%	200	76.6
	6.0%	400	75.7

Table 5: The experiments for evaluating the influence of the two main hyper-parameters of the MCD optimizer, which are the initial latency reduction (with a fixed decay of 0.98 per iteration) and the number of samples (J). All discovered networks have almost the same latency (51ms).

Training Super-Network	Evaluating Samples	Training Discovered DNN	Top-1 Accuracy of Executions (%)		
			1	2	3
		✓	76.1	76.1	76.2
	✓	✓	76.1	76.2	76.4
✓	✓	✓	76.1	76.2	76.4

Table 6: The accuracy variation of NetAdaptV2. The ✓ denotes the step is executed three times, and the others are executed once. For example, the last row corresponds to executing the entire algorithm flow of NetAdaptV2 three times. For the MCD optimizer, the initial latency reduction is 1.5%, and the number of samples per iteration is 100 ($J = 100$). The latency of all discovered networks is around 51ms, and the accuracy values are sorted in ascending order.

tency of discovered networks is around 51ms. According to the last row of Table 6, which corresponds to executing the entire algorithm flow of NetAdaptV2 three times, the accuracy variation is 0.3%. The variation is fairly small because simply training the same discovered network three times results in an accuracy variation of 0.1% as shown in the first row. Moreover, when we fix the super-network and execute the MCD optimizer three times as shown in the second row, the accuracy variation is the same as that of executing the entire NetAdaptV2 three times. The result suggests that the randomness in training a super-network does not increase the overall accuracy variation, which is preferable since we only need to perform this relatively costly step one time.

4.2. Depth Estimation

4.2.1 Experiment Setup

NYU Depth V2 [28] is used for depth estimation. We reserve 2K training images for evaluating the performance of samples and train the super-network with the rest of the training images. The initial network is FastDepth [42]. Following FastDepth, we pre-train the encoder of the super-network on ImageNet. The batch size is 256, and the learning rate is 0.9 decayed by 0.963 every epoch. After pre-training the encoder, we train the super-network on NYU Depth V2 for 50 epochs with a batch size of 16 and an initial learning rate of 0.025 decayed by 0.9 every epoch. For the MCD optimizer, we generate 150 ($J = 150$) samples per iteration. We search with latency measured on a Google Pixel 1 CPU. The latency reduction is 1.5% in the first iteration and is decayed by 0.98 every iteration. For training the discovered network, we use the same setup as training the super-network, except that the initial learning rate is 0.05.

Method	RMSE (m)	Delta-1 Accuracy (%)	Latency (ms)	Search Time (GPU-Hours)	
				ImageNet	NYU Depth
NetAdaptV1 [1]	0.583	77.4	87.6	96	65
NetAdaptV2	**0.576**	**77.9**	**86.7**	96	**27**

Table 7: The comparison between NetAdaptV2 and NetAdaptV1 on depth estimation and NYU Depth V2 [28].

4.2.2 Search Result

The comparison between the proposed NetAdaptV2 and NetAdaptV1 [1], which is used in FastDepth [42], is summarized in Table 7. NetAdaptV2 reduces the search time by 2.4× on NYU Depth V2, and the discovered DNN outperforms that of NetAdaptV1 by 0.5% in delta-1 accuracy with comparable latency. Because NYU Depth V2 is much smaller than ImageNet, the reduction in the total search time is less than that of applying NetAdaptV2 on ImageNet. The search time spent on ImageNet is for pre-training the encoder, which is a common practice and indispensable when training DNNs for depth estimation on NYU Depth V2.

5. Conclusion

In this paper, we propose NetAdaptV2, an efficient neural architecture search algorithm, which significantly reduces the total search time and discovers DNNs with state-of-the-art accuracy-latency/accuracy-MAC trade-offs. NetAdaptV2 better balances the time spent per step and supports non-differentiable search metrics. This is realized by the proposed methods: channel-level bypass connections, ordered dropout, and multi-layer coordinate descent optimizer. The experiments demonstrate that NetAdaptV2 can reduce the total search time by up to 5.8× on image classification and 2.4× on depth estimation and discover DNNs with better performance than state-of-the-art works.

Acknowledgement

This research was funded by the National Science Foundation, Real-Time Machine Learning (RTML) program, through grant No. 1937501, a Google Research Award, and gifts from Intel and Facebook.

References

[1] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *European Conference on Computer Vision (ECCV)*, 2018.

[2] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[3] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations (ICLR)*, 2019.

[4] B. Chen, G. Ghiasi, H. Liu, T.-Y. Lin, D. Kalenichenko, H. Adam, and Q. V. Le, "Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13604–13613, 2020.

[5] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11390–11399, 2019.

[6] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[7] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[8] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations (ICLR)*, 2019.

[9] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *International Conference on Computer Vision (ICCV)*, October 2019.

[10] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," *ArXiv*, vol. abs/1903.11728, 2019.

[11] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations (ICLR)*, 2020.

[12] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le, "Bignas: Scaling up neural architecture search with big single-stage models," in *European Conference on Computer Vision (ECCV)*, 2020.

[13] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning (ICML)*, 2018.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *International Conference on Machine Learning (ICML)*, 2018.

[15] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P.-J. Kindermans, and Q. V. Le, "Can weight sharing outperform random architecture search? an investigation with tunas," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[16] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *European Conference on Computer Vision (ECCV)*, 2020.

[17] A. Gordon, E. Eban, O. Nachum, B. Chen, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

[19] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[20] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, P. Vajda, and J. E. Gonzalez, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12962–12971, 2020.

[21] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," in *arXiv preprint arXiv:1904.02877*, 2019.

[22] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization," in *arXiv preprint arXiv:1907.00959*, 2019.

[23] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang, "Atomnas: Fine-grained end-to-end neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2020.

[24] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient architecture search," in *International Conference on Learning Representations (ICLR)*, 2020.

[25] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *International Conference on Computer Vision (ICCV)*, 2019.

[26] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, *Efficient Processing of Deep Neural Networks*. Morgan & Claypool, 2020.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[28] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *European Conference on Computer Vision (ECCV)*, 2012.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[31] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning (ICML)*, 2019.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[33] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv*, vol. abs/1706.02677, 2017.

[34] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

[35] X. Chu, B. Zhang, and R. Xu, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," *arXiv preprint arXiv:1907.01845*, 2019.

[36] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision (ECCV)*, 2016.

[38] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *European Conference on Computer Vision (ECCV)*, 2020.

[39] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," in *British Machine Vision Conference (BMVC)*, 2019.

[40] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[41] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proceedings of Machine Learning Research*, vol. 115, pp. 367–377, 22–25 Jul 2020.

[42] Wofk, Diana and Ma, Fangchang and Yang, Tien-Ju and Karaman, Sertac and Sze, Vivienne, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," in *International Conference on Robotics and Automation (ICRA)*, 2019.