EffConv: Efficient Learning of Kernel Sizes for Convolution Layers of CNNs

Alireza Ganjdanesh, Shangqian Gao, Heng Huang

Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA
{alireza.ganjdanesh, shg84, heng.huang}@pitt.edu

Abstract

Determining kernel sizes of a CNN model is a crucial and non-trivial design choice and significantly impacts its performance. The majority of kernel size design methods rely on complex heuristic tricks or leverage neural architecture search that requires extreme computational resources. Thus, learning kernel sizes, using methods such as modeling kernels as a combination of basis functions, jointly with the model weights has been proposed as a workaround. However, previous methods cannot achieve satisfactory results or are inefficient for large-scale datasets. To fill this gap, we design a novel efficient kernel size learning method in which a size predictor model learns to predict optimal kernel sizes for a classifier given a desired number of parameters. It does so in collaboration with a kernel predictor model that predicts the weights of the kernels - given kernel sizes predicted by the size predictor - to minimize the training objective, and both models are trained end-to-end. Our method only needs a small fraction of the training epochs of the original CNN to train these two models and find proper kernel sizes for it. Thus, it offers an efficient and effective solution for the kernel size learning problem. Our extensive experiments on MNIST, CIFAR-10, STL-10, and ImageNet-32 demonstrate that our method can achieve the best training time *vs.* accuracy trade-off compared to previous kernel size learning methods and significantly outperform them on challenging datasets such as STL-10 and ImageNet-32. Our implementations are available at https://github.com/Alii-Ganjj/EffConv.

Introduction

Convolutional neural network (CNNs) have consistently shown top performance in image classification tasks in the last decade (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2015; Szegedy et al. 2016; He et al. 2016; Liu et al. 2022b). They consist of multiple layers of convolution operators with learnable weights that, combined with pooling layers and strided convolutions, downsample an input image gradually. Such a design blended with normalization and nonlinear layers enables them to make an information bottleneck to extract low-resolution task-relevant information from high-resolution low-level details such as pixels' intensities. Despite their success, developing new high-accuracy architectures is a complex task involving numerous hyperparameters. One of the critical design choices that directly affects the model's performance is the size of the kernels for convolutional layers.

A prominent design for CNNs is to use numerous layers of blocks with predetermined kernel sizes. (Simonyan and Zisserman 2015; Szegedy et al. 2016; He et al. 2016; Ding et al. 2022). However, by doing so, the network's predictions are not differentiable *w.r.t* kernel sizes. Thus, finding the optimal setting for them (given a desired number of parameters) becomes a hyperparameter tuning problem that can be solved using cross-validation, casting it as a constrained discrete optimization problem, or leveraging heuristic design tricks. Cross-validation gets infeasible as the number of combinations grows exponentially *w.r.t* number of layers and possible kernel sizes. The constrained optimization problem should be solved with techniques such as Neural Architecture Search (NAS) (Zoph and Le 2017) that require powerful computational resources that may not be available for low-resourced applications. Although using heuristic tricks (Ding et al. 2022; Dollár, Singh, and Girshick 2021; Liu et al. 2022b) to design kernel sizes has shown remarkable results, their intuitions are mainly based on an architecture's performance on ImageNet (Russakovsky et al. 2015), which is shown does not necessarily translate to a more effective model on other domains, with even negative correlations between the metrics (Tuggener, Schmidhuber, and Stadelmann 2021; Girish et al. 2022) in some cases.

Learning a model's kernel sizes jointly with its weights during training is a promising direction to overcome the mentioned challenges. A group of methods take a set of basis functions (*e.g.,* Delta-Diracs (Dai et al. 2017) or Gaussian functions (Pintea et al. 2021)) and learn dilation factors for these functions to alter a kernel size. However, dilation limits the learned kernels' bandwidth and hurts the model's performance. Recently, FlexConv (Romero et al. 2022a) proposed to learn high-bandwidth kernels by defining them as a product of a predicted kernel by a neural network (MLP) and a Gaussian kernel with learnable parameters. For each convolution layer, its corresponding MLP predicts a kernel with the same spatial dimensions as the input image. Then, the predicted kernel is multiplied by the Gaussian kernel. Finally, the resulting kernel is cropped according to a threshold. Nevertheless, doing so has several key drawbacks. First,

if we denote the input size as N, FlexConv needs to compute N^2 MLP forward passes for each convolution layer in each iteration. Thus, its training time grows quadratically *w.r.t* input size, making it highly inefficient even for datasets with moderate input size such as STL-10 (Coates, Ng, and Lee 2011). It also makes using FlexConv prohibitively expensive for very deep architectures. Secondly, cropping by thresholding wastes the computations of the MLP and is an inefficient way to determine kernel sizes.

We propose a new kernel size learning method that has significantly lower computational requirements than previous methods and, at the same time, outperforms them on different tasks. We develop a model called *size predictor* that learns to determine optimal kernel sizes for a CNN. During training, the size predictor collaborates with a hypernetwork (Ha, Dai, and Le 2017) named *kernel predictor*. In each iteration, at first, the size predictor decides the size of kernels for the CNN. Then, the kernel predictor adaptively predicts kernel weights given the specified sizes by the size predictor. This scheme prevents wasting computations by the kernel predictor that is happening in FlexConv after cropping. Moreover, our method needs orders of magnitude lower forward passes during training, making our method dramatically faster than FlexConv. Finally, the weights of both models are optimized to minimize the classification objective while keeping the parameter count of the CNN at the desired number. By doing so, our method can find a high-performance configuration for kernel sizes of a CNN with a small fraction of training epochs needed to train it from scratch. After finding a proper configuration, we discard the size predictor as well as kernel predictor and train the resulting CNN with the same training settings as the original CNN. Therefore, our method is able to discover and train a high accuracy setting for the kernel sizes with much lower computations than other methods. Our experimental results on MNIST, CIFAR-10, STL-10, and ImageNet-32 demonstrate that our method consistently shows better accuracy *vs.* training time trade-off compared to other kernel size learning methods, even outperforming them in both metrics in most cases. We summarize our contributions as follows:

- We introduce a novel kernel size learning method for CNNs that requires a significantly lower computational budget than existing methods.
- We design a size predictor model that is trained jointly with a kernel predictor model to predict optimal kernel sizes of a classifier. Training these models requires only a small fraction of the training epochs of the CNN.
- Our experiments on several benchmark datasets illustrate that our method can achieve more competent architectures in much lower training epochs than other kernel size learning methods, providing an efficient and effective solution for the kernel size learning problem.

Related Works

Innovation of novel performant CNN architectures for different applications is a complicated task consisting of numerous factors. One of the crucial factors is determining kernel sizes for convolution layers. Most of the proposed ideas design architectures consisting of multiple layers with fixed preset sizes. Early works (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2015; Szegedy et al. 2016; He et al. 2016) use blocks with kernels of sizes 1-7px. The main downside of using fixed architectures is that the training objective will not be differentiable *w.r.t* kernel sizes. Thus, an optimal setting should be found by cross validation, solving a constrained discrete optimization, or developing heuristic design tricks. However, the number of potential architectures increases exponentially *w.r.t* the number of layers and kernel sizes range, making cross validation impractical. The high number of possible architectures also translates into a vast search space for NAS methods (Zoph and Le 2017; Tan and Le 2019; Howard et al. 2019; Han et al. 2020) used to solve the constrained discrete optimization problems for finding optimal kernel sizes. Thus, NAS methods require an extreme computation budget to find the desired configuration. Using heuristics to design kernel sizes has shown significant results recently (Dollár, Singh, and Girshick 2021; Liu et al. 2022b; Ding et al. 2022; Liu et al. 2022a). RepLKNet. (Ding et al. 2022) introduces five guidelines to design CNNs with large kernel sizes up to 31px. It uses small kernels parallel to large ones and achieves comparable results to Swin Transformer (Liu et al. 2021). ConvNeXt (Liu et al. 2022b) follows the design of Swin Transformer to develop large kernel CNNs. However, these models' intuitions and design tricks mainly rely on improving the Top-1 accuracy of the resulting architectures on ImageNet (Russakovsky et al. 2015). Thus, their superior accuracy on ImageNet may not reflect on other domains, as shown by recent studies (Tuggener, Schmidhuber, and Stadelmann 2021; Girish et al. 2022).

Learning kernel sizes during training can alleviate the weaknesses for predesigned architectures. Deformable ConvNets (Dai et al. 2017) learn offsets for each pixel of a filter to obtain deformable kernels. Also, (Pintea et al. 2021; Tomen, Pintea, and Van Gemert 2021) model kernels as a learned combination of Gaussian derivative filters with a tunable variance parameter to control its effective size. OFS-CNN (Han et al. 2018) use learnable padding operations, and (Shelhamer, Wang, and Darrell 2019; Xiong et al. 2020; Tabernik, Kristan, and Leonardis 2020) model filters as learnable dilated Gaussian functions to learn kernel sizes. Yet, dilated basis functions/kernels result in kernels with limited bandwidths, restricting the final model's performance. Recently, FlexConv (Romero et al. 2022a), inspired by continuous kernel convolutions (Wang et al. 2018; Shi et al. 2020; Thomas et al. 2018; Schütt et al. 2017; Simonovsky and Komodakis 2017; Romero et al. 2022b), proposed to model kernels of a CNN as a product of a kernel predicted by a continuous function (implemented by a MLP) and a learned Gaussian kernel. They train a separate MLP and Gaussian kernel for each kernel. The MLP predicts kernels with the same resolution of an input. Then, the Gaussian kernel is multiplied with it, and the resulting kernel is cropped using a tunable threshold. Nevertheless, such scheme requires multiple forward passes and the cropping method wastes computations. Our method can effectively overcome these limitations such that our kernel predictor

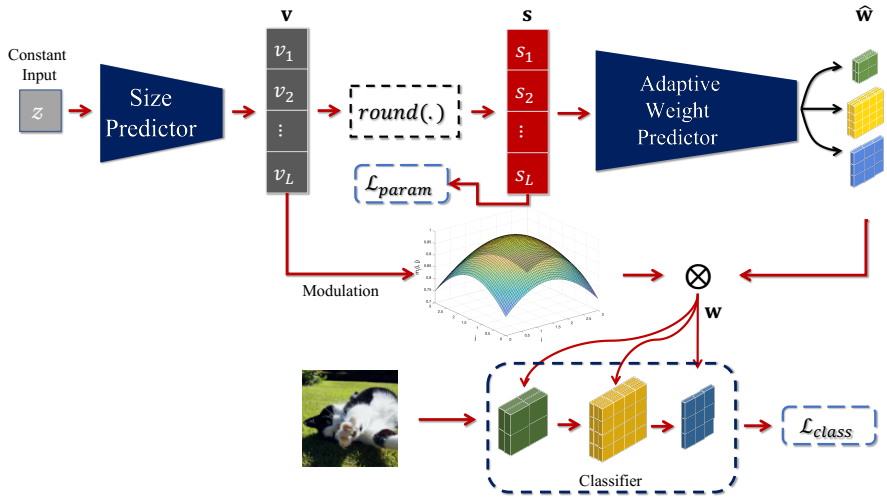


Figure 1: Overview of our method. Our size predictor model learns to predict the kernel sizes for the classifier. It predicts *soft* kernel sizes $\mathbf{v}$ that are rounded to integer values. Then, our adaptive weight predictor model predicts optimal kernel weights $\hat{\mathbf{w}}$ given the predicted sizes. We modulate the predicted weights using masks m_l parameterized by the soft sizes $\mathbf{v}$ to make the resulting weights $\mathbf{w}$ differentiable $w.r.t$ the size predictor's weights. Finally, the weights $\mathbf{w}$ are used as the kernel weights of the classifier, and the training is guided by the classification objective ($\mathcal{L}_{class}$) and the parameters budget loss ($\mathcal{L}_{param}$).

adaptively predicts kernels with sizes predicted by the size predictor. Thus, it only needs forward passes proportional to predicted sizes and not the size of input images, thereby making our method much more efficient compared to FlexConv.

Methodology

We elaborate on different components of our method in the following sub-sections. We show an overview of our approach in Fig. 1.

Size Predictor

We develop a size predictor model that dynamically predicts kernel sizes during training. We implement our size predictor with a GRU (Cho et al. 2014) unit followed by feed forward layers. In each step, it takes a fixed input z and predicts the kernel sizes as follows:

$$\mathbf{v} = \sigma(f_{sp}(z; \theta_{sp})) \tag{1}$$

$$\mathbf{s} = round(\mathbf{v}) \tag{2}$$

f_{sp} represents the size predictor, and $round(\cdot)$ rounds the input to its nearest integer. σ is the nonlinearity that we use to ensure the predicted kernel sizes are in the designed range. We implement it with $\sigma(x) = k * tanh(x/\tau + b) + k + 1$. With such a design, predicted sizes in $\mathbf{s}$ will be in $[1, 2k+1]$. We set τ to enlarge the active region of $tanh$ and prevent the vanishing gradients problem. We choose b such that the initially predicted sizes be close to the original design of the architectures. Given a predicted size vector $\mathbf{s}$, the number of parameters for the resulting model will be:

$$p_{pr} = \sum_{l=1}^{L} c_l \times c_{l-1} \times s_l \times s_l \tag{3}$$

where c_l is the number of output channels for the l-th convolution layer. To regulate the size predictor to fix the parameter count of the classifier to the desired number p_d, we use the following objective:

$$\mathcal{L}_{param} = \begin{cases} log(\frac{p_{pr}}{p_d}) & p_{pr} > p_d \\ log(\frac{p_d}{p_{pr}}) & \text{Otherwise} \end{cases} \tag{4}$$

The regularization in Eq. 4 has also been utilized for network pruning (Ganjdanesh, Gao, and Huang 2022; Gao et al. 2020). As the $round(\cdot)$ function is not differentiable, we calculate the gradients of $\mathcal{L}_{param}$ $w.r.t$ θ_{sp} using the Straight Through Estimator (Bengio, Léonard, and Courville 2013).

Kernel Predictor

We use a kernel predictor that estimates proper weights for the CNN given the kernel sizes determined by the size predictor. Our motivation for such a design is that the kernel sizes change dynamically during training, and using a fixed set of trainable parameters for kernel weights is challenging. Thus, we utilize a model that can effectively adjust its output size given input sizes. We implement our kernel predictor with a GRU (Cho et al. 2014) block for each convolution layer in the model. We represent the number of input/output channels of the l-th convolution layer with c_{l-1}/c_l. Accordingly, we set the output size of its corresponding GRU being $c_{l-1} \times c_l$. Now, if the predicted size for this layer is s_l, its GRU can adaptively predict the weights for it by performing $s_l \times s_l$ time step forward passes, predicting the weights for each spatial dimension of the kernel one at a time:

$$\hat{w}_l = f^l_{kp}(z'_l; s_l; \theta^l_{kp}) \tag{5}$$

z'_l is the constant input of the GRU. Such a design remarkably reduces the computation needed to predict a kernel's weights compared to the continuous convolution kernel of FlexConv (Romero et al. 2022a). The reason is that our kernel predictor needs to compute s_l^2 time steps (forward passes) to predict a kernel's weights, but FlexConv requires N^2 forward passes (N is the size of the input to the CNN), regardless of the final size of the kernel, making it extremely inefficient even for datasets with an average input size such as STL-10 (Coates, Ng, and Lee 2011). In addition, the training efficiency can potentially be improved by using model parallelism algorithms (Xu, Huo, and Huang 2020).

Modulating Kernel Size in the Predicted Weights: Despite the efficiency of our kernel predictor in determining a kernel's weights, its predictions are neither differentiable *w.r.t* s_l nor v_l as s_l only determines the number of time step calculations. Thus, naive usage of our kernel predictor does not provide any feedback for the size predictor model during training when using gradient-based optimization schemes.

We propose modulating the predicted kernel size into the predicted weights as a workaround. Inspired by adaptive attention span (Sukhbaatar et al. 2019), we multiply a mask m_l parameterized by the soft kernel size v_l to the predicted weights $\hat{w}_l$. We design m_l as:

$$m_l(i,j) = exp(-\frac{(i-c_l)^2 + (j-c_l)^2}{v_l^2}) \\ c_l = v_l/2, \;\; i,j \in \{1,2,\cdots s_l\} \tag{6}$$

Finally, we apply the mask m_l to the predicted weights $\hat{w}_l$ and use the resulting kernel w_l in the classifier for training:

$$w_l = m_l \odot \hat{w}_l \quad l \in \{1,\cdots,L\} \tag{7}$$

where $\odot$ means element-wise product. We use our exponential mask instead of the piece-wise linear one proposed by (Sukhbaatar et al. 2019) as it makes all of the weights of the resulting kernel differentiable *w.r.t* v_l, making the gradient flow (Hochreiter et al. 2001) for v_l easier. In contrast, in the latter, only the weights at the edges have such property, limiting the flow. Our mask attenuates the edges of kernels and almost keeps the middle parts intact, encouraging the model not to predict unnecessarily large kernels.

We train the size predictor, kernel predictor, and the parameters of the CNN to minimize the classification objective (Cross-Entropy loss) while keeping the number of parameters of the CNN at the desired budget. Thus, our final objective is:

$$\min_{\theta_{sp},\theta_{kp},\theta_c} \mathcal{L} = \mathcal{L}_{class}(\theta_{sp},\theta_{kp},\theta_c) + \lambda \mathcal{L}_{param}(\theta_{sp}) \tag{8}$$

θ_c represents the parameters of the CNN, and λ is a hyperparameter. We show in our experiments that our method only needs a small fraction of the training epochs of a CNN to train a size predictor for it. After that, we discard the size predictor as well as kernel predictor models and use the learned kernel sizes for the CNN to train it with its default training settings. We summarize our training algorithm in Alg. 1.

Algorithm 1: Our Kernel Size Learning Algorithm

Input: Training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{D}$; CNN $f_c(.;\theta_c)$; size predictor $f_{sp}(\cdot;\theta_{sp})$; size predictor nonlinearity parameters $\{k, b, \tau\}$; kernel predictor $f_{kp}(\cdot;\theta_{kp})$; size learning epochs E_s; training epochs E_T.
Output: CNN with learned kernel sizes and weights.
`/* Learning Kernel Sizes of the CNN */`
for $e := 1$ *to* E_s **do**
1. Sample a mini-batch $(\mathbf{x}, \mathbf{y})$ from $\mathcal{D}$.
2. Calculate kernel sizes $\mathbf{s}$ and soft predictions $\mathbf{v}$ of the size predictor (Eqs. 1, 2) using $f_{sp}(\cdot;\theta_{sp})$ and $\{k, b, \tau\}$.
3. Calculate $\hat{w}_l$ using the predicted sizes $\mathbf{s}$ and the kernel predictor $f_{kp}(\cdot;\theta_{kp})$ according to Eq. 5
4. Compute the modulating masks m_l using soft predictions $\mathbf{v}$ of the size predictor in Eq. 6.
5. Apply the masks m_l to $\hat{w}_l$ in Eq. 7 and use the resulting w_l as the weights of the CNN.
6. Use the sampled $(\mathbf{x}, \mathbf{y})$ to calculate the classification loss $\mathcal{L}_{class}$ of the CNN.
7. Calculate the parameters budget loss $\mathcal{L}_{param}$ using $\mathbf{s}$ and Eq. 4
8. Compute $\mathcal{L}$ in Eq. 8, backpropagate the gradients *w.r.t* $\theta_{sp}, \theta_{kp}, \theta_c$, and update them.
end
`/* Train the CNN with new kernel sizes */`
9. Determine the optimal kernel sizes of the CNN using the trained size predictor.
10. Initialize the CNN using the predicted kernel sizes.
11. Train the CNN for E_T epochs with its standard settings.
Return: Trained CNN with new kernel sizes.

Experiments

In this section, at first, we provide our experimental setup. Then, we evaluate our proposed method against baselines on four different tasks. Finally, we explore the behavior of our model through two ablation studies.

Experimental Setup

Datasets: We use four datasets in our experiments to analyze our model: MNIST (LeCun et al. 1998), CIFAR-10 (Krizhevsky, Hinton et al. 2009), STL-10 (Coates, Ng, and Lee 2011), and ImageNet-32 (Chrabaszcz, Loshchilov, and Hutter 2017).

Architectures: We mainly focus on finding kernel sizes for ResNet (He et al. 2016) architectures with depths in $\{20, 26, 32, \cdots, 56\}$ in our experiments. We explore our model's performance for Wide-ResNet (Zagoruyko and Komodakis 2016) for CIFAR-10 experiments as well. For all models, we learn the kernel sizes for the residual layers and keep the first input 3×3 convolution kernel.

Training Details: For all architectures and datasets except ImageNet, we train the size predictor and kernel predictor models for 10 epochs. We do so for ImageNet-32 experiments for 5 epochs. Remarkably, we found that training our size predictor and kernel predictor models does not need hyperparameter tuning across architectures and datasets. We use the AdamW optimizer (Loshchilov and Hutter 2019)

for all models for the kernel size learning stage. We use a learning rate of 0.0001 with a weight decay of 0.001 for the size predictor and the CNN's parameters (the ones other than convolution kernels such as Batch Normalization (Ioffe and Szegedy 2015) and Fully connected layers). For the kernel predictor, we utilize a learning rate of 0.001 and the same weight decay. Then, we train the resulting CNN with the training settings of the original model following the settings described in FlexConv (Romero et al. 2022a). For all experiments, we set the size predictor nonlinearity parameters to be $(k, \tau, b) = (3, 3, -0.35)$ so that our possible size range will be $[1, 7]$. In addition, the initial values of predicted soft kernel sizes will be close to 3, which is the original design for ResNet (He et al. 2016) and Wide-ResNet (Zagoruyko and Komodakis 2016) architectures. We also set the parameter $\lambda = 2$ for the parameters budget loss in Eq. 8.

Evaluation Metrics: We use the accuracy of the models/-training time per epoch to compare the performance/efficiency of the models, following FlexConv (Romero et al. 2022a). We calculate the time per epoch of our model as the average of times for training epochs of the stage for learning kernel sizes as well as training the resulting CNN from scratch. We call our method EffConv-X/Wide-EffConv-X-Y, representing the ResNet-X/Wide-ResNet-X-Y model with the depth X, widen factor Y, and kernel sizes learned by our method. We use a Lambda machine with 8 NVIDIA RTX A5000 GPUs with 256 CPU cores for our experiments. Due to the space limit, we provide other details of our experiments in supplementary.

Model	Size	Accuracy	Time (Sec / Epoch)
Efficient-CapsNet (Mazzia, Salvetti, and Chiaberge 2021)	0.16M	99.74 ± 0.0002	-
Network in Network (Lin, Chen, and Yan 2013)	N/A	99.53	-
VGG-5 (Kabir et al. 2022)	3.65M	99.72	-
FlexNet-16 (Romero et al. 2022a)	0.67M	99.7 ± 0.0	~ 205[1]
EffConv-20 (Ours)	0.66M	99.80 ± 0.01	35.86

Table 1: Results on the MNIST dataset.

Results

We report a mean and standard deviation of 3 runs of our model with different random seeds for all experiments.

MNIST

We summarize the results on the MNIST dataset in Tab. 1. As can be seen, our method can achieve the highest accuracy. Although MNIST is a saturated benchmark, we can observe that our method can find a performant model in much lower training time ($\sim 6\times$ less) compared to FlexConv.

CIFAR-10

Tab. 2 represents the results of experiments on the CIFAR10 dataset. We can observe that our method can perform better than N-JetNet (Pintea et al. 2021) and DCN (Tomen, Pintea, and Van Gemert 2021) models with different number of parameters. Both of these methods model a convolution kernel as a truncated Taylor series (called N-Jet (Jacobsen et al. 2016)) of Gaussian derivative filters to learn kernel sizes. N-JetNet-ALLCNN shows significantly worse accuracy compared to our model's variants with 0.66M parameters. N-JetNet-CIFARResNet32 shows higher accuracy, which is on par with our EffConv-20 model with a similar number of parameters, but our model has much lower depth than the former.

Model	Size	Accuracy	Time (Sec / Epoch)
N-JetNet-ALLCNN (Pintea et al. 2021)	0.66M	89.91 ± 0.03	-
N-JetNet-CIFARResNet32 (Pintea et al. 2021)	0.52M	92.28 ± 0.26	-
DCN-σ^{ji} (Tomen, Pintea, and Van Gemert 2021)	0.47M	89.7 ± 0.3	-
CKCNN-7 (Romero et al. 2022b)	0.70M	71.7	-
CKCNN-16 (Romero et al. 2022b)	0.63M	72.1 ± 0.2	68
$\text{CKCNN}_{\text{MAGNet}}$-16 (Romero et al. 2022b)	0.67M	86.8 ± 0.6	102
FlexNet-3 (Romero et al. 2022a)	0.27M	90.4 ± 0.2	-
FlexNet-5 (Romero et al. 2022a)	0.44M	91.0 ± 0.5	-
$\text{FlexNet}_{\text{Gabor}}$-16 (Romero et al. 2022a)	0.67M	91.9 ± 0.2	161
FlexNet-16 (Romero et al. 2022a)	0.67M	92.2 ± 0.1	127
CIFARResNet-44 (He et al. 2016)	0.66M	92.83	22
CIFARResNet-44 w/ CKConv (k=3)	2.58M	86.1 ± 0.9	-
CIFARResNet-44 w/ FlexConv	2.58M	81.6 ± 0.8	-
EffConv-20 (Ours)	0.20M	91.78 ± 0.16	
EffConv-20 (Ours)	0.50M	92.30 ± 0.11	
EffConv-20 (Ours)	0.66M	92.35 ± 0.05	17.77
EffConv-32 (Ours)	0.66M	92.99 ± 0.02	23.82
EffConv-44 (Ours)	0.66M	93.35 ± 0.17	27.65
Wide-EffConv-28-1	0.66M	93.14 ± 0.23	15.99

Table 2: Results on the CIFAR10 dataset.

It can be seen that our method significantly outperforms CKCNN (Romero et al. 2022b) models in terms of both accuracy and training time with a similar parameter budget. Our conjecture for the cause of weak results of CKCNN is that it is mainly designed for modeling sequential data and not images.

FlexConv (Romero et al. 2022a) can obtain models with higher accuracy models compared to DCN and CKCNN, but it does so at the cost of higher training time per epoch. Our method shows higher accuracy with a similar number of parameters while having a much lower training time. In the low parameter regime, FlexConv-3 with 0.27M parameters can achieve an average of 90.4% accuracy. At the same time, our EffConv-20 model with 0.20M parameters shows an average accuracy of 91.78%. In the higher parameter setting, FlexNet-16 with 0.67M parameters shows lower average accuracy compared to EffConv-16 with 0.66M parameters while requiring more than $7\times$ (127 *vs.* 17.77) training time per epoch. This result highlights the critical advantage of our model, *i.e.,* finding performant deeper models with much lower computations than FlexConv. The cost of

[1]The number is provided by the authors by correspondence.

training FlexConv models drastically increases with the increase in the model's depth, which prevents it from training models deeper than 16 layers. Conversely, we show in our ablation studies that our model can readily find proper kernel sizes for much deeper models like ResNet-56. One can also find that directly plugging CKConv and FlexConv layers into the CIFARResNet-44 model's blocks severely degrades its performance while increasing its number of parameters by about $4\times$. In contrast, while keeping the number of parameters the same as CIFARResNet-44 (0.66M), our EffConv-44 can find a better configuration for its kernel sizes with a slight increase in its training time without requiring special block designs like CKConv and FlexConv. Finally, we can observe that our Wide-EffConv-28-1 can achieve an average accuracy of 93.14, which is between EffConv-32 and EffConv-44 while requiring about half the training time of EffConv-44. This result is congruent with previous works (Zagoruyko and Komodakis 2016) that observed that Wide-ResNet models perform similarly to deeper ResNets while having lower training/inference time thanks to their smaller depth.

Finally, we visualize the learned sizes and weights of the kernels for our EffConv-20 model with 0.66M parameters in Fig. 2. As can be seen, in contrast with FlexConv (Romero et al. 2022a) that puts larger kernels in the deeper layers, our model does not show such a behavior, and the largest kernel is set in the 5$^{\text{th}}$ layer. We provide the optimal kernel sizes for other architectures in supplementary.

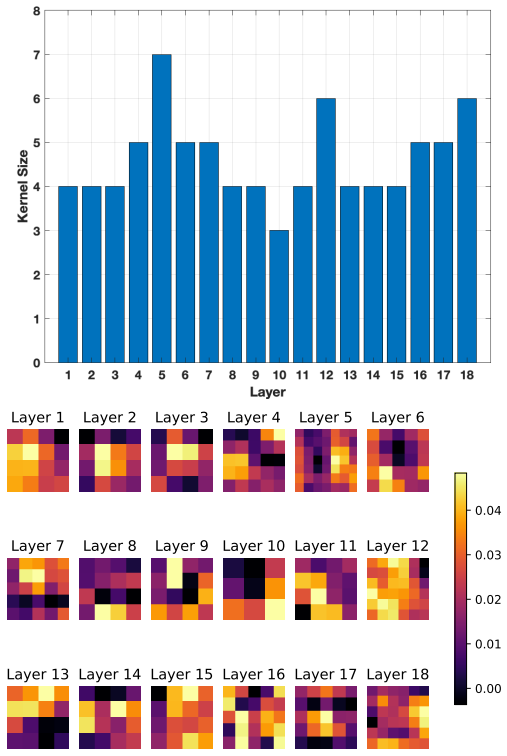


Figure 2: Learned sizes (top) and weights (bottom) for the kernels of our EffConv-20 model with 0.66M parameters on CIFAR-10.

Table 3: Results on the STL-10 dataset.

Model	Size	Accuracy	Time (Sec / Epoch)
CIFARResNet-18 (He et al. 2016)	11.7M	81.0	14.2
FlexNet-16 (Romero et al. 2022a)	0.67M	68.6 ± 0.7	~ 1800[2]
EffConv-20 (Ours)	0.66M	73.06 ± 0.53	13.2
EffConv-20 (Ours)	0.71M	73.60 ± 0.58	14.41
EffConv-20 (Ours)	0.78M	74.02 ± 0.27	14.45

STL-10

We present the results on the STL-10 dataset in Tab. 3. STL-10 (Coates, Ng, and Lee 2011) is a challenging benchmark as it has a small training dataset (5k samples) and a more extensive test set (8k samples) that can better show the differences between methods compared to CIFAR-10 and MNIST. As can be seen, FlexConv (Romero et al. 2022a) becomes a highly inefficient method on STL-10 even though it has a small training dataset. The reason is that the input images have a size of 96px, *i.e.*, $9\times$ larger than 32px images in CIFAR-10. Thus, FlexConv needs to compute 96^2 forward passes of the MLP for each convolution kernel to compute its weights in each iteration, making it require about 30 minutes to complete each epoch. In contrast, thanks to our adaptive weight predictor model design, our method can successfully find the proper kernel sizes in a significantly lower time showing training time on par with the CIFARResNet-18. In addition, our method can noticeably shrink the gap between low parameter models and the high parameter CIFARResNet-18 model. With a similar 0.66M parameter budget, EffConv-20 can outperform FlexConv with 4.46% accuracy, and at the same time, finding and training such a model in $136\times$ less time. Thus, EffConv is more practical compared to FlexConv for applications with large input sizes.

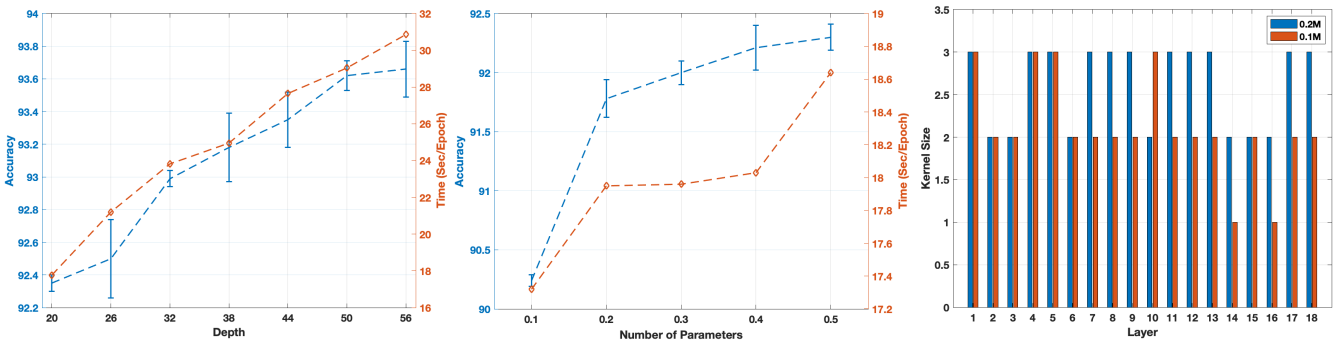
Table 4: Results on the ImageNet-32 dataset.

Model	Size	Accuracy		Time (Sec / Epoch)
		Top-1	Top-5	
CIFARResNet-32 (He et al. 2016)	0.53M	26.41 ± 0.13	49.37 ± 0.15	-
WRN-28-1 (Zagoruyko and Komodakis 2016)	0.44M	32.03	57.51	-
FlexNet-5 (Romero et al. 2022a)	0.44M	24.9 ± 0.4	47.7 ± 0.6	~ 740[2]
EffConv-20 (Ours)	0.50M	36.07 ± 0.18	60.96 ± 0.1	215.27

ImageNet-32

We provide the results on the ImageNet-32 dataset in Tab. 4. We can find that WRN-28-1 significantly outperforms ResNet-32, similar to the results mentioned in CIFAR-10 experiments above and previously shown (Zagoruyko and Komodakis 2016) in the literature. Moreover, FlexConv cannot achieve a competitive performance compared to baselines. Due to its computational inefficiency bottleneck, training deep models on the ImageNet-32 with 1M samples is very expensive for FlexConv. Thus, it has to resort to training a model with only 5 layers, resulting in a limited capacity and unsatisfactory performance. However, EffConv can train a

[2]The number is provided by the authors by correspondence.



(a) (b) (c)

Figure 3: Results of our ablation studies.

$4\times$ deeper model (20 layers) in $3.4\times$ less time. With slightly less parameters count, EffConv-20 significantly outperforms CIFARResNet-32 with 9.66%/11.59% average top-1/top-5 accuracy. It also shows about 4%/3.4% higher average top-1/top-5 accuracy compared to WRN-28-1.

In summary, our experiments on four benchmarks clearly demonstrate that our method EffConv can find proper kernel sizes for a classifier more effectively and efficiently than the previous kernel size learning methods. It can provide a practical solution to the problem of determining kernel sizes for a CNN classifier.

Ablation Studies

We perform two ablation experiments to explore the behavior of our model.

Fixed Parameter Budget, Varying Depth: In the first experiment, we use our model to find proper kernel sizes for ResNet (He et al. 2016) models with depths in $\{20, 26, \cdots, 56\}$ with a fixed parameter budget of 0.66M. Fig. 3(a) demonstrates the results. The left vertical axis represents the accuracy of the resulting models, and the right one shows the training time per epoch. We can observe that, as expected, the accuracy and training time of the learned model increases when we increment its depth. However, the performance almost gets saturated from ResNet-50 to ResNet-56 when these models should decrease their parameters while preserving their depth. Thus, they become thin and deep models, so increasing their depth does not provide noticeable performance returns. We also emphasize that our method can readily find proper kernel sizes for deep models such as ResNet-56 while requiring $\sim$ 31 seconds per epoch. At the same time, FlexConv (Romero et al. 2022a)/CKCNN (Romero et al. 2022b) require about $4\times$/$3\times$ more training time (see Tab. 2) for their 16-layer models ($3.5\times$ fewer layers than ResNet-56) and achieve much worse performance.

Fixed Depth, Varying Parameter Budget: In our second experiment, we use a ResNet-20 model (fixed depth) and change its desired parameter counts when we find its kernel sizes with our model. We set the number of parameters from 0.1M to 0.5M, roughly half and twice the number of parameters of the original ResNet-20. We present the results in Fig. 3(b). We can find that the accuracy increases with the number of parameters, and the slope of changes is low after 0.4M parameters. Moreover, with only 0.1M parameters, our model shows higher performance (90.24 ± 0.05) than N-JetNet-ALLCNN/DCN-σ^{ji} (see Tab. 2), while these models have much higher parameters. One can also notice a nonlinear change in training time. It increases from 0.1M to 0.2M but then almost remains the same from 0.2M to 0.4M despite doubling the number of parameters. Then, again it increases from 0.4M to 0.5M. In contrast, the training time increases almost linearly with the number of layers (Fig. 3(a)) but with a slope of less than one. Finally, Fig. 3(c) visualizes the kernel sizes for the EffConv-20 with 0.1M/0.2M parameters. Interestingly, we can find that our model determines the same size configuration for the first layers of two models that are known to extract low-level general features such as edges, and the models differ in deeper layers.

Conclusion

In this paper, we introduce a novel kernel size learning method for a classifier that overcomes the computational inefficiencies of the previous methods while outperforming their performance. We develop a size predictor model that learns to predict the optimal kernel sizes for the classifier given a desired budget for its parameters' count. We train the size predictor using a second model called kernel predictor that adaptively determines the kernel weights of the classifier given the predicted sizes of the size predictor. Both models are guided by the classification and our parameter budget objectives. Our new model can discover the proper configuration for the kernel sizes and train the resulting model with much lower computations compared to the previous methods. Experiments on four prominent benchmarks demonstrate that our method outperforms the baselines in terms of the classifier's accuracy and significantly reduces their training time per epoch given a fixed parameter budget, showing larger margins on more challenging tasks. Ablation studies also reveal that our method is able to find kernel sizes for much deeper architectures that were not feasible for baselines due to their inefficiencies.

Acknowledgements

This work was partially supported by NSF IIS 1852606, 1838627, 1837956, 1956002, 2211492, CNS 2213701, CCF 2217003, DBI 2225775.

References

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Cho, K.; van Merrienboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In Wu, D.; Carpuat, M.; Carreras, X.; and Vecchi, E. M., eds., *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, 103–111. Association for Computational Linguistics.

Chrabaszcz, P.; Loshchilov, I.; and Hutter, F. 2017. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Coates, A.; Ng, A.; and Lee, H. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 215–223. JMLR Workshop and Conference Proceedings.

Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; and Wei, Y. 2017. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, 764–773.

Ding, X.; Zhang, X.; Han, J.; and Ding, G. 2022. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11963–11975.

Dollár, P.; Singh, M.; and Girshick, R. 2021. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 924–932.

Ganjdanesh, A.; Gao, S.; and Huang, H. 2022. Interpretations Steered Network Pruning via Amortized Inferred Saliency Maps. In Avidan, S.; Brostow, G.; Cissé, M.; Farinella, G. M.; and Hassner, T., eds., *Computer Vision – ECCV 2022*, 278–296. Cham: Springer Nature Switzerland.

Gao, S.; Huang, F.; Pei, J.; and Huang, H. 2020. Discrete Model Compression With Resource Constraint for Deep Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girish, S.; Dey, D.; Joshi, N.; Vineet, V.; Shah, S.; Mendes, C. C. T.; Shrivastava, A.; and Song, Y. 2022. One Network Doesn't Rule Them All: Moving Beyond Handcrafted Architectures in Self-Supervised Learning. *arXiv preprint arXiv:2203.08130*.

Ha, D.; Dai, A. M.; and Le, Q. V. 2017. HyperNetworks. In *International Conference on Learning Representations*.

Han, K.; Wang, Y.; Zhang, Q.; Zhang, W.; Xu, C.; and Zhang, T. 2020. Model rubik's cube: Twisting resolution, depth and width for tinynets. *Advances in Neural Information Processing Systems*, 33: 19353–19364.

Han, S.; Meng, Z.; Li, Z.; O'Reilly, J.; Cai, J.; Wang, X.; and Tong, Y. 2018. Optimizing filter size in convolutional neural networks for facial action unit recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5070–5078.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hochreiter, S.; Bengio, Y.; Frasconi, P.; Schmidhuber, J.; et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, 1314–1324.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456. PMLR.

Jacobsen, J.-H.; Van Gemert, J.; Lou, Z.; and Smeulders, A. W. 2016. Structured receptive fields in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2610–2619.

Kabir, H. D.; Abdar, M.; Khosravi, A.; Jalali, S. M. J.; Atiya, A. F.; Nahavandi, S.; and Srinivasan, D. 2022. Spinalnet: Deep neural network with gradual input. *IEEE Transactions on Artificial Intelligence*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Liu, S.; Chen, T.; Chen, X.; Chen, X.; Xiao, Q.; Wu, B.; Pechenizkiy, M.; Mocanu, D.; and Wang, Z. 2022a. More ConvNets in the 2020s: Scaling up Kernels Beyond 51x51 using Sparsity. *arXiv preprint arXiv:2207.03620*.

Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; and Guo, B. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 10012–10022.

Liu, Z.; Mao, H.; Wu, C.-Y.; Feichtenhofer, C.; Darrell, T.; and Xie, S. 2022b. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11976–11986.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Mazzia, V.; Salvetti, F.; and Chiaberge, M. 2021. Efficient-capsnet: Capsule network with self-attention routing. *Scientific reports*, 11(1): 1–13.

Pintea, S. L.; Tömen, N.; Goes, S. F.; Loog, M.; and van Gemert, J. C. 2021. Resolution learning in deep convolutional networks using scale-space theory. *IEEE Transactions on Image Processing*, 30: 8342–8353.

Romero, D. W.; Bruintjes, R.-J.; Tomczak, J. M.; Bekkers, E. J.; Hoogendoorn, M.; and van Gemert, J. 2022a. FlexConv: Continuous Kernel Convolutions With Differentiable Kernel Sizes. In *International Conference on Learning Representations*.

Romero, D. W.; Kuzina, A.; Bekkers, E. J.; Tomczak, J. M.; and Hoogendoorn, M. 2022b. CKConv: Continuous Kernel Convolution For Sequential Data. In *International Conference on Learning Representations*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252.

Schütt, K.; Kindermans, P.-J.; Sauceda Felix, H. E.; Chmiela, S.; Tkatchenko, A.; and Müller, K.-R. 2017. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30.

Shelhamer, E.; Wang, D.; and Darrell, T. 2019. Blurring the line between structure and learning to optimize and adapt receptive fields. *arXiv preprint arXiv:1904.11487*.

Shi, S.; Wang, Z.; Shi, J.; Wang, X.; and Li, H. 2020. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE transactions on pattern analysis and machine intelligence*, 43(8): 2647–2664.

Simonovsky, M.; and Komodakis, N. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3693–3702.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Sukhbaatar, S.; Grave, E.; Bojanowski, P.; and Joulin, A. 2019. Adaptive Attention Span in Transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 331–335. Florence, Italy: Association for Computational Linguistics.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Tabernik, D.; Kristan, M.; and Leonardis, A. 2020. Spatially-adaptive filter units for compact and efficient deep neural networks. *International Journal of Computer Vision*, 128(8): 2049–2067.

Tan, M.; and Le, Q. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, 6105–6114. PMLR.

Thomas, N.; Smidt, T.; Kearnes, S.; Yang, L.; Li, L.; Kohlhoff, K.; and Riley, P. 2018. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*.

Tomen, N.; Pintea, S.-L.; and Van Gemert, J. 2021. Deep Continuous Networks. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 10324–10335. PMLR.

Tuggener, L.; Schmidhuber, J.; and Stadelmann, T. 2021. Is it enough to optimize cnn architectures on imagenet? *arXiv preprint arXiv:2103.09108*.

Wang, S.; Suo, S.; Ma, W.-C.; Pokrovsky, A.; and Urtasun, R. 2018. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2589–2597.

Xiong, Z.; Yuan, Y.; Guo, N.; and Wang, Q. 2020. Variational context-deformable convnets for indoor scene parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3992–4002.

Xu, A.; Huo, Z.; and Huang, H. 2020. On the acceleration of deep learning model parallelism with staleness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2088–2097.

Zagoruyko, S.; and Komodakis, N. 2016. Wide Residual Networks. In Wilson, R. C.; Hancock, E. R.; and Smith, W. A. P., eds., *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.