Convolutional Neural Networks without Pooling or Fixed Filter Sizes

Anonymous WACV submission

Paper ID 832

Abstract

An important parameter of Convolutional Neural Networks (CNN) is the size of the filters used in convolutions. State of the art approaches tend to use filters of the same size (e.g. 3×3), but adding pooling layers, which is equivalent to increase the effective *filter sizes (i.e. the size with respect to the initial image) by the pooling factor but with a reduced computational cost. However, the pooling layers position and therefore the effective filter size in CNN is normally predefined. In this paper we investigate the importance of learning the effective filter sizes and propose a way of doing it that: i) is differentiable and therefore the filter sizes can be learned jointly with the network weights in an efficient way ii) can select one or multiple filters size depending on the needs. Experimental results show that our approach can improve the performance of several networks configurations due to the optimal learned pooling strategy and the use (if needed) of multiple filter sizes.*

1. Introduction

CNNs have a large number of hyper-parameters to optimize. Even considering the simplest convolutional neural network, one should always define the number of layers, number of channels and the size of the convolutional filters. While for automatically selecting the number of layers (e.g. dynamic routing [22, 21]) and channels (e.g. structured network pruning [16, 5]) of a neural network quite some work has already been presented, the selection of the filter size has so far been overlooked. We believe this is mostly due to the difficulty of dealing with filters of different size. Being able to select the right filter size for each layer of a convolutional neural network (CNN) can help to obtain a better computation-accuracy trade-off and therefore more efficient models. Additionally, knowing what is the optimal filter size can help to better understand the way that convolutional neural network find meaningful patterns in images, and therefore what are the relevant features for a given task.

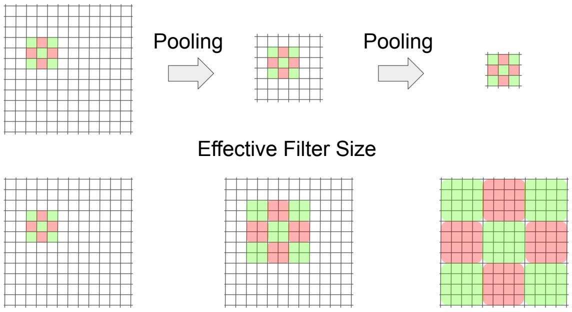


Figure 1: **Effective Filter Size.** CNNs typically interleave convolution layers using fixed-size filters (e.g. 3×3) with pooling layers (upper row). The filter size effectively increases respect to the original image.

In modern architectures, filters are normally of size 3×3, because it is the minimum size that allows the model to consider a symmetric spatial neighbourhood of a pixel and it has been shown that multiple layers of those filters can have the same receptive field as bigger filters [**?**]. Thus, the filter size does not seem an issue. However, as shown in Fig, 1 the 3×3 size represents the size of the filter with respect to the considered feature maps. What is instead more relevant, is the *effective* filter size with respect to the original image. This tell us the real size of the patches that we are considering in the image. As shown in Fig. 1 the size of the filter with respect to the real image is defined by the reduction of the feature map size, i.e. how many pooling layers have been used so far. Thus, in the rest of the paper we will consider learning the size of the convolutional filters or learning where to place pooling layers as equivalent problems.

In a different context, learning the filter size can be considered as a form of network architecture search. While most recent works in NAS have focused on learning the *micro* structure of a network [14, 24, 17], i.e the content of a set of pre-structured building blocks, in this work we aim to learn the *macro* structure of a network, i.e. where to pool layers. We argue that macro structure of a network is difficult to learn (especially for differentiable approaches) because for instance in our case, changing the position of the pooling layer in the network changes the size of a feature

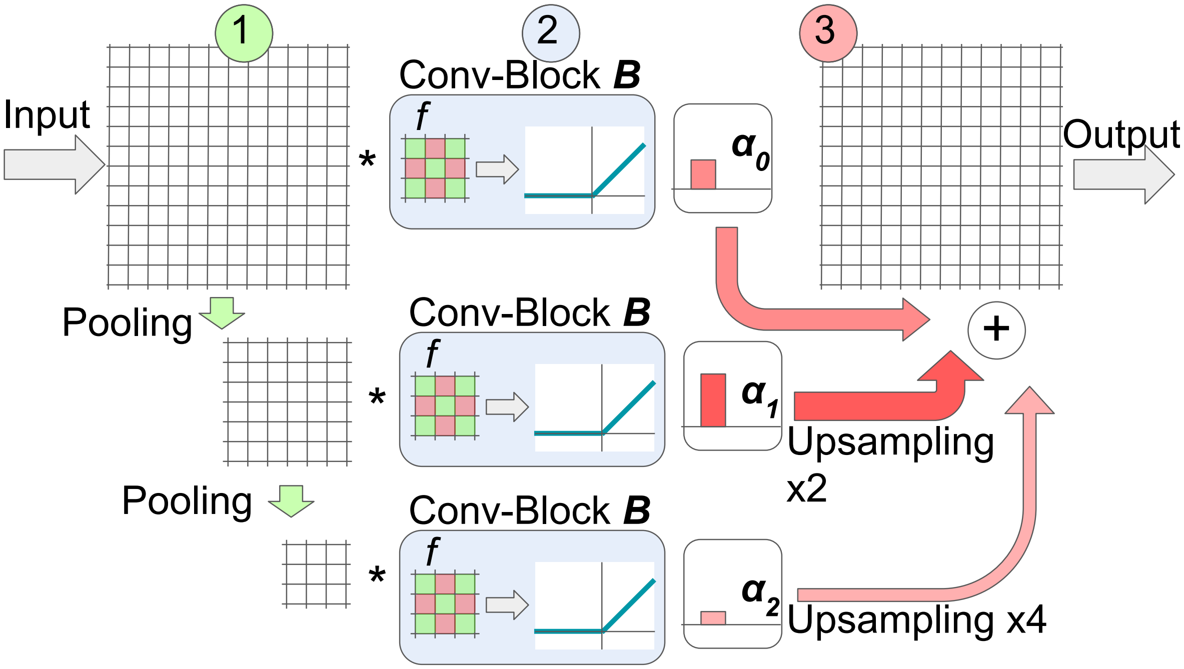


Figure 2: **Multiresolution Convolutional Layer** 1. The input feature map is downsampled to multiple resolutions via pooling. 2. A Conv-Block $\mathbf{B}$ is applied at each resolution. 3. The convolution results are weighted by mixing parameters $\{\alpha_0, \alpha_1, \alpha_2\}$, upsampled and summed to obtain the output feature map at the resolution of the original input.

map and therefore it invalidates all the subsequent blocks. Thus, in this approach we find a way to change the filter sizes without changing the feature map resolution.

As shown in Fig. 2, in order to learn the right filter size for each convolutional layer we convolve every feature map with the same set of filters at different resolution. Each resolution is then re-sampled to the maximum feature map resolution and associated to a normalized weight ($\{\alpha_i\}_{i=0}^{R}$ in the figure). Finally the output feature map is the sum of the weighted output of each resolution. This procedure is relatively simple and allows our model to have a generic representation with the same input and output feature map. The learned weights associated to each feature map resolution will tell us which resolution is the most appropriate for each layer. At the end of the training feature maps with low weights can be removed to obtain our final network architecture.

The contributions of this work are the following:

- We propose a simple approach to learn the size of the filters for each layer of a convolutional neural network, which is equivalent to select the position of the pooling layers in the network.

- The proposed approach is differentiable and the right network architecture can be learned jointly with the network weights.

- Experimental results with different convolutional basic blocks show that with the proposed approach by learning the right filter sizes can lead to architectures whose computational cost is comparable with the hand defined architectures but with improved accuracy.

The reminder of the paper is defined as follows. In related work we compare our approach with previous methods with a similar aim. Then in section 3 we describe our proposed method. Finally we assess the validity of the approach with a complete set of experiments and we draw conclusions.

2. Related work

Box Convolutions An previous attempt to learn the filter sizes was performed by Burkor and Lempitsky [1] who define filters locations as continuous values represented by a bi-linear interpolation of the discrete feature map values. In this way the filter dimensions are differentiable and can be optimized by back-propagation. By combining this with the integral image representation, they were able to compute any filter size with a fixed computational cost. Unfortunately, considering the overhead of bi-linear interpolation (one value is computed as weighted sum of 4 feature map locations) as well as the overhead of the pre-computation of the integral image, make the method too slow for generic convolutional filters (in their work they use simplified filters for segmentation).

Multi-resolution Representations Multiple resolutions and multiple scales have been broadly used in CNN. Here we limit ourselves to methods that are highly related to our approach. Previous work used a representation based on multiple resolutions of the features maps to improve the representation [9, 2], avoid predefined pooling layers [19] or reduce the computational cost [3, 8]. Ke *et al*. [9] proposed to extend every feature map of a CNN with a multiple resolution representation. Chen *et al*. [2] uses multiple branches to build a strong and slow (high resolution and many channels) and weak and fast (half resolution fewer convolutional layers) representation of the same model. Liu *et al*. [12] combine a two resolution representation in a single CNN layer for improved efficiency. These approaches show in different ways that the use of feature maps at multiple resolution improve the accuracy computation trade-off. However, those approaches still require predefined pooling layers.

Saxena and Verbeek [19] is probably the closest work to ours. They try to generalize the CNN structure by using a multi-resolution representation so that a explicit pooling is not needed. They use a dense multi-resolution representation in which all feature map resolutions are computed at the same time and each resolution communicates to the finer and coarser resolution by strided convolutions. This approach obtained excellent results, but it is computationally and memory-wise quite expensive because for each resolution three different convolutions are used (upsampling, same resolution, downsampling). In section 4 we will directly compare our model with theirs in terms of classification accuracy and performance. Their approach was subsequently extended to reduce its computational cost at inference [8]. The same model is also explicitly used for learning a network architecture for semantic segmentation [12].

Attention Our model has some resemblance with attention mechanisms too [23]. As in soft-attention, our method summarizes a set of representation into a learned weighted sum. In this case we merge representations at different feature map resolutions. In this case, in order to sum the different resolution we need to perform upsampling first. The most important difference of our method with respect to attention is that here the weights associated to every resolution are ”static”, in the sense that they are learned by backprop and do not change over time. In contrast, in attention the weighting parameters are estimated by another neural network and therefore they can change and adapt to the input. We consider the ”static” weights a better solution for our problem because finally we are interested in finding a ”static” sequence of filter size that is optimal for a certain dataset. However, we consider also the dynamic update of the weight an interesting future direction of research. In the same context, Jia *et al*. [6] developed a module called squeeze and excite that associate a weight to each filter channel for improved performance. Technically the two methods are quite similar, but applied to different problems.

Neural Architecture search The proposed approach can be seen as neural architecture search (NAS), for a very specific but important parameter of the network, the filters size. Most research on NAS focus on finding the inner structure of network [24, 13, 17] i.e. the operations in cells that are then stacked to form a network [14]. The outer architecture that controls the spatial resolution of the feature map across layers (or equivalently the filters size) by operations such as maxpooling or strided convolutions is designed manually. Thus in this work we focus only on the outer structure and we leave as future work the possible combination of the two. In terms of optimization, we use the same differentiable principle for architecture search as Darts [14]. Instead of learning weights for different networks branches, we learn weights for different feature map resolutions.

3. Learning Filter Size

In this section we present our approach to learn the filter sizes of a neural network in a differentiable manner. In the first part we introduce a formulation to describe a generic neural network and the connection between pooling and filter size, which is key for our approach. Then we formulate our multi-resolution convolutional block and how we learn filter sizes with it. Finally we show how to extend this approach to learn also the depth of a network.

3.1. Pooling and Filter Size

Let B be a function representing a basic convolution block, where the two inputs at layer l are a 3D feature map $h(x, y, c_{in})_l$ and a filter set $f(h, w, c_{in}, c_{out})_l$, and the output is a feature map $h(x, y, c_{out})_{l+1}$ serving as input of the next layer $l + 1$

$$h_{l+1}(x, y, c_{out}) = \mathbf{B}(h_l(x, y, c_{in}), f_l(h, w, c_{in}, c_{out})) \tag{1}$$

Our analysis is independent of the numbers of input and output feature channels c_{in} c_{out} at each block, and thus in the following, for the sake of clarity, our formulation of feature maps and filters is reduced to 2D spatial dimensions (x, y). Furthermore, it applies generally to a variety of block implementations, including typical convolution plus non-linearity to more complex architectures such as depthwise convolutions [18] or residual blocks [4] , and we thus omit adopting a particular block form.

A very generic convolutional neural network will be the combination of a interleaved sequence of convolutional blocks B and spatial pooling blocks S that receive a feature map $h(x, y)$ and reduce their spatial size, for instance by a stride 2 sub-sampling.

$$h_{l+1}(x, y) = \mathbf{S}_r(h_l(x, y)) = h_l(rx, ry) \tag{2}$$

Also for this block, we do not care about its exact content for the moment.

Notice that pooling can be seen as an efficient way to increase the effective filter size with respect the image. For instance, the convolution at double the filter size can be approximated by reducing the input feature map resolution by a factor of 2 via a pooling block, while requiring fewer parameters:

$$\begin{aligned} h_{l+1}(x, y) &= \mathbf{B}(\mathbf{S}_2(h_l(x, y)), f_l(h, w)) \\ &= \mathbf{B}(h_l(2x, 2y)_l, f_l(h, w)) \\ &\approx \mathbf{B}(h_l(x, y, c_{in})_l), f_l(2h, 2w,)) \end{aligned} \tag{3}$$

Thus, in this paper we will use a representation based on multiple feature map sizes to automatically learn filter size parameters, or equivalently select the best layers at which apply pooling.

3.2. Multi-resolution Convolution

An overview of our approach is shown in Fig.2. Our approach aim to learn the convolutional filter sizes by associating a learned parameter to each convolution block B at multiple resolution and select the best, i.e. the block that received more importance in the training procedure. For doing that we build a meta-block M defined as follows:

$$\begin{aligned} h_{l+1}(x, y) &= \mathbf{M}(h_l(x, y)) \\ &= \sum_{r=1}^{R} \alpha_{l,r} \mathbf{U}_{2^r}(\mathbf{B}(\mathbf{S}_{2^r}(h_l(x, y)), f_l(h, w))) \end{aligned} \tag{4}$$

This block applies the same convolution block $\mathbf{B}$ at different resolutions r and with the same filters f_l. The resulting feature maps are then multiplied by a normalized coefficient $\alpha_{l,r}$, re-scaled to the initial feature map resolution (x, y) resolution with an upsampling operation $\mathbf{U}$ and summed.

The normalized coefficient $\alpha_{l,r}$ learns the relative strength of a certain resolution with respect to the others for a given layer l and is computed as a softmax over feature map resolutions:

$$\alpha_{l,r} = \frac{\exp(\alpha_{l,r})}{\sum_s \exp(\alpha_{l,s})} \tag{5}$$

The downsampling and upsampling of the feature maps allow us to perform convolution at different scales in an efficient way. In fact, the most expensive operation is the convolution computed in each block B at the maximum feature map resolution. As in our model the feature map resolution is divided by two recursively, it easy to see that for each feature map the computational cost will be reduced by four times so that the final computational cost will always be less than two times the high resolution convolution. Also, the additional upsampling and down-sampling operations used to change the resolution of the feature map are negligible in terms of computation. With this method, we can consider the final filter size as a weighted combination of the size of each resolution, in which the weight is represented by $\alpha_{l,r}$. Once the model is fully trained it can be used directly or we can remove the useless layers with a final pruning of the with very low value and fine-tuning to let the network to fully adapt to the new setting. In the experimental evaluation (section 4) we will show results for both configurations.

3.3. Learning the network depth

The same relaxation technique that we use to learn the filter size can be applied to determine the length of the network. In this case, the input of the last layer L of the network is the weighted sum of the outputs of each layer. Similarly, mixing weights are determined by parameters β that are learned jointly with α on the validation set.

$$\begin{aligned} & h_L(x, y) = \sum_{l=1}^{L} \beta_l \mathbf{B}^M(h(x, y)_l) \\ & = \sum_{l=1}^{L} \beta_l \sum_{r=1}^{R} \alpha_l, r\mathbf{U}_{2^r}(\mathbf{B}(\mathbf{S}_{2^r}(h(x, y)_l), f_l(h, w))) \end{aligned} \tag{6}$$

One possible drawback of this approach is the need of using the same number of channels for every layer in order to sum them up at the last layer. There can be ways to avoid this constraint, but we leave this for further research. In the experimental section we will show that is possible to learn the filter sizes and the network depth at the same time with a single training.

3.4. Training

The training of our network is separated in two parts: i) train the neural network filters f_l ii) train the mixing weight associated to the filter sizes ($\alpha_{l,r}$) (and β_l if needed). In order to avoid to learn weights associated to filters that overfit the training data, similarly to [14], we split the training data $\mathcal{D}$ in two parts: $\mathcal{D}_a$ and $\mathcal{D}_b$. The first part is used for learning f_l while the second is used to learn $\alpha_{l,r}$. Training is perform as usually in mini-batches to find the right trade-off between memory consumption and parallelization. At each training iteration we interleave a randomly sample mini-batch from $\mathcal{D}_a$ for updating f_l and a mini-batch from $\mathcal{D}_b$ for updating $\alpha_{l,r}$. The training loss is the standard cross-entropy loss between the estimated classes and the ground truth class labels. The mixing weights are initialized with the same value at the beginning of the training.

3.5. Sharing filter weights

For this joint learning, in order to obtain good results it is very important to avoid the α_l, r to fall in wrong configurations from which is difficult to recover (e.g. using in the first layers low resolution maps and therefore loosing high frequency information). For this reason, we decided to use the same filters f_l at each resolution. We initially experimented with different filters for each feature map resolution $F_{l,r}$, but we noticed that i) the initial stage of the training is very slow due to the fact that different layers contain different information on different filters and when they are summed they should learn to agree in the represented information ii) the training can fall in local minimae due to the fact that a filter at a given resolution can quickly become more important than the others and bias the training. Using the same filters for each resolution for each layer is therefore preferred because i) it implicitly the summed channels agree at each resolution; ii) it is a form of regularization, in fact, at the beginning of the training when the mixing weights are uniform, it corresponds to a scale invariant network; iii) sharing the weights reduces the number of parameters and therefore it reduces overfitting certain resolutions.

4. Experiments

We evaluate our method on the CIFAR10 dataset [11]. We first compare our approach with the most significant baselines in order to show the advantage of our approach. Then we show the learned networks and the corresponding performance for several configurations and two basic blocks. Finally, we compare our approach with FabircNet, the only previous work that also aimed to not use pooling layers.

4.1. Implementation details

CIFAR10 is composed of $60,000$ 32×32 images of 10 object categories, divided into training and testing sets of sizes $50,000$ and $10,000$ images. We further split the training set into $40,000$ images for training the network parameters and $10,000$ for estimating the values of alphas. We train the network with stochastic gradient descent (SGD) with cosine annealing learning rate decay [15] for 400 epochs. For learning alphas, we use ADAM [10] optimizer with initial learning rate 10^{-4}, momentum $(0.5, 0.999)$ and weight decay 3×10^{-4}. The training and learning alphas are performed iteratively.

The basic block of our model is Conv-ReLU-Batchnorm with a constant number of channels across the layers and a filter size of 3×3. For Eff-Conv in table 4, we used MBConv6 [20] with 3×3 filter size. MBConv block consists of an 1×1 expansion convolution that expands the number of channels by factor of 6, followed by a 3×3 depthwise convolution. Squeeze and excitation [7] is then followed by another 1×1 convolution.

4.2. Baselines

In Table 1 we compare our approach with several baseline configurations in terms of classification accuracy and computation (in FLOPS). All network configurations used for this experiment have 10 layers with 64 channel per layer and a total of $0.3M$ parameters.

The first is a configuration with no pooling in table (row 1), to show the importance of pooling in terms of both accuracy and computational cost. With no pooling, the effective filter size is the same at each layer and the network performance is reduced. At the same time, as we do not reduce the feature maps until the last layer (at which we still use an average pooling to convert the feature map in a single vector), which increases the computational cost of the method.

Our second baseline is a predefined pooling network, *i.e.* a classical network configuration. To build this configuration we use the rule of thumb of decreasing the feature map resolution every 2-3 layers. The exact network configuration is shown in Fig. 3(a). As shown in the table (row 2), this network is efficient and performs relatively well. However, it is not clear in which situations this rule of thumb applies. There might be other network configurations (in terms of pooling) that can provide better results.

The third baseline is a network with a fixed, uniform mixing weight $\alpha_{l,r}$ structure. We call this model scale invariant (row 3) because the convolution blocks B_l apply the same filters at different scales and accord equal importance all feature map resolution. This would in principle allow the model to detect objects at different scales. This is a desired property for certain tasks (e.g. in object detection we want to detect object at any scale). However, for classification normally there is some bias due to the scale of the object in the image. Forcing the network to be scale invariant discards this bias information, and leads to a reduction of performance by 0.8%.

Finally we present our model in four different flavors. First of all we present the performance of our model after training (*i.e.* keeping all the feature map resolutions). The learned configuration of the network is shown in Fig. 3(b). In this case the computational cost is the same as the scale invariant model, so around X times the computational cost of our pre-defined model. However this extra cost comes with a neat improvement in performance, obtaining a classification accuracy of 92.5%. This is due to the fact that the network has learned the optimal filter sizes. In this case, as we keep all the resolutions with associated weights, we can think of the learned filter as a continuous value that is the weighted combination of the discrete filter sizes used at the different feature map resolutions.

To reduce the computational cost of our approach, we can remove the feature map resolutions (or equivalently filter sizes) with low alpha mixing weights following learning, and fine tune the new configuration. We consider two different ways of doing this. First we use a fixed threshold on mixing weights. In the table we consider two values for the threshold: 0.1 (row 5, and Fig. 3(c) to see the learned weights) and 0.2 (row 6). As the mixing weights are normalized, these thresholds correspond to remove all the feature maps resolutions that contribute either 10% or 20% of the entire layer. With this approach we can highly reduce the computational cost of our network while keeping almost the same results (same accuracy for 10% pruning, loss of $0.X$ for 20% pruning). We also consider the case of keeping only the best resolution and retrain the model (row 7, and Fig.3(d)). This is an interesting configuration because it corresponds to a typical generic CNN in which only one feature map resolution per layer is used. In terms of filter size, in this case we can think that we quantize the size of the filter to one of the discrete sizes used during training. Note also that in this case, our model is equivalent to a typical CNN, and the technique of upsampling and summing feature maps across resolutions becomes unnecessary.

We can see that in this setting the model performs still well, but slightly worse than before, losing 1.4% points. There are two things to note here regarding the improvement of our full model vs. reduced variants. First, that learning optimal locations for pooling (equivalently choosing the right filter size) is important and poorly chosen configurations lead to worse performance. We were not able to identify predefined configurations improving upon our model (one res.) Secondly, as multi-resolution filtering improves performance, the salient image information at a given layer is generally spread across more than one scale. An example of a recent work exploiting this idea is OctaveNet [3] in which the authors use a convolutional layers with two feature resolutions for an improved performance-computation trade-off.

	Acc.	MFLOPS
No Pooling	87.3	0.31
Predefined Pooling	90.0	0.26
Scale Invariant	89.2	0.41
Our Model	92.5	0.41
Our Model (thr 0.1)	92.5	0.29
Our Model (thr 0.2)	92.3	0.27
Our Model (one res.)	91.1	0.26

Table 1: **Comparison with baselines.** We compare the accuracy of our model with some baseline approaches on CIFAR10.

4.3. Number of Channels

We evaluate the effect of varying the number of channels on the proposed approach. Again, we use 3 different baselines to have reference points for comparison. From table 2 we can see that increasing the number of channels helps for all settings. Also in this case the relative ranking of the different configurations is very similar to the previous experiment. Something interesting to notice here is the fact that our model as well as the scale invariant version seem to improve more by adding more channels than the predefined model. This might be because scale invariant and our model use the same filters for multiple resolutions. Thus, when there are more filters if using more resolutions, some filters can adapt to one scale and others to another. In contrast, when the number of channels is relatively low (e.g. 16), it may be insufficient to represent the multi-resolution structure and therefore the performance of the method is reduced.

	C=16	C=32	C=64
No Pooling	71.8	82.8	87.3
Scale Invariant	78.2	84.0	89.2
Predefined Pooling	83.3	89.0	90.0
Our Model	83.9	89.9	92.5

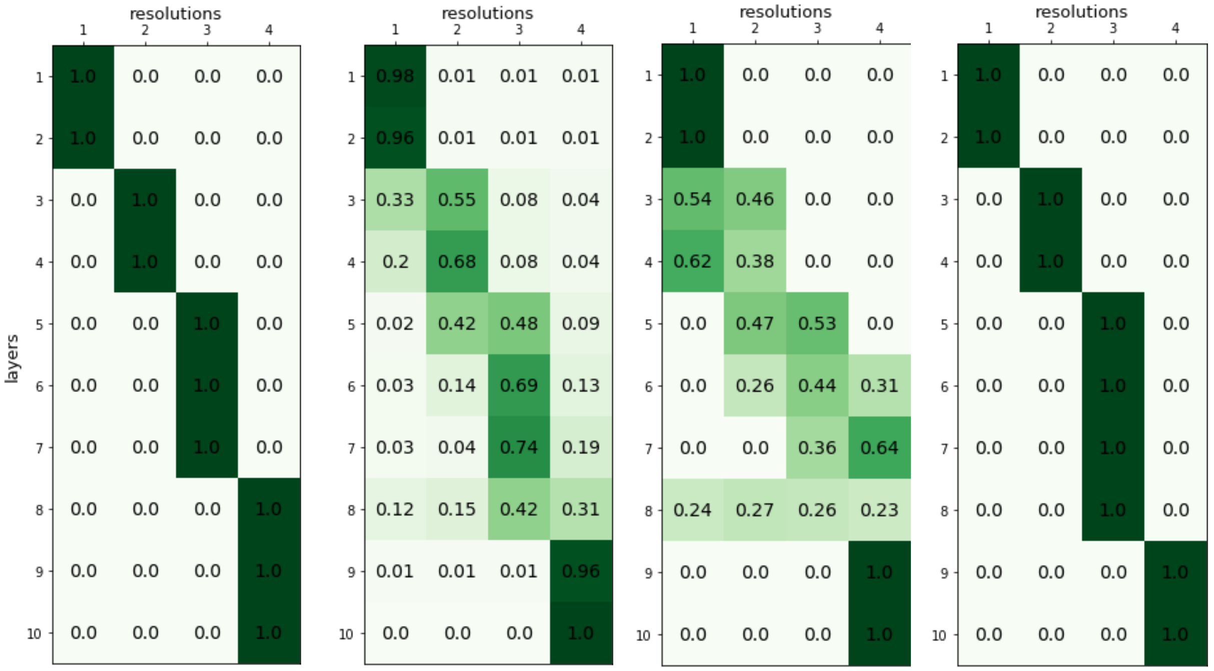
Table 2: **Number of channels.** We compare the accuracy of our model with 10 layers with some baselines with different numbers of channels on CIFAR10.

4.4. Learning Filter size and depth

In this experiment we want to learn not only the filter sizes of the network but also its the depth. For doing that we use the formulation in equ.6 and we test it on two different networks depths:$D = 10$ and $D = 20$ layers. Table 3 summarizes the results of this experiments. The learned depth (LD) for the two configurations is quite similar. For the network with 10 layers our algorithm learns that 8 layers are sufficient, while for the network of 20 layers it learns that 9 layers are sufficient. This is promising because we expect that the optimal depth of the network should not depend on the maximum allowed depth. However, for both networks, when learning the depth, the accuracy of the network drops. In this experiment we do not prune the connection that have low value. Thus, this lower results can be caused by to the overlap of the different paths that connect each layer to the final one. Further investigation is required.

	D=10	LD	D=20	LD
No Learning Depth	92.5	-	92.9	-
Learning Depth	92.0	8	91.5	9

Table 3: **Network depth** on CIFAR10.



(a) Predefined

	1	2	3	4
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0
4	0.0	1.0	0.0	0.0
5	0.0	0.0	1.0	0.0
6	0.0	0.0	1.0	0.0
7	0.0	0.0	1.0	0.0
8	0.0	0.0	0.0	1.0
9	0.0	0.0	0.0	1.0
10	0.0	0.0	0.0	1.0

(b) Learned

	1	2	3	4
1	0.98	0.01	0.01	0.01
2	0.96	0.01	0.01	0.01
3	0.33	0.55	0.08	0.04
4	0.2	0.68	0.08	0.04
5	0.02	0.42	0.48	0.09
6	0.03	0.14	0.69	0.13
7	0.03	0.04	0.74	0.19
8	0.12	0.15	0.42	0.31
9	0.01	0.01	0.01	0.96
10	0.0	0.0	0.0	1.0

(c) Learned (thr 0.1)

	1	2	3	4
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	0.54	0.46	0.0	0.0
4	0.62	0.38	0.0	0.0
5	0.0	0.47	0.53	0.0
6	0.0	0.26	0.44	0.31
7	0.0	0.0	0.36	0.64
8	0.24	0.27	0.26	0.23
9	0.0	0.0	0.0	1.0
10	0.0	0.0	0.0	1.0

(d) Learned (one res.)

	1	2	3	4
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0
4	0.0	1.0	0.0	0.0
5	0.0	0.0	1.0	0.0
6	0.0	0.0	1.0	0.0
7	0.0	0.0	1.0	0.0
8	0.0	0.0	1.0	0.0
9	0.0	0.0	0.0	1.0
10	0.0	0.0	0.0	1.0

Figure 3: **The mixing weight structure of four models used in baseline comparisons.** Each grid represents the mixing weight structure $\alpha_{l,r}$ applied to filters across layers and resolutions for the particular model. Models include a) predefined architecture, one resolution per layer. b) Our full learned model and reduced variants c) thresholded at $\alpha_{l,r} \geq 0.1$ and d) one resolution per layer. Note how in our learned model a), the distinct diagonal pattern that emerges is analogous to the fine-to-coarse structure of filter size found in typical predefined CNN. However in our model, filter weight is spread across multiple resolutions at each layer, rather than a single fixed resolution.

4.5. Comparison with FabricNet

We compare our method with FabricNet [19], the closest work to ours. In table 4 we compare the two results obtained by Saxena and Verbeek with our method on CIFAR10 in terms of classification error and number of parameters. For our method for completness we report also computation in terms of millions of FLOPS. For our method we evaluate three different models. The first one is the model that we use for the previous experiment with 10 layers and 64 filters. It performs comparable to the best model of [19] but with a fraction of their parameters. Our model is much more efficient in terms of parameters because it reuses the same filters on every resolution. In contrast FabricNet uses different filters for each resolution. Furthermore, it uses additional convolutions and therefore more parameters for upsampling and downsamping feature map. We also evaluate a version with the 8 layers and 128 filters. In this case our accuracy is inferior to FabriNet. Finally we also evaluate a version with an efficient convolutional block [20] (see implementation details) and again the same number layers and filters. In this case, with a still very reduce number of parameters we obtain a better performance than FabricNet.

5. Conclusions

In this paper we have presented a method for learning the effective filter size in a CNN or equivalently learn where the place the pooling layer. The method is based on estimating the importance of feature maps resolution thought weights that can be learned by back-propagation. In this way we have made the problem differentiable and the estimation of the weights can happen jointly with the learning of the CNN weights. We have evaluated our approach with different networks and it is always able to estimate a configuration for the pooling layers that is comparable or better than a predefined one. Finally we have shown that the proposed approach is simpler and faster than previous approaches, while providing better results.

References

[1] E. Burkov and V. Lempitsky. Deep neural networks with box convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Ad-*

Method	Block Type	# Filters	# Layers	Error	Params	MFLOPS
FabricNet [19]	Sparse-Conv	64	16	18.89	2M	-
	Conv	128	8	7.43	21.2M	-
Ours	Conv	64	10	7.5	0.3M	0.41
	Conv	128	8	8.8	0.9M	1.23
	Eff-Conv	128	8	7.0	1.5M	1.69

Table 4: **Comparison with FabricNet** We compare the accuracy of our model in different configurations with FabricNet on CIFAR10.

vances in Neural Information Processing Systems 31, pages 6211–6221. Curran Associates, Inc., 2018. 2

[2] C.-F. R. Chen, Q. Fan, N. Mallinar, T. Sercu, and R. Feris. Big-little net: An efficient multi-scale feature representation for visual and speech recognition. In *International Conference on Learning Representations*, 2019. 3

[3] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. *CoRR*, abs/1904.05049, 2019. 3, 6

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[5] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017. 1

[6] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. 3

[7] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 5

[8] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018. 3

[9] T. Ke, M. Maire, and S. X. Yu. Neural multigrid. In *CVPR*. 3

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[11] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). 5

[12] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019. 3

[13] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 3

[14] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 3, 4

[15] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. 5

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. In *ICLR17*, 2018. 1

[17] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 1, 3

[18] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 3

[19] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pages 4053–4061, 2016. 3, 7, 8

[20] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR. 5, 7

[21] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. *International Journal of Computer Vision*, Jun 2019. 1

[22] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. S. Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018. 1

[23] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR. 3

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 3