The Myth of the Pyramid

Ramon Izquierdo-Cordova Walterio Mayol-Cuevas
Department of Computer Science
University of Bristol
United Kingdom
izquierdocr@outlook.com, walterio.mayol-cuevas@bristol.ac.uk

Abstract

A deep-rooted strategy for building convolutional neural networks in computer vision is to increase the number of filters every time the feature map resolution is decreased. The notion ruling this pyramidal design is that the expressivity of the network increases with a higher number of filters to compensate for losses caused for lower resolutions. This paper challenges the practice by testing a set of variate distribution of filters, named filter templates, on popular CNN architectures (VGG, ResNet, MobileNet and MnasNet). The experimental results show that the superiority of the pyramidal design holds on the ImageNet dataset but fails for other datasets such as MNIST, CIFAR and TinyImageNet, and for other tasks such as audio classification. CNN models with different filter distributions deliver higher accuracy with reduced resource consumption suggesting the pyramidal design has been optimised to Imagenet and that each model-dataset pair benefits from tuning the number and distribution of filters. To further illustrate the benefits of exploring other distributions, this paper shows that the best performing model from the NASBench101 dataset can increase its accuracy over the original pyramidal design with reductions of parameters up to 68 per cent by using templates. Overall, our experiments point to new opportunities for model designers to find more efficient models.

1. Introduction

Convolutional Neural Networks (CNNs) have been notably successful in many domains. However, after all the continuous progress in CNN models, an element in their design remains almost unchanged. That is, the practice of increasing the number of filters in deeper layers, and essentially doubling the number of filters when a pooling layer halves the resolution of the feature map. This pyramidal design is rooted in the philosophy of deep learning, aiming to build hierarchical representations, reusing simple concepts in lower levels to form complex higher-level ones [4, 18]. It is generally believed that a progressive increase in the number of kernels compensates for a possible loss of the representation caused by the spatial resolution reduction [31], as well as improves performance by keeping a constant number of operations in each layer [7].

This pattern was first proposed in [31] with the introduction of LeNet and can be observed in a diverse set of models such as VGG[46], ResNet[23] and MobileNet[25]. Even models obtained from neural architecture search (NAS), such as NASNet [59], follow this principle since many automatic model discovery methods are mainly formulated to search for layers and connections. Note that all the models explored use the pyramidal design.

Since LeNet's emergence in 1998, there has been overwhelming adoption of the pyramidal filter distribution. It has been taken for granted that this design is optimal for all models and datasets, at least in the computer vision domain. Recently, researchers [13, 44, 49] are looking back to the uniform distribution of filters from the 1980 Fukushima's Neocognitron design [16]. They underline that, by following a uniform pattern, it is possible to achieve higher performance and structure simplicity, making architectures easier to implement [44, 49] . Our work calls for extending the search space to other filter distributions that can produce more efficient models.

To challenge the myth that the pyramidal distribution is of universal applicability, we evaluate a small set of predefined distributions, that we call templates. These are straightforward linear and piece-wise linear functions that can be easily implemented in most of the existing classical CNN and state-of-the-art models. Furthermore, we complement the definition with a fast method to match the number of floating point operations (FLOPs) of the original design (or any other FLOPs budget), allowing for fair comparison of models.

To use a template, our method takes a base model and redefines its filter distribution with a new pattern, while keeping the rest of the model unchanged. The process preserves the number and types of layers, including pooling ones. Therefore, feature map resolutions at each level are those of the original model. Once trained, the new model requires similar FLOPs.

Templates can easily by applied to a wide range of existing CNNs excepting few cases. For example, the PyramidNet architecture [22] is designed with zero-padded shortcut connections within blocks which only allows constant or incremental distributions of filters.

Experimental evidence shows that simple changes to the pyramidal distribution of filters can improve accuracy while reducing the number of parameters and/or memory footprint. Experiments also highlight that our tested models, although significantly changed in their original filter distribution, exhibit high resiliency w.r.t. accuracy, a phenomenon that requires further investigation. This hints to greater freedom to, for example, choose an appropriate number of channels given constraints, without significantly sacrificing performance.

2. Related Work

The evolution of neural networks has been described in numerous sources e.g. [2, 3, 33, 40, 42, 45]. They usually describe features extrinsic to network architectures such as activation and loss functions, parameter optimisation or regularisation; and architectural innovations, such as multi-path modules, deeper layers, residual connections and grouped convolutions [27]. Something that is less covered are the designers' arguments for choice of number of filters within layers. We found that from LeNet [31] almost all subsequent convolutional models have been using a pyramidal filter distribution. As far as we know, besides some limited exploration experiments performed only on ImageNet [41] with the ZFNet model [56], there is no other justification for universally adopting this incremental distribution than "to keep the richness of the representation" [31].

2.1. Recent Architectures With A Uniform Distribution of Filters

We note some works have started to revisit one other popular distribution, the uniform distribution introduced in Fukushima's Neocognitron [16]. Examples include: the Isometric Neural Network (INN) [44], that argues that the significance of the internal resolution of hidden layers (internal feature map resolutions) is more crucial than the resolution of the input image; the Vision Transformer (ViT) [13], composed of constant size transformer encoders keeping equal resolution throughout all layers; the Convolutional Mixer (ConvMixer) [49], with a similar filter distribution approach adopted resulting in a relatively simple architecture outperforming ViT and ResNet on ImageNet; and the work in [26] where other distributions of filters other than the pyramidal are evaluated on image classification tasks.

These works underline that a different distribution, mostly uniform in these cases, can achieve performance gains and structure simplicity. Our work calls for extending the search space to other filter distributions that can produce more efficient models.

2.2. Neural Architecture Search

Modern algorithms for neural architecture search are intended to produce high performing models with minimal human interaction [14, 58]. One of the biggest challenges in NAS is the infinite search space. A frequent solution is to take a subset of the possible values of the elements of the architecture, such as types of layers, number of filters and interconnections. Even so, the problem remains complex [8, 15, 24] and many approaches rely on previously published search spaces [35, 48]. Interestingly, the search space for most NAS methods is restricted to different sets of layers and their connections. However, the filter distribution in each layer follows the pyramidal pattern.

Exceptions are the new methods for channel number search (CNS) designed to automatically find the best number of filters for each layer in a neural network [19, 32]. Yet, most of the architectures used as base models to initialise the automatic search in CNS methods share the practice of increasing filters [5, 12, 52, 55] resembling the LeNet design[31].

3. Defining a Set of Filter Distribution Templates

To define the set of templates to investigate filter distribution effects, we are inspired by findings of PyramidNet [22] that presents two variations to the incremental pattern that increase accuracy in a modified ResNet architecture. The authors propose to use multiplicative and additive increases in the number of filters making the transitions of feature maps dimensions between layers smoother than in the original pattern. This is motivated by the harmful effects of sharply increasing filters between blocks, a phenomenon that has only been studied in residual networks. However, we have found that the effect extends to plain architectures too. Thus, to define our templates, we chose to use the additive pattern, the most successful of the two, and gradually change the number of channels across the architecture with linear segments.

We selected as the first template, one with the same incremental distribution but with a smooth step **(a)** (see Figure 1right). The second template is a distribution with a constant number of filters **(b)** as in the original Neocognitron. Another immediate option, contrary to the increasing distribution, is a decreasing distribution of filters **(c)** [51]. Finally, inspired by the distributions of blocks from the resulting ResNet101 and VGG models found in [19] and [30, 54], we

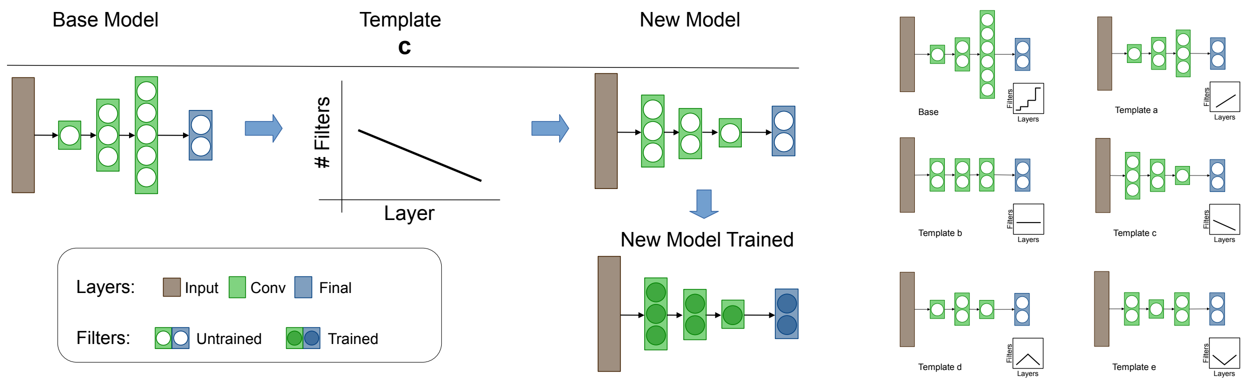

Figure 1. The pyramidal pattern of increasing filters per layer is widely adopted in convolutional models (left). We propose to apply a small set of templates to an existing model to change its filter distribution (right). After being trained from scratch, resulting models are competitive in accuracy compared to the original model, however, they require less computational resources.

define a template in which filters agglomerate in the centre, increasing the internal resolution **(d)** and, on the contrary, where filters are reduced in the centre of the model **(e)**.

One first approach to implementing a change of filters in a model is to keep the original number of filters in the resulting model and redistribute them differently across its layers. Nevertheless, final models end up with different resource demands and therefore making a fair comparison is problematic. Parameters, FLOPs and inference time have been used as a proxy for comparing models with different designs [6]. We believed that using one metric is insufficient for a fair comparison. As an example, for processing the CIFAR datasets, our implementations of VGG and ResNet count an approximate number of parameters (20.03 million versus 23.52 million, respectively), but ResNet's FLOPs (1307 MFLOPs) are more than three times VGG ones (399 MFLOPs). To facilitate comparisons, we match one metric while comparing the other. In particular, we fix models obtained from templates to match the number of FLOPs of the original distribution and then compare a second metric such as parameters, memory footprint or inference time.

In a more formal way, we define a convolutional neural network base model as a set of numbered layers $L = 1, ..., D+1$, each with f_l filters in layer l. $D+1$ is the final classification layer whose size is given by the task. The ordered set of all filters in the model is $F_{1:D} = \{f_1, \dots, f_D\}$ and the total number of FLOPs, the resource to be matched between templates, is given by some function $\mathcal{R}(F_{1:D})$. We want to find a new distribution of filters $F'_{1:D}$ in which

$$\mathcal{R}(F_{1:D}) \approx \mathcal{R}(F'_{1:D}) \tag{1}$$

and to test if the common heuristic of distributing $F_{1:D}$ having $f_{l+1} = 2f_l$ each time the feature map is halved, is advantageous to the model over $F'_{1:D}$ when evaluating performance, memory footprint and inference time.

Our templates are defined as simple linear segments, or a combination of them, in which $min(F'_{1:D}) = min(F_{1:D}) = n_{min}$ and $max(F'_{1:D}) = n \in \mathbb{N}$ satisfying constrain (1).

3.1. Similar FLOPs Optimisation

CNS methods aim to find individual values for the number of filters in each layer, and thus, exploration space becomes huge. For our method, the optimisation of the values is eased by the constrain in (1) and the way the templates are defined. Given that they are built with linear segments, only two natural numbers are required to compute the number of filters in each layer. One is fixed (n_{min}), and it is found by taking the lower number of filters generally in the first hidden layer of the original model. To find the second (n), we rely on the monotonic relationship between n and the model's FLOPs, valid for all templates. We use a simple binary search starting with the maximum number of filters in the original model and then reducing or increasing the value of n depending if the modified model has more or less FLOPs than the original. For obtaining the number of filters in intermediate layers in each linear segment, we round the evaluation of the linear equation produced by n_{min} and n according to the layer position within the segment. The only particular change in the method is made when using a template with a uniform pattern, in which case we make $n_{min} = n$. The whole procedure is performed before training, carrying minimal computational costs for redefining the model and estimating the new FLOPs value. We provide a precise value of filters for each layer of all models and templates tested in this work in Tables 7, 8, 9 and 10 of the appendix.

4. Templates on Image Classification

We investigated the effects of applying different templates to the distribution of filters in well known CNNs. We highlight that the resulting models produced from templates have similar FLOPs to the original model from which they are obtained.

4.1. Datasets and Models

We selected MNIST, FashionMNIST, CIFAR-10, CIFAR-100 [28], CINIC-10 [10], Tiny-Imagenet [29], and ImageNet [41]. Apart from the latter, these are datasets with diverse number of samples and classes allowing fast training. The first four datasets contain sets of 50,000 and 10,000 samples for train and validation, respectively. MNIST and Fashion-MNIST contain 28 x 28 grayscale images divided into 10 classes each. CIFAR datasets have associated labels from 10 and 100 classes and colour images with a resolution of 32x32. CINIC-10 contains 90,000 images in each, the training and validation sets with the same resolution and classes as the CIFAR-10 dataset. ImageNet is a 1000-class dataset with more than one million variable-size images. Tiny-Imagenet is a reduced version of the original Imagenet dataset with only 200 classes and images with a resolution of 64 x 64 pixels.

We evaluated VGG[46] and ResNet[23] models, which represent some of the most influential CNN architectures on the ImageNet challenge in previous years [9, 41] as well as MobileNetV2[43], one highly optimised model, and MnasNet[48], an automatically produced architecture from a NAS method.

4.2. Implementation Details

Experiments have models fed with images with the standard augmentation techniques of padding, random cropping and horizontal flipping and excepting ImageNet, with cutout [11] using one patch of 16 x 16 pixels. Our experiments were run in an NVidia Titan X Pascal 12GB GPU adjusting the batch size to 64 for TinyImagenet and 256 for the rest of the datasets. For ImageNet we utilised one node with two NVidia P100 GPUs.

Models on MNIST-like datasets were trained for 150 epochs using stochastic gradient descent (SGD) with a scheduled learning rate of 0.01 decreased with gamma 0.2 at epochs 75 and 110; weight decay of 1e-5 and momentum of 0.9. For CIFAR and CINIC-10, models were trained for 200 epochs using the same conditions: SGD with a learning rate of 0.1 scheduled with gamma 0.2 at epochs 60, 120 and 160; weight decay of 1e-5 and momentum of 0.9. For ImageNet and TinyImagenet, models were trained for 90 epochs using SGD with a scheduled learning rate of 0.1 decreased with gamma 0.1 at epochs 45, 70 and 85; weight decay of 1e-1 and momentum of 0.9.

4.3. Effects of Templates on Classical Models

Table 1 shows properties of resulting models for each architecture after using templates. Parameters, memory footprint and inference time are reduced in almost all cases. These results already show the effect of different distributions despite the template patterns were selected following simplicity and diversity but not focused on efficiency.

In particular, for the classical models, we observe increases in accuracy up to 2.11 percent points over the VGG base model primarily obtained with template **d**. Reductions of 90% in parameters, 79% in memory usage and 22% in inference time are produced by using template **c** while accuracy is still slightly superior on all datasets. The fastest model with a reduction of 24% in inference time is reached with template **b**.

The behaviour for ResNet differs from that of VGG in some aspects. The impact on resource consumption is lower. Template **c** shows savings of 85% in parameters, 30% in memory usage and almost 20% in inference time. The highest accuracies are obtained in half of the datasets by template **a**. The smallest model in memory is obtained with template **d** reaching maximum accuracy in CIFAR datasets with 32% less memory.

We also note a pattern behaviour related to each template. Accuracy improves in many datasets with templates **a** and **d**. Template **b** emerges as a good trade off between resource consumption and accuracy and templates **c** and **e** give the biggest reduction in resources by sacrificing some accuracy. We provide plots of our experiments with classical models in Figure 3 of the appendix.

4.4. Effects of Templates on Optimised Models

MobileNet and MnasNet architectures were optimised to perform well on mobile devices focusing on obtaining high accuracy and low inference time. The former was optimised by experts and the latter was optimised through a neural architecture search method. We show resource demands of both base models and their modifications produced by templates in Table 1. Although it is expected that the margin of improvement on these highly optimised models be considerably lower, we found that templates can reduce resource demands up to 77% in parameters and 11% in memory footprint.

Inference time depends more on the degree the computational graph allows parallelism. Our method keeps the same layer distribution and interconnection in the model. Thus, the computational graph remains similar. We also keep similar FLOPs for all the templates in our experiments. We think the difference in inference time is due to how well the size and number of feature maps fit the GPU memory. Although MobileNet and MNASNet more optimised, models with templates reach almost 5% of reduction in inference time.

We observe that template **e** produces the biggest savings in parameters and memory for MobileNet while still surpassing the base accuracy. The highest performance is obtained

Table 1. Resource consumption on CIFAR-10 for original architectures and resulting models after applying templates. FLOPs remain similar in comparison to base models.

Template	Param (Millions)		Mem (MB)		Inference Time (ms)	
vgg19 base	20.03	% ↓	87.0	% ↓	1.85	% ↓
vgg19 a	17.23	13.9	76.5	12.0	1.85	0.0
vgg19 b	3.17	84.1	23.0	73.5	**1.40**	**24.3**
vgg19 c	**1.89**	**90.5**	**17.8**	**79.5**	1.43	22.7
vgg19 d	8.07	59.7	39.8	54.2	1.46	21.0
vgg19 e	2.06	89.7	**17.8**	**79.5**	1.43	22.7

Template	Param (Millions)		Mem (MB)		Inference Time (ms)	
resnet50 base	23.52	% ↓	185.5	% ↓	5.35	% ↓
resnet50 a	14.17	39.7	146.8	20.8	4.83	9.7
resnet50 b	4.85	79.3	132.1	28.7	**4.29**	**19.8**
resnet50 c	**3.48**	**85.2**	128.9	30.5	4.30	19.6
resnet50 d	8.36	64.4	**125.8**	**32.1**	4.32	19.2
resnet50 e	3.68	84.3	132.1	28.7	4.34	18.8

Template	Param (Millions)		Mem (MB)		Inference Time (ms)	
mobilenet base	2.23	% ↓	28.3	% ↓	3.81	% ↓
mobilenet a	1.42	36.3	27.2	3.8	3.86	-1.3
mobilenet b	0.80	64.1	28.3	0.0	3.91	-2.6
mobilenet c	0.59	73.5	27.2	3.8	3.71	2.6
mobilenet d	1.12	49.7	27.2	3.8	**3.68**	**3.4**
mobilenet e	**0.51**	**77.1**	**25.1**	**11.3**	3.92	-2.8

Template	Param (Millions)		Mem (MB)		Inference Time (ms)	
mnasnet base	3.11	% ↓	**82.8**	% ↓	3.79	% ↓
mnasnet a	1.57	49.5	98.5	-18.9	3.68	2.9
mnasnet b	1.05	66.2	101.7	-22.8	3.85	-1.5
mnasnet c	**0.79**	**74.5**	101.7	-22.8	3.82	-0.7
mnasnet d	1.14	63.3	98.5	-18.9	**3.61**	**4.7**
mnasnet e	0.93	70.0	100.6	-21.4	3.75	1.0

with templates **a** and **b** for this model. For MnasNet, template **a** emerges as the best one regarding accuracy and being capable of reducing almost 50% of parameters. Moreover, it shows reductions of 2.9% on model latency despite being this the main goal used in its NAS method.

We note that both MobileNetV2 and MnasNet perform best in tiny-Imagenet dataset. We believe this is because tiny-Imagenet is strongly related to Imagenet, and designs have been optimised to the latter. However, template **a** is still competitive with reductions of 1.5 and 1.8 points in accuracy but savings up to 36% and 49% in parameters for MobileNet and MnasNet, respectively, on this specific dataset. We show plots for these results in Figure 4 of the appendix.

5. Templates on ImageNet with Classical Models

As expected, results on the ImageNet dataset show the pyramidal pattern yields better accuracy in models than templates (see Table 4). Yet, templates produce models with lower number of parameters. For example the model obtained from VGG with template **d** uses only 29% of the original parameters sacrificing 4.6 points in accuracy. For ResNet, template **d** can produce a model with the accuracy of the original VGG using 35% of the original ResNet parameters. With template **a**, we obtained a model with 38% less parameters and a difference of less than one point of accuracy.

For VGG and ResNet, our experiments show that the pyramidal filter distributions of both models (and possibly many other CNNs) are overfitting ImageNet. For other datasets, a different distribution could make models better in accuracy and/or more efficient. Benefits of exploring new distribution are applicable to NAS methods too. We provide and example and extend this discussion in section 7.

6. Templates on Audio Classification

According to [39], in the absence of a well-established theory to find the optimal design hyperparameters for a CNN architecture for a specific task (size of kernels, pooling, number of channels and interconnections with successive layers), researchers have mostly opted to experimentally select the best performing model from a range of, usually alike, alternatives. While this statement is made for the field of audio processing, we agree that it is true for many of the other domains of deep learning architecture design.

We have found in audio processing architectures, as well in computer vision, that many complex models are not developed from scratch. They use classic architectures as a backbone, being ResNet one of the most popular [20]. So, we decided to test out templates using the well known ResNet50 architecture, adapting only the number of filters according to the new definition of templates and the final dense layer to adjust the output as required for the dataset to be evaluated.

Raw audio samples come from a one-dimensional signal indexed in time [39]. Nevertheless, they are often translated into two-dimensional time-frequency representations, such as mel-frequency cepstral coefficients (MFCCs), which is the standard representation used for audio data processing [17, 36, 57]. MFCC spectrograms, unlike images in computer vision applications, do not represent an instant in time. Instead, they are built taking constant-length segments from the raw audio signal. Yet, the resulting representation can be interpreted as a single image and processed using classic convolutional neural networks [21].

6.1. Audio Datasets

The GTZAN dataset [50] is used for for music genre recognition (MGR) [47]. GTZAN contains 1000 music clips with a duration of 30 seconds each. The clips, sampled at a rate of 22.5kHz, are grouped into 10 distinct genre classes. The

Table 2. VGG19 and ResNet50 performances with the original distribution of filters and five templates evaluated on six datasets. Flops are kept to similar values between templates of same models (399 MFLOPs for VGG19 and 1307 MFLOPs for ResNet50). After filter redistribution, most models surpass the base accuracy with less resources. Results show average of three repetitions.

Template	mnist	fashionmnist	cifar10	cifar100	cinic10	tiny imagenet
vgg19 base	99.769 ± 0.011	95.47 ± 0.05	94.90 ± 0.10	73.91 ± 0.08	85.79 ± 0.10	57.34 ± 0.30
vgg19 a	99.772 ± 0.019	95.59 ± 0.16	95.03 ± 0.26	74.47 ± 0.10	86.13 ± 0.14	57.21 ± 0.56
vgg19 b	99.779 ± 0.005	95.51 ± 0.03	95.01 ± 0.10	73.34 ± 0.24	86.37 ± 0.02	56.88 ± 0.31
vgg19 c	99.775 ± 0.024	95.28 ± 0.06	95.04 ± 0.19	72.27 ± 0.35	86.15 ± 0.09	54.50 ± 0.24
vgg19 d	99.779 ± 0.040	**95.65 ± 0.09**	**95.21 ± 0.08**	**74.64 ± 0.13**	**86.49 ± 0.05**	**59.45 ± 0.18**
vgg19 e	**99.789 ± 0.024**	95.33 ± 0.17	94.75 ± 0.07	71.13 ± 0.28	85.85 ± 0.03	54.26 ± 0.35
resnet50 base	99.772 ± 0.009	95.58 ± 0.10	95.91 ± 0.29	78.31 ± 0.54	88.78 ± 0.93	65.57 ± 0.47
resnet50 a	99.766 ± 0.022	**95.66 ± 0.16**	96.10± 0.07	79.00 ± 0.05	**89.60 ± 0.05**	**66.06 ± 0.53**
resnet50 b	99.762 ± 0.019	95.48 ± 0.13	96.07 ± 0.08	78.91 ± 0.08	89.36 ± 0.09	65.01 ± 0.43
resnet50 c	**99.779 ± 0.037**	95.41 ± 0.08	96.13 ± 0.20	77.92 ± 0.18	89.27 ± 0.15	64.07 ± 0.17
resnet50 d	99.775 ± 0.022	95.61 ± 0.08	**96.20 ± 0.11**	**79.43 ± 0.24**	89.30 ± 0.29	65.59 ± 0.39
resnet50 e	99.749 ± 0.030	95.41 ± 0.07	95.79 ± 0.03	77.99 ± 0.48	89.15 ± 0.02	64.49 ± 0.57

Table 3. MobileNetV2 and MnasNet performances with the original distribution of filters and five templates evaluated on six datasets. Flops are kept to similar values between templates of same models (68 MFLOPs for MobileNetV2 and 314 MFLOPs for MnasNet on CIFAR10). Despite both original architectures have been highly optimised, most resulting models from applying templates surpass the base accuracy. Results show average of three repetitions.

Template	mnist	fashionmnist	cifar10	cifar100	cinic10	tiny imagenet
mobilenetV2 base	99.726 ± 0.024	94.38 ± 0.04	93.37 ± 0.10	72.62 ± 0.03	82.19 ± 0.04	**61.47 ± 0.39**
mobilenetV2 a	**99.772 ± 0.019**	94.69 ± 0.14	93.69 ± 0.20	**73.31 ± 0.39**	82.83 ± 0.04	58.98 ± 2.11
mobilenetV2 b	99.752 ± 0.019	**94.85 ± 0.11**	**94.26 ± 0.15**	73.27 ± 0.31	**83.85 ± 0.17**	55.40 ± 2.62
mobilenetV2 c	**99.772 ± 0.026**	94.72 ± 0.11	93.54 ± 0.20	67.96 ± 0.40	83.03 ± 0.10	41.73 ± 5.04
mobilenetV2 d	99.752 ± 0.009	94.57 ± 0.04	93.57 ± 0.11	68.92 ± 0.11	82.33 ± 0.21	45.02 ± 5.75
mobilenetV2 e	99.756 ± 0.020	94.66 ± 0.22	93.82 ± 0.18	70.18 ± 0.31	83.20 ± 0.10	52.61 ± 1.86
mnasnet base	99.736 ± 0.005	95.17 ± 0.07	94.92 ± 0.07	76.46 ± 0.30	86.25 ± 0.07	**61.78 ± 0.27**
mnasnet a	99.756 ± 0.015	**95.52 ± 0.07**	**95.62 ± 0.17**	**76.73 ± 0.48**	**87.28 ± 0.09**	59.02 ± 0.62
mnasnet b	**99.772 ± 0.029**	95.34 ± 0.10	95.49 ± 0.10	75.82 ± 0.26	86.94 ± 0.11	56.51 ± 0.31
mnasnet c	99.752 ± 0.017	95.50 ± 0.11	95.23 ± 0.12	74.01 ± 0.50	86.65 ± 0.14	49.12 ± 0.11
mnasnet d	99.746 ± 0.011	95.47 ± 0.06	95.53 ± 0.11	74.96 ± 0.25	86.93 ± 0.12	51.62 ± 0.61
mnasnet e	99.759 ± 0.011	95.48 ± 0.08	95.41 ± 0.08	75.55 ± 0.26	86.88 ± 0.16	57.62 ± 0.29

ESC-50 dataset is designed to provide a benchmark for environmental sound classification [38] (laughter, cat meowing, glass breaking or brushing teeth are some examples of environmental sounds), a distinct task from speech or music classification. The ESC-50 dataset consists of 2000 labelled environmental recordings distributed between 50 classes. Each instance has a length of 5 seconds sampled at 44.1 kHz.

6.2. Implementation Details

For training, we use the code and hyperparameters provided by [37], which were found via grid search [34]. Learning rate and weight decay were set a 0.0001 and 0.001 respectively. We use a batch size of 32 for ESC-50 and 16 for GTZAN. The learning rate was decreased by a factor of 10 for every 30 epochs from a total of 70 epochs.

6.3. Results

The results of this experiment are shown in Table 5. They show the average accuracy of three runs. By using templates on ResNet50 we can see marginal improvements in accuracy of 0.96% and 2.14% for GTZAN and ESC-50 over the base ResNet model compared to the best performing template. However, when we look at the resource consumption, savings in memory footprint and inference time reach 15% for template d while the number of parameters shows a considerable reduction of 65%. We could take template c on GTZAN and obtain a similar accuracy with only 14.82% of the original parameters. As in all the experiments performed in this paper, the different templates use similar FLOPs to the original ResNet architecture. In this way, we show that there are no hidden costs of applying our templates other than the simple step of redistributing neurons and training our small set.

There is no absolute winner template in this task of audio classification. Instead, each particular template provides a compromise of advantages even between datasets.

Table 4. VGG19 and ResNet50 performances with the original distribution of filters and five templates evaluated on ImageNet. Despite the pyramidal base pattern reaches the highest accuracy in both models, some other filter distributions show considerable reductions in parameters.

			Templates				
Models		base	a	b	c	d	e
VGG19	Acc	72.936	72.804	66.406	62.198	68.248	60.874
(19.6 GFLOPs)	Param	143.672	155.523	54.759	35.618	41.798	58.070
ResNet50	Acc	75.309	74.603	70.667	68.425	72.823	69.160
(4.1 GFLOPs)	Param	25.557	15.847	5.405	3.844	8.998	4.214

Table 5. Accuracy and resource utilisation of ResNet50 with templates on GTZAN and ESC-50 audio classification datasets.

		Filter Templates				
Metric	base	a	b	c	d	e
Accuracy GTZAN	85.59	85.92	87.26	85.75	**87.43**	85.08
Accuracy ESC-50	69.66	**70.33**	68.16	65.75	69.91	67.25
Param (Millions)	23.61	14.23	4.88	**3.50**	8.39	3.71
Memory (MB)	395.51	350.54	383.10	385.15	**337.40**	392.29
Inference (ms)	5.47	7.47	4.56	7.75	4.66	**4.48**

7. Templates on NASBench-101

NASBench-101 [53] was introduced to provide a common framework to evaluate new proposed exploration and performance estimation strategies for NAS methods. The dataset delivers training and validation performances of all convolutional neural network architectures on the CIFAR-10 dataset. All networks are built by stacking identical groups of layers called cells which are followed by a downsampling layer. The network finish with a dense layer that conducts the final classification. In this sense, the difference between networks is found in the cell design. Cells are described by directed acyclic graphs with up to 9 vertices and 7 edges. The set of valid operations at each vertex are 3x3 convolution, 1x1 convolution, and 3x3 max-pooling.

The search space of NASBench-101 contains 423,624 individual CNN networks, each of them being trained and evaluated several times on CIFAR-10. Given that networks are evaluated at several steps, the dataset contains over 5 million trained models.

7.1. Implementation Details

NASBench-101 is a benchmark framework that provides a full set of tools programmed in TensorFlow [1] to evaluate neural networks following the pattern defined in the search space. The search space of NASBench is restricted to the pyramidal distribution of filters. Hence we performed minor changes to the original code to add the different templates.

For all NASBench models, the authors used the same set of hyperparameters. By running a coarse grid search on the average accuracy of 50 randomly sampled designs from the space, this collection of hyperparameters was chosen to be robust across different architectures. We use the same training parameters defined in the framework.

7.2. Results

On NASBench, we evaluated and compared the best model in the dataset and another ResNet-like network highlighted in [53].

Models in the NASBench-101 dataset are not constrained in resources in any way. Restrictions are created indirectly by the types of layers and connections, the number of cells in each module and the number of modules. Because our method follows the same graph as the original architectures, we constrain the models using templates to operate under the same amount of FLOPs to have a similar point of reference.

The experimental results are shown in Table 6. We have mentioned that a fair comparison of models is challenging not only with models inside the dataset but also with models in the literature in general. FLOPs and parameters of the best performing network in NASBench-101 are more than five times the ones of the ResNet-like network, while gaining in accuracy is less than three per cent.

Models obtained with templates show a considerable reduction of computational costs. Template *b* with both models uses one-fifth of the original parameters. Template *c* obtains a higher accuracy than the best performing model with one-third of the parameters. We are not aiming for templates to outperform any NAS method. We state that combining templates with NAS methods can deliver further improvements to final CNN models at a very low cost. Moreover, by exploring the proposed (and other) new distributions, it is possible to find more efficient models. Each template enables different metrics to be enhanced.

8. Discussion and Conclusions

Since the early origins of CNN models, starting with the Neocognitron, the distribution of filters began with a uniform pattern but switched to a pyramidal pattern since the

Table 6. accuracy and parameters of the best model in NASBench-101 dataset and a ResNet-like model produced with an extended search space. By using templates, both models are capable of obtaining further accuracy with fewer parameters using the same FLOPs.

			Templates				
Models		base	a	b	c	d	e
Best architecture	Acc	95.35	95.20	95.02	**95.44**	95.06	95.26
3664 MFLOPs	Param	32.42	27.49	**6.44**	10.38	17.26	8.73
ResNet-like	Acc	92.64	**93.85**	91.80	92.65	91.81	92.67
687 MFLOPs	Param	6.04	5.18	**1.24**	1.79	3.30	1.63

LeNet introduction in 1989. From then, the pyramidal design has remained largely unquestioned in almost all neural networks including classical and resource-optimised ones. Methods that modify the number of filters, such as pruning, neural architecture search, and channel number search, also initiate their exploration from models following the pyramidal design. We argue that researchers keep using the pyramidal design because models are mainly tested on the ImageNet dataset. Therefore, the hyperparameters defining their structure (including filter distribution) are optimised to ImageNet.

This work introduced a definition of templates that allows matching a predefined number of FLOPs with no significant overheating in the search process. A set of experiments with filter distribution templates were performed in several domains to evaluate their representation capability.

Overall, redistributing templates enhances performance and reduces resources for the models and domains tested. Exploring novel filter distributions has advantages that go beyond the domain of image classification. Consequently, the suggested templates offer a straightforward mechanism for quickly achieving performance gains compared to the computationally expensive NAS approaches.

Despite significant changes in filter distributions from the original architectures, the variation in accuracy for all models after using templates is less than 5% for image classification. These results defy the common wisdom that CNN models are required to capture more diverse features in deeper layers and show that lower-dimensional representations are still useful in deeper layers. Moreover, lowering filters can be beneficial for some datasets.

Experiments indicate that for each model tested, there is no particular distribution of filters that guarantees the best accuracy on all tasks. Furthermore, templates can improve differently on the same task but different datasets. This means that the results of automatically searching for the number of channels in small datasets such as CIFAR should be carefully extrapolated to others. In the opposite direction, models with distributions that work well on extensive datasets should be changed (e.g., using templates) to perform efficiently on different domains.

The approach presented in this work allows a model's architect to apply a set of templates for changing the number of filters originally assigned to each layer before training from scratch. This redesign can be easily achieved without any previous training process to select particular weights. In essence, the application of filter distribution templates offers an alternative approach to the iteration-intensive automatic architecture search and model pruning methods.

It is clear there are contributions to be made in terms of questioning if the architectures that have been designed for ImageNet are applicable everywhere. The existence of dataset-dependent architecture requires faster ways to find networks performing well in each case. Authors of new architectures such as the isometric neural networks, the vision transformer, and the convolutional mixer have looked back and adopted the uniform distribution again. We hope this work motivates to continue the search for other filter distributions that benefit the evolution and better understanding of more efficient models.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016. 7

[2] Neena Aloysius and M Geetha. A review on deep convolutional neural networks. In *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 0588–0592. IEEE, 2017. 2

[3] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021. 2

[4] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, 2012. 1

[5] Maxim Berman, Leonid Pishchulin, Ning Xu, Matthew B Blaschko, and Gérard Medioni. Aows: Adaptive and optimal network width search with la-

tency constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11217–11226, 2020. 2

[6] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6: 64270–64277, 2018. 3

[7] Joseph Lin Chu and Adam Krzyżak. Analysis of feature maps selection in supervised learning using convolutional neural networks. In *Canadian Conference on Artificial Intelligence*, pages 59–70. Springer, 2014. 1

[8] Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6659–6669, 2021. 2

[9] Musab Coşkun, Özal YILDIRIM, UÇAR Ayşegül, and Yakup Demir. An overview of popular deep learning methods. *European Journal of Technique*, 7(2):165–176, 2017. 4

[10] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018. 4

[11] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 4

[12] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *arXiv preprint arXiv:1905.09717*, 2019. 2

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 2

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 2

[15] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10628–10637, 2020. 2

[16] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 1, 2

[17] Todor Ganchev, Nikos Fakotakis, and George Kokkinakis. Comparative evaluation of various mfcc implementations on the speaker verification task. In *Proceedings of the SPECOM*, pages 191–194, 2005. 5

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 1

[19] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018. 2

[20] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Esresnet: Environmental sound classification based on visual domain models. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4933–4940. IEEE, 2021. 5

[21] Grzegorz Gwardys and Daniel Michał Grzywczak. Deep image features in music information retrieval. *International Journal of Electronics and Telecommunications*, 60(4):321–326, 2014. 5

[22] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5927–5935, 2017. 2

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 4

[24] Min-Fong Hong, Hao-Yun Chen, Min-Hung Chen, Yu-Syuan Xu, Hsien-Kai Kuo, Yi-Min Tsai, Hung-Jen Chen, and Kevin Jou. Network space search for pareto-efficient spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3053–3062, 2021. 2

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[26] Ramon Izquierdo-Cordova and Walterio Mayol-Cuevas. Towards efficient convolutional network models with filter distribution templates. *arXiv preprint arXiv:2104.08446*, 2021. 2

[27] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, 2020. 2

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 4

[29] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015. 4

[30] Guillaume Leclerc, Manasi Vartak, Raul Castro Fernandez, Tim Kraska, and Samuel Madden. Smallify:

Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018. 2

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998. 1, 2

[32] Eugene Lee and Chen-Yi Lee. Neuralscale: Efficient scaling of neurons for resource-constrained deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1478–1487, 2020. 2

[33] Yandong Li, ZB Hao, and Hang Lei. Survey of convolutional neural network. *Journal of Computer Applications*, 36(9):2508–2515, 2016. 2

[34] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018. 6

[35] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 2

[36] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*. Citeseer, 2000. 5

[37] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. Rethinking cnn models for audio classification. *arXiv preprint arXiv:2007.11154*, 2020. 6

[38] Karol J Piczak. Esc: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1015–1018, 2015. 6

[39] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019. 5

[40] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017. 2

[41] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2, 4

[42] Madhusmita Sahu and Rasmita Dash. A survey on deep learning: Convolution neural network (cnn). In *Intelligent and Cloud Computing*, pages 317–325. Springer, 2021. 2

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 4

[44] Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 1, 2

[45] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7: 53040–53065, 2019. 2

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 4

[47] Bob L Sturm. A survey of evaluation in music genre recognition. In *International Workshop on Adaptive Multimedia Retrieval*, pages 29–66. Springer, 2012. 5

[48] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 2, 4

[49] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022. 1, 2

[50] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002. 5

[51] Ihsan Ullah and Alfredo Petrosino. About pyramid structure in convolutional neural networks. In *2016 International joint conference on neural networks (IJCNN)*, pages 1318–1324. IEEE, 2016. 2

[52] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, and Jian Cheng. Revisiting parameter sharing for automatic neural channel number search. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[53] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019. 7

[54] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019. 2

[55] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 2

[56] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 2

[57] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of mfcc. *Journal of Computer science and Technology*, 16(6): 582–589, 2001. 5

[58] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2017. 2

[59] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. 1

The Myth of the Pyramid

Supplementary Material

A. Filters by Layer for Models using Templates

In order to facilitate the reproducibility of experiments, we present in this section the values for each layer obtained by applying templates to the four models tested in our work in Section 3. In the case of ResNet we set the value of filters at the level of each layer inside residual modules, as presented in Table 10. VGG design consists of simple layers, so we change filters in each of them (Table 7). For the rest of architectures, we set the value of filters at the level of modules (Tables 8 and 9). The last layers of all models are fully connected, and the dataset imposes the number of neurons. We add the schematics of the templates as a visual aid in figure 2.

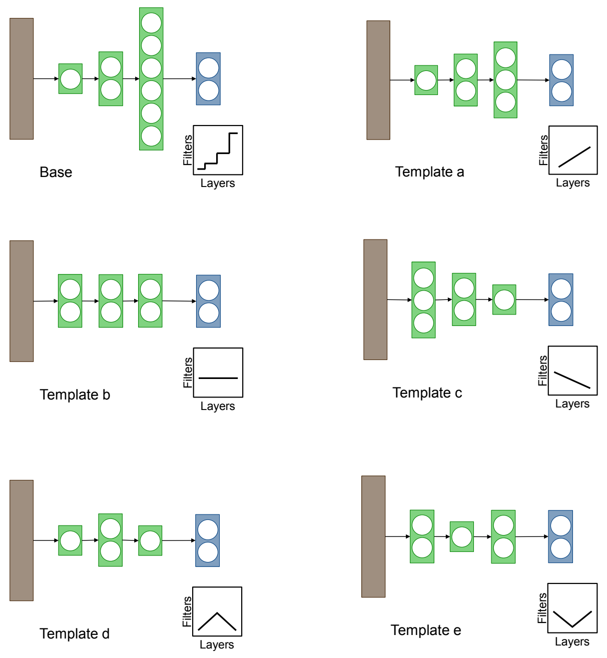

Figure 2. Schematic distribution of filters per layer in templates. Base distribution is the original pyramidal distribution. Templates follow a smoother transition in the number of filters between layers. Number of filters does not match between models but they are adjusted to fit the original number of FLOPS of the base model.

Table 7. VGG19 with the original distribution of filters and five templates. All models count similar number of FLOPs.

Template	Filter Values
Base (Original values)	64, 64, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512, 512, 512, 512, 512
a	64, 99, 133, 168, 203, 238, 272, 307, 342, 377, 411, 446, 481, 516, 550, 585
b	153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153, 153
c	165, 158, 152, 145, 138, 131, 125, 118, 111, 104, 98, 91, 84, 77, 71, 64
d	64, 105, 146, 186, 227, 268, 308, 349, 390, 343, 297, 250, 204, 157, 111, 64
e	175, 161, 147, 133, 120, 106, 92, 78, 64, 80, 96, 112, 127, 143, 159, 175

Table 8. Original distribution of filters for MobileNet2 after applying five templates.

Template	Filter Values
Base (Original values)	16, 24, 32, 64, 96, 160, 320, 1280
a	16, 36, 56, 76, 97, 117, 137, 157
b	61, 61, 61, 61, 61, 61, 61, 61
c	77, 68, 60, 51, 42, 33, 25, 16
d	16, 38, 59, 80, 102, 73, 45, 16
e	92, 73, 54, 35, 16, 41, 67, 92

Table 9. Distribution of filters for MNASNet showing the original design and filters from five templates.

Template	Filter Values
Base (Original values)	32, 16, 24, 40, 80, 96, 192, 320, 1280
a	32, 46, 60, 73, 87, 101, 114, 128, 142
b	76, 76, 76, 76, 76, 76, 76, 76, 76
c	102, 93, 84, 76, 67, 58, 50, 41, 32
d	32, 49, 66, 84, 101, 84, 66, 49, 32
e	141, 114, 86, 59, 32, 59, 86, 114, 141

Table 10. Original distribution of filters for ResNet50 and five templates. All models count similar number of FLOPs. Filter redistribution is made at the lever of layers within modules. Expansion layers within modules in the same block are kept with equal filters to fit residual connections.

Template	Filter Values
Base (Original values)	64, [[64,64,256], [64,64,256], [64,64,256]], [[128,128,512], [128,128,512], [128,128,512], [128,128,512]], [[256,256,1024], [256,256,1024], [256,256,1024], [256,256,1024], [256,256,1024], [256,256,1024]], [[512,512,2048], [512,512,2048], [512,512,2048]]
a	64, [[64,73,256], [83,92,256], [102,111,256]], [[120,130,480], [139,148,480], [158,167,480], [177,186,480]], [[195,205,780], [214,224,780], [233,242,780], [252,261,780], [271,280,780], [289,299,780]], [[308,317,1232], [327,336,1232], [346,355,1232]]
b	64, [[123,123,492], [123,123,492], [123,123,492]], [[123,123,492], [123,123,492], [123,123,492], [123,123,492]], [[123,123,492], [123,123,492], [123,123,492], [123,123,492], [123,123,492], [123,123,492]], [[123,123,492], [123,123,492], [123,123,492]]
c	64, [[134,132,536], [129,127,536], [125,123,536]], [[120,118,480], [116,114,480], [111,109,480], [107,105,480]], [[102,100,408], [98,96,408], [93,91,408], [89,87,408], [84,82,408], [80,78,408]], [[75,73,300], [71,69,300], [66,64,300]]
d	64, [[64,76,256], [88,99,256], [111,123,256]], [[134,146,536], [158,170,536], [182,193,536], [205,217,536]], [[228,240,912], [252,239,912], [227,214,912], [202,189,912], [177,164,912], [152,139,912]], [[127,114,508], [102,89,508], [77,64,508]]
e	64, [[144,139,576], [134,129,576], [124,119,576]], [[114,109,456], [104,99,456], [94,89,456], [84,79,456]], [[74,69,296], [64,69,296], [75,80,296], [85,91,296], [96,101,296], [107,112,296]], [[117,123,468], [128,133,468], [139,144,468]]

B. Accuracy of Models After Applying Templates

This appendix presents plots for the experimental results related to Section 4 including four CNN architectures (VGG, ResNet, MobileNetV2 and MnasNet) evaluated on six different datasets. Each architecture's original filter distribution was transformed using the proposed templates. Final models obtained from a particular architecture count similar FLOPs for making fair comparisons. Once the models are changed, they are trained from scratch. We observe that a different template provides the best improvement for each pair model-task. The difference between templates is more notorious when comparing models belonging to different categories (classical or highly-optimised).

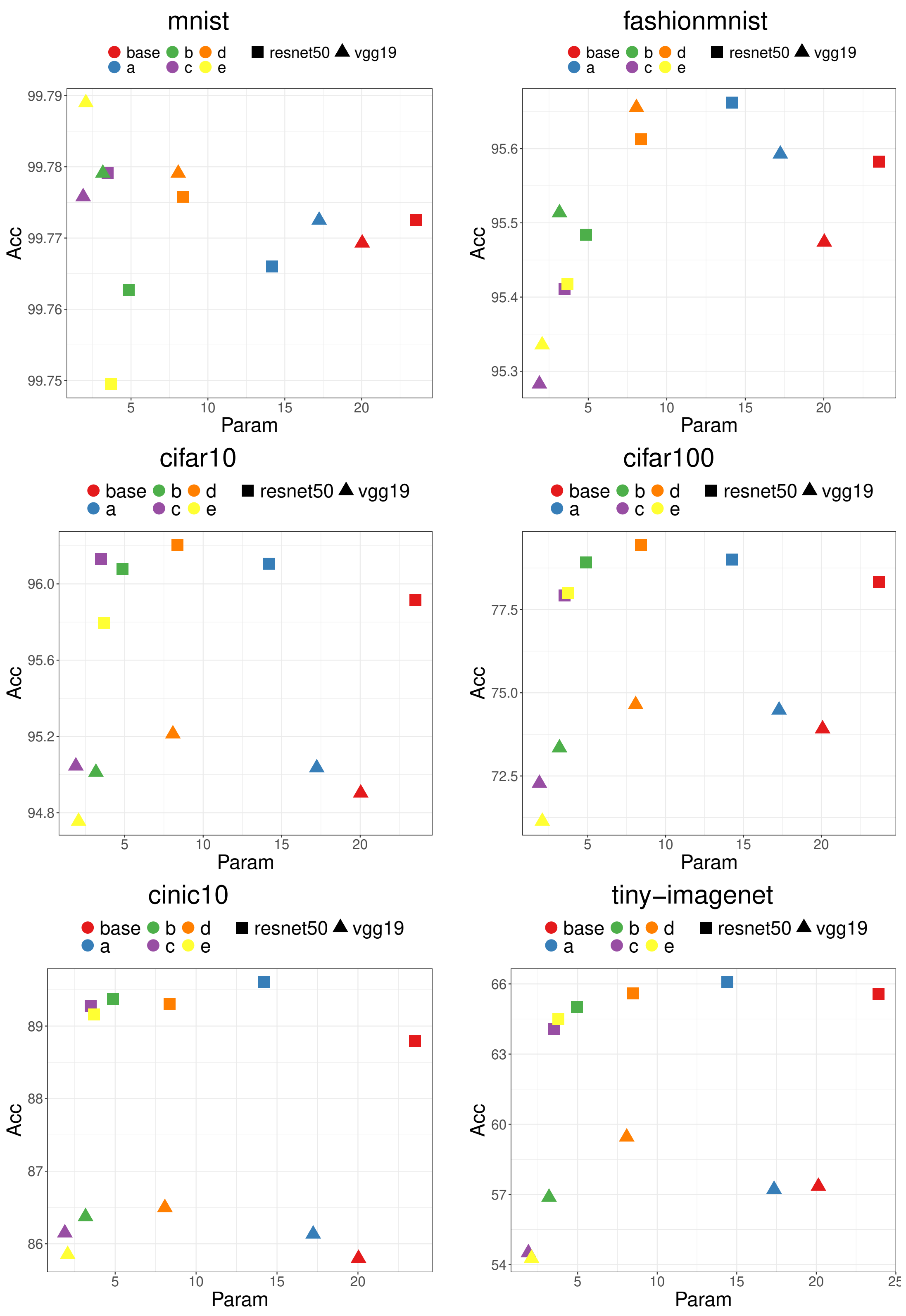

Figure 3. Accuracy of VGG and ResNet models after applying templates reported for several datasets. Base is the original distribution of filters. In many cases, templates outperform the base architecture. However, all of them use much fewer parameters than the base model. Note that models produced with templates from VGG have less than a third of FLOPs of those produced from ResNet.

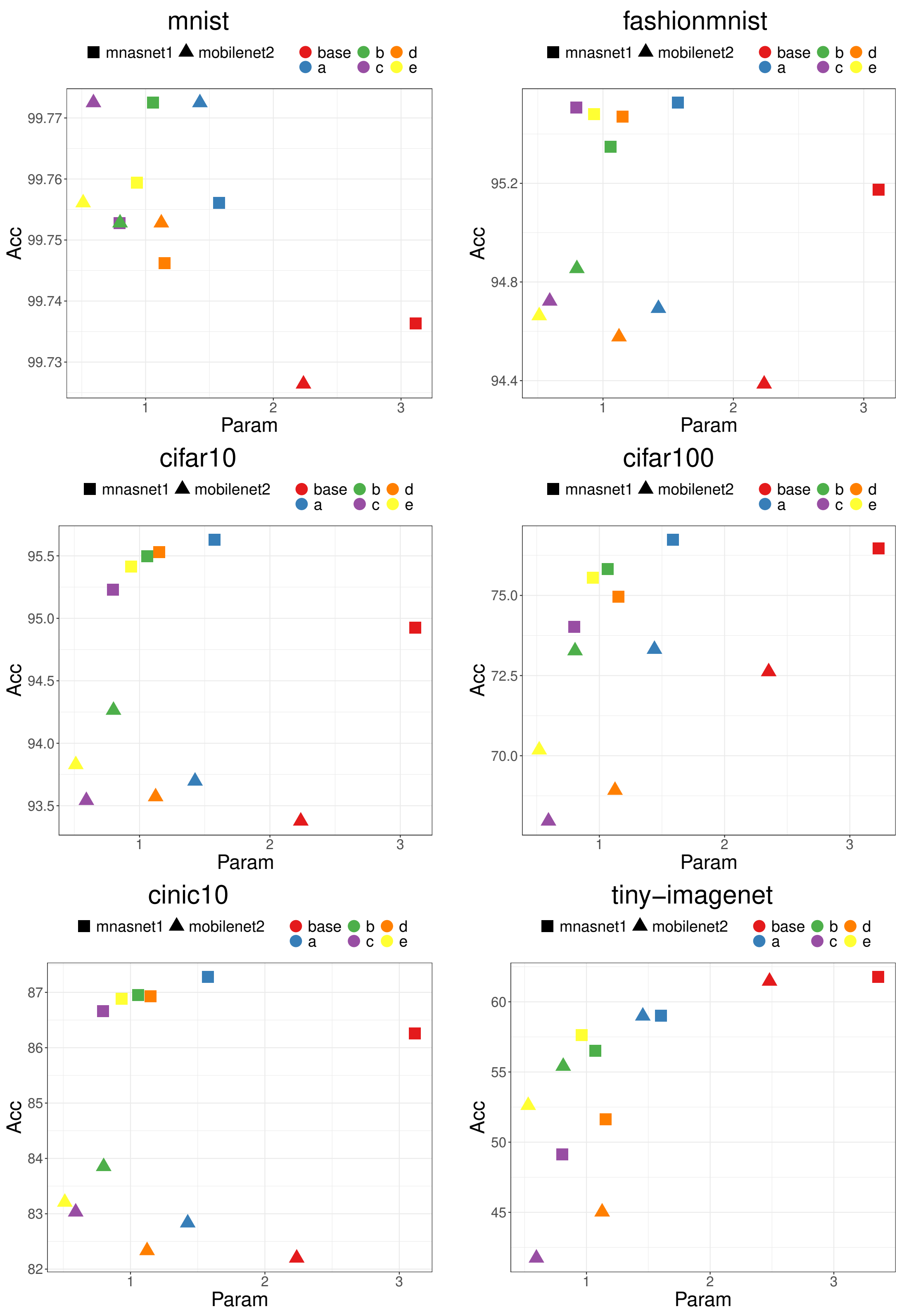

Figure 4. Parameter efficiency of MobileNetV2 and MnasNet models with templates reported for several datasets. Base is the original distribution of filters. In many cases, templates outperform the base architecture. However, all of them use much fewer parameters than the base model. Note that models produced with templates from MnasNet have more than 4X FLOPs of those produced from MobileNetV2.