Author:
Izquierdo Cordova, Ramon

Title:
Towards more efficient CNN models with filter distribution templates

Towards More Efficient CNN Models with Filter Distribution Templates

By

RAMON IZQUIERDO CORDOVA

Department of Computer Science
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

APRIL 2022

Word count: forty-five thousand, one hundred forty-four

Abstract

Researchers in the field of Deep Learning have used convolutional neural network architectures to solve computer vision tasks claiming improved performance with reduced computational cost. Designing these networks remains a complex task, relying primarily on human experience. These new architectures have been created with important innovative elements (ReLu, dropout and batch-normalisation layers) and new structures (residual connections, grouped convolutions). However, a particular feature has remained unchanged: the pyramidal pattern for distributing the number of filters in each layer, increasing filters when feature map resolutions decrease. Initially proposed in 1989 in the LeNet architecture, this pyramidal design is found in classical and state-of-the-art CNN architectures. Also, it is the starting point for the exploration space of pruning methods and automatic model search. The reason behind this incremental design relies on the roots of the deep learning definition aiming to learn hierarchical levels of representation, making complex concepts in higher levels by reusing simpler low-level ones. Because many convolutional networks are tested mainly on the ImageNet dataset, and the pyramidal distribution is tuned to fit that dataset, it is unclear if the pattern works well in other datasets and domains or if other distributions could yield better performances.

This thesis introduces the concept of filter distribution templates, a small set of filter distribution patterns differing from the widely adopted pyramidal distribution for reassigning filters in an existing convolutional network without varying the original architecture. This research experimentally shows that models produced with templates are superior to those using the pyramidal distribution of filters in several popular datasets from the domains of image classification and camera pose estimation. Additionally, this work describes and evaluates how templates can work on top of convolutional network compression techniques to obtain higher accuracy or reduction ratio.

Further, this thesis proposes an enhanced set of templates that improve final models' accuracy by smoothing the variation in the number of filters between layers. The new set is provided with a fast mechanism to match a predefined FLOPs budget. Extended experiments on image and audio classification show that models obtained with the new templates yield higher performances with fewer resources. For example, experiments with three popular handcrafted architectures (VGG, ResNet and MobileNetV2) and one automatically discovered (MNASNet) trained on MNIST, CIFAR, CINIC10 and

TinyImagenet datasets show that models with these alternative distributions are more resource-efficient reaching reductions up to 90% in parameters and 79% in memory needs while matching or surpassing the original model accuracy.

Advantages in performance and resource requirements were also found in a representation embeddings task, a domain where the network's internal representation is more important than its final output. Templates were again helpful to architectures discovered with neural network search methods surpassing the highest accuracy model in the NASBench-101 dataset requiring one-third of the original parameters. Finally, this theis explored the variations in representation spaces produced by templates. Attempts were conducted to correlate accuracy and representation spaces using CKA similarity. However, results imply that finding the best template is challenging and requires more exploration and analysis.

We hope this thesis illuminates new directions to neural network designers, both automated and manual, that help construct more efficient architectures and inspire the re-think of model building assumptions in deep learning.

DEDICATION AND ACKNOWLEDGEMENTS

I would like to thank my supervisor Walterio Mayol-Cuevas for his patience, dedication and guidance during the development of my doctoral studies. Certainly, He supported me in much more than academic issues.

Thanks to each one in the VILab who share a chat while enjoying a cup of coffee (or cider): Abel, Angeliki, Benjamin, Bridget, Davide, Eduardo, Faegheh, Farnoosh, Hazel, Igor, Janis, Jonny, Laurie, Miguel, Mike, Obed, Perla, Sam, Sasha, Tenshi, Toby, Vangelis, Will, Will, Xingrui, Yanan, Yao, Young, Yuhang and Zeynel. I immensely enjoyed listening to all sort of enriching topics.

Thanks to Osamah and Laila, Luis and Martha, Matt and Ksenia, Gustavo and Estefania, Alexander and Dayana, Benjamin and Martha, Marek and Carmen, Nordine and Winona, David and Bee, Simon and Fabiola, Julia, Barbara, and to all the people who kindly welcomed us, opened their homes to us and showed us the life in each of their cultures. That experience changed us.

Thanks to all my in-law relatives, Omega, Deyoses, Claudia and Tello, Gisela and David, Adrian and Fabiola, for being in charge of Rocio's concerns and needs. They brought us peace of mind when we needed it.

I want to thank my parents for being examples of hardworking and for their tolerance of my long absences. Thanks to my siblings for their encouragement to take this PhD. Particular gratitude to Chela, Chevo and Perla for their support with the many tasks I had to leave during my studies.

Infinite thanks to my precious group of R's (Rocio, Rocio and Romilly). You continuously pushed me to finish this thesis. You really helped. I hope I hadn't stolen too much time from you and that the memories we created during our stay in Bristol last forever.

I want to thank the National Council of Science and Technology (CONACYT México). Its scholarship programs allow many students to access quality postgraduate courses worldwide.

Finally, I greatly thank and recognise the People of Mexico as the true investor in forming human scientific capital in my country. I feel committed to giving back the great support I have received.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

Acronyms

2D	Two-dimensional.
ANN	Artificial Neural Networks.
BN	Batch Normalisation.
CCA	Canonical Correlation Analysis.
CKA	Centered Kernel Alignment.
CNN	Convolutional Neural Network.
CNS	Channel Number Search.
CUDA	Compute Unified Device Architecture.
DoF	Degree of Freedom.
FLOPS	Floating Point Operation Per Second.
FLOPs	Floating Point Operations.
GAN	Generative Adversarial Network.
GPU	Graphics Processing Unit.
HR	High-resolution.
HSIC	Hilbert-Schmidt Independence Criterion.
IB	Information Bottleneck.
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge.
INN	Isometric Neural Networks.
KD	Knowledge Distillation.
LR	Low-resolution.
MAE	Mean Absolute Error.

MFCC	Mel-Frequency Cepstral Coefficient.
MLP	Multi-layer Perceptron.
MSE	Mean Squared Error.
NAS	Neural Architecture Search.
NLP	Natural Language Processing.
OBD	Optimal Brain Damage.
PCA	Principal Component Analysis.
PSNR	Peak Signal-to-Noise Ratio.
RGB	Red, Green, Blue colour model.
RGB-D	Red, Green, Blue and Depth Sensor.
RL	Reinforcement Learning.
SfM	Structure from Motion.
SGD	Stochastic Gradient Descent.
t-SNE	t-Distributed Stochastic Neighbor Embedding.
UMAP	Uniform Manifold Approximation and Projection.

CHAPTER 1

INTRODUCTION

Modern artificial intelligence applications have become ubiquitous in our era. When they are correctly tuned, existing systems can surpass human-level performance in numerous specific tasks. This explosion of impressive achievements has been primarily obtained thanks to the use of deep learning.

The core strength of deep learning comes from deep neural networks and their ability to learn relevant representations of the world. Initially inspired by modelling the neurons in a biological brain, neural networks are built by connecting artificial neurons in parallel to conform sequential layers. How these layers and neurons are organised defines the different deep learning network architectures.

In the deep learning field, a neural network design is currently commonly created to solve one particular problem. For example, the VGG network was created to achieve maximum accuracy on the ImageNet dataset in the image classification domain. Yet, it is also common to use the resulting architecture to solve other tasks (in this work, we will use indistinctly the terms task and domain). We are not referring here to transfer learning, the process of taking a model trained in a dataset and then utilising it in a different one in the same or a distinct domain after a small retraining in the new dataset. Instead, we refers to the process of borrowing an existing untrained architecture and minimally changing it to fit the requirements of the new dataset to eventually train it from scratch in the new dataset. To illustrate the difference, let's take the case of ResNet. It was designed for ImageNet, but it has also been trained from scratch in the CIFAR-10 dataset and as a backbone of Faster R-CNN in PASCAL VOC and COCO datasets [73].

On the other hand, by fine-tuning a ResNet model trained on ImageNet, researchers have used it in a diverse range of tasks such as tracking objects in video surveillance [91], malicious software classification [166], and video object segmentation [28].

As the number of domains where deep learning is applied has increased, researchers have proposed a vast number of neural network architectures claiming enhanced performance and less computational resource consumption demands. However, conceiving these architectures remains a complex task, relying on experience and, in some cases, trial and error procedures. Currently, there is no full understanding of the rules that govern model design.

Despite the advances and widespread interest in neural networks, there is still a notable gap between theory and practice. While practitioners have made outstanding achievements, theorists have followed behind, with studies that usually include unrealistic assumptions that lead to inaccurate results in understanding deep neural networks as they are typically used.

Convolutional Neural Network (CNN), a particular deep learning architecture, has been notably successful in many vision tasks. However, after all the continuous progress in CNN models, an element in their design remains almost unchanged. There is a practice of increasing the number of filters in deeper layers, basically doubling the 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 [14, 63]. 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 [106], as well as improves performance by keeping a constant number of operations in each layer [30].

This pattern was first proposed in [106] with the introduction of LeNet and can be observed in a diverse set of models such as VGG[183], ResNet[73] and MobileNet[82]. Even models obtained from neural architecture search (NAS), such as NASNet [222], follow this principle since many automatic model discovery methods are mainly formulated to search for layers and connections. In contrast, the number of filters in each layer remains fixed.

Techniques used by NAS are now applied to channel number search (CNS) to find the optimal distribution of filters in a convolutional neural network [200] based on the intuition that the incremental design could not be the best option. Approaches perform a search process considering a set of promising values for the filter distribution that depart

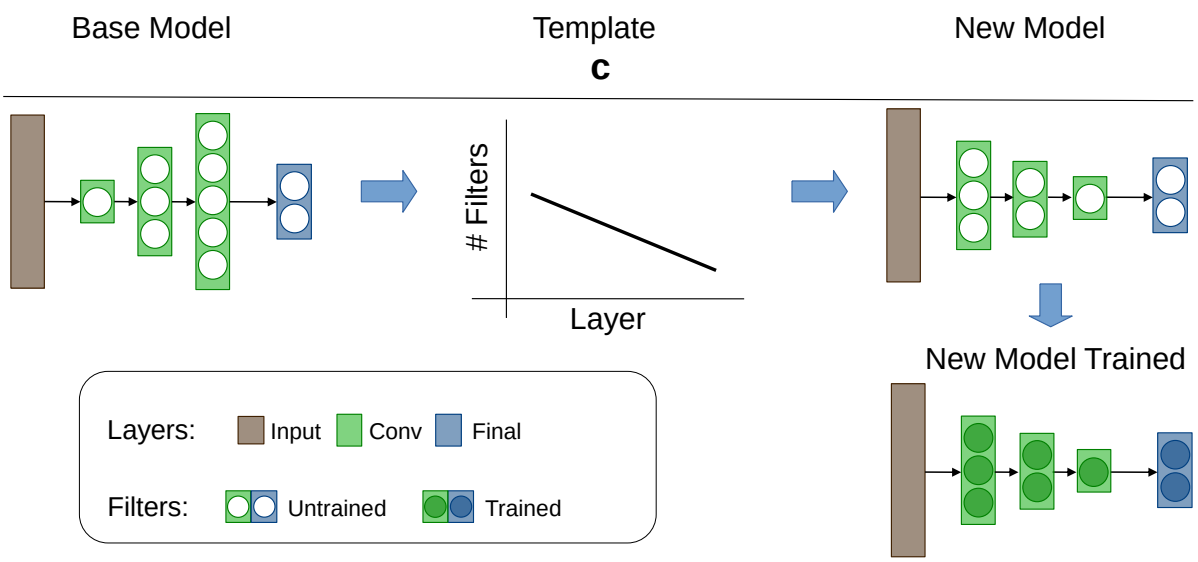


Figure 1.1: The pyramidal pattern of increasing filters per layer is widely adopted in convolutional models (left). We propose to apply a small set of templates to an existing model to change its filter distribution (center). After being trained from scratch (right), resulting models are competitive in accuracy compared to the original model, however, they require less computational resources.

from the original ones of a base pattern, generally the incremental one. Being based on NAS, the main limitation for widely using these CNS algorithms resides in the high computational cost implied in the exploration of the search space, which requires training and evaluating a vast number of candidate models before finding a good distribution of filters.

To overcome the limitation of CNS methods that incur high computational costs, this work is based on the idea that exploration should be performed in a few simple and radically different distributions to the pyramidal pattern instead of exploring individual widths in each layer. This work introduces a small set of these predefined distributions, called templates, intuitively created firstly by using highly diverse patterns defined by quadratic functions. Later, they are improved by using simple linear equations or combinations of them that can be easily implemented in most of the existing CNN classical and state-of-the-art models.

As depicted in Figure 1.1, to use a template, the method takes a base model and redefines its filter distribution with the pattern provided 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 floating point operations (FLOPs), produces

better or at least competitive accuracy, and consumes less of other resources. Unfortunately, this research has not found a way of uncovering the most appropriate template that could work with each task. Nonetheless, being the set of templates small enough, a simple sequential search is still fairly competitive in time compared to automatic methods. Moreover, experiments show an emerging pattern in which some templates present favourable features according to the requirements of the task to be performed.

Experimental evidence shows that simple changes to the pyramidal distribution of filters in some CNN models improve accuracy while reducing the number of parameters or memory footprint. Experiments also highlight that tested models, although significantly changed in their original filter design, present high resiliency in accuracy, a phenomenon that requires further research and explanation. If resilience is a general condition for all neural networks, the deep learning community will benefit in practicability, allowing practitioners to freely choose an appropriate number of channels according to their priorities in resource consumption without sacrificing the performance of the network significantly.

1.1 Contributions

The conventional wisdom in deep learning is that increasing the filters in deeper layers of neural networks increases the diversity of high-level attributes leading to a better generalisation [70, 106].

This thesis offers the following main contributions:

- Chapter 3 challenges the widely adopted idea of incrementally allocating filters in a convolutional neural network by adopting a small set of entirely different patterns called templates for redistributing filters without varying the rest of the original architecture.

- Chapter 4 explores the scope to which the use of templates can be beneficial to convolutional neural networks and compares resulting models from other filter number manipulation techniques such as compression methods.

- Chapter 5 proposes a linear-segment definition based on additive filter changes from the PyramidNet design to develop a new set of improved templates that matches a FLOPs budget.

1.2 Thesis Outline

Chapter 2 examines the development of convolutional neural networks from their first conception, including models that won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [170]. This chapter outlines the primary improvements of listed models and shows that the ImageNet dataset has influenced architectures to follow a pyramidal filter distribution with no other argument than empirical results of improved accuracy. Other strategies for optimising models that impact the distribution of filters, such as pruning, distillation and neural architecture search, are also discussed.

Chapter 3 introduces the concept of templates to examine whether the standard pyramidal configuration of filters in most known CNN models is beneficial to the task of image classification. Experiments in this chapter compare the accuracy of models by redistributing an equal number of filters in all templates without controlling the redistribution effects on resources. Additionally, the chapter presents a resource matching experiment showing the higher efficiency of templates in resource usage.

Chapter 4 explores the tasks of global localisation and super-resolution using templates with some classical CNN architectures to determine the scope in which a different distribution of filters produces enhanced models. The chapter compares the reduction effects of templates with three different pruning techniques and evaluated the templates functioning in a serial pipeline with MorphNet, a CNS method.

Chapter 5 redefines the set of templates adopting linear segments to form the patterns for the filter distributions. The new templates definition, inspired by the additive PyramidNet pattern, produces higher accuracies and allows to match a predefined budget of FLOPs. Templates are tested in an expanded group of domains where the performance of models depends more on the internal representation than the final outputs. Templates are also evaluated in the NASBench-101 dataset illustrating the importance of exploring different distributions of filters.

Further, chapter 5 presents a study to find differences in the internal representations of each template and then, relate the best performing template with features provided for the CKA similitude metric.

Chapter 6 summarises the research presented in this thesis, outlines our main findings and puts them into perspective concerning their implications for the neural network designer community.

CHAPTER 2

RELATED WORK

This chapter revisits the evolution of convolutional neural networks since their first appearance. We particularly mention models that have been successful in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [170]. We describe their main innovations and show that one feature has been barely investigated in convolutional network models: the pattern design for distributing filters and how it influences the efficiency of models. This pattern comes from the earlier architectures and has been adopted in most subsequent models. Just a handful of architectures, all of them recently created, have been built using a different distribution. The pattern has been so widely accepted that even in most automatic discovery methods, this feature is not considered in the exploration space, but it is just adopted as the default distribution. This chapter also revisits other techniques conceived to optimise models that affect the distribution of filters like pruning and distillation. We enumerate them in this chapter as well as methods for automatic discovery of models.

2.1 Evolution of Convolutional Neural Networks

Although the evolution of neural networks has been described in numerous articles [6, 7, 115, 161, 171, 179], we go again through the different models having a glimpse on features extrinsic to network architectures such as activation and loss functions, parameter optimisation or regularisation. Additionally, we focus on architectural innovations, such as multi path modules, deeper layers, residual connections and grouped

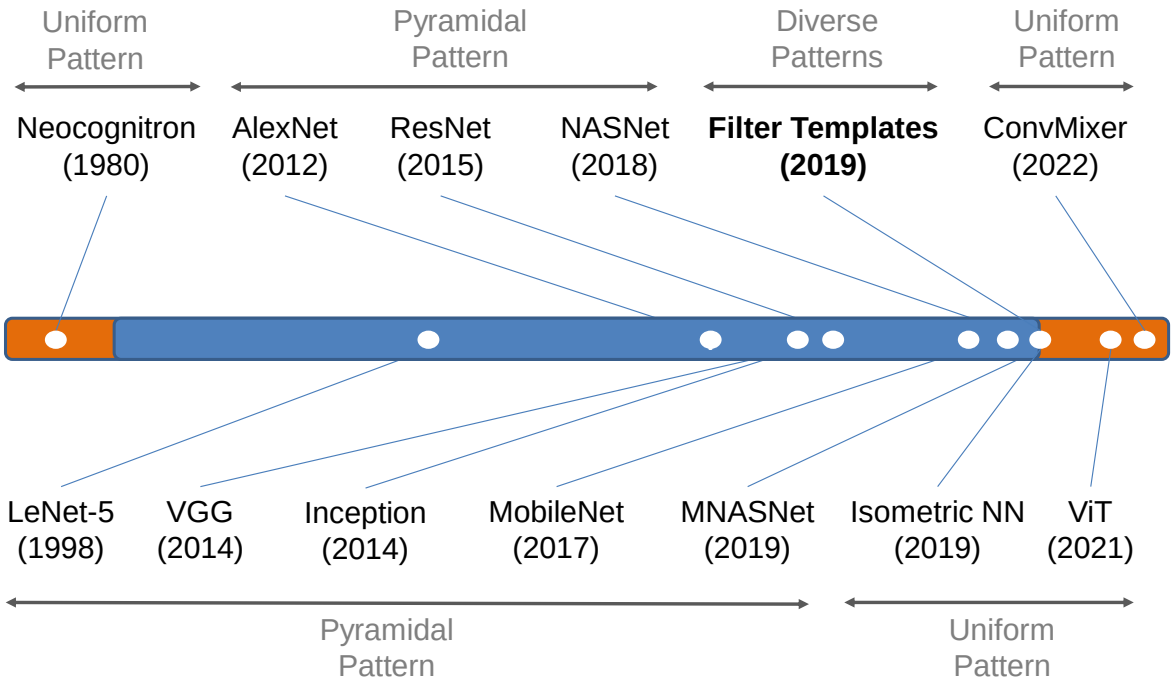
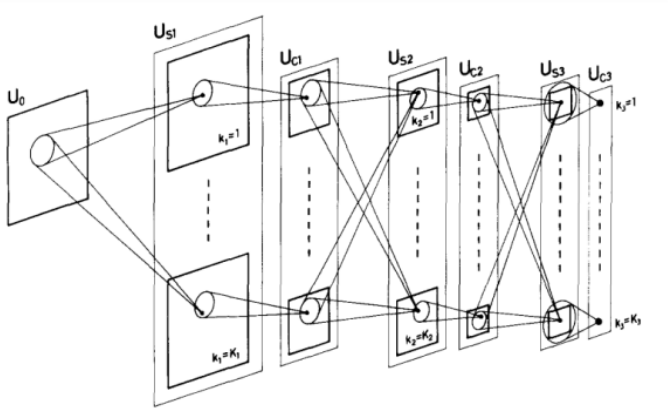


Figure 2.1: Year of introduction of well-known CNN architectures cited in this section showing used patterns of filter distribution. Our work is shown in bold.

convolutions, like the work in [94] but emphasise designers' decisions on the distribution of the number of filters within the layers. This characteristic is significant in our work, and we will show later that from the first successful CNN architecture, almost all subsequent convolutional models have been using an increasing pattern for the distribution of neurons. As far as we know, there is no other justification for keeping this incremental distribution than "to keep the richness of the representation" [106].

Since LeNet's emergence in 1998, there has been a massive expansion of research around the pyramidal distribution. It has been taken for granted that the design is optimal for all models and datasets, at least in the computer vision domain. Recently, researchers are looking back building new architectures using a different pattern to the pyramidal design [46, 174, 192]; however, they are only exploring the uniform pattern. This work calls for extending the search to other different exploration spaces for filter distributions, and it shows that some of these new distributions can produce more efficient models (See Figure 2.1).



The neural network proposed here has been simulated on a digital computer. In the computer simulation, we consider a seven layered network: $U_0 \rightarrow U_{S1} \rightarrow U_{C1} \rightarrow U_{S2} \rightarrow U_{C2} \rightarrow U_{S3} \rightarrow U_{C3}$. That is, the network has three stages of modular structures preceded by an input layer. The number of cell-planes K_l in each layer is 24 for all the layers except U_0. The numbers of excitatory cells in these seven layers are: 16×16 in U_0, $16 \times 16 \times 24$ in U_{S1}, $10 \times 10 \times 24$ in U_{C1}, $8 \times 8 \times 24$ in U_{S2}, $6 \times 6 \times 24$ in U_{C2}, $2 \times 2 \times 24$ in U_{S3}, and 24 in U_{C3}. In the last layer U_{C3}, each of the 24 cell-planes contains only one excitatory cell (i.e. C-cell).

Figure 2.2: Layers in Fukushima's Neocognitron with the description of a uniform pattern for filters' distribution. Image and text from [56].

2.1.1 Early Neural Networks

Origins of artificial neural networks (ANN) go back to several decades ago with the work published in the forties by McCulloch and Pitts, describing a mathematical model for the behaviour of nets of biological neurons [131]. Among the model's limitations were the lack of a learning procedure and the use of binary data. A more complete neuron model was introduced in the Perceptron [167]. It works as a binary classifier firing the neuron when a weighted sum of the inputs exceeds a predefined threshold. The single-layer model incorporated an algorithm for learning, and it was not limited to processing binary inputs; however, it was found the Perceptron was capable of solving only linearly separable functions [135]. Ivakhnenko and Lapa published the first functional networks with multiple layers, later known as multilayer perceptron (MLP), in 1965 [177]. Still, only until the work of Rumelhart, Hinton and Williams in 1986, it was shown neural networks could find solutions for non-linear classification problems. Using backpropagation as a learning technique, these multilayer networks can produce meaningful internal representations in intermediate layers [169]. The first model resembling modern convolutional networks was Fukushima's Neocognitron, published in 1980 (see Figure 2.2). This model has the ability to be invariant to changes in position or tiny distortions in shape in stimulus patterns [56].

2.1.2 First Convolutional Networks

The architecture of the Neocognitron carries a hierarchical multilayered structure in which the neurons in deeper layers respond to more complex features of the input pattern.

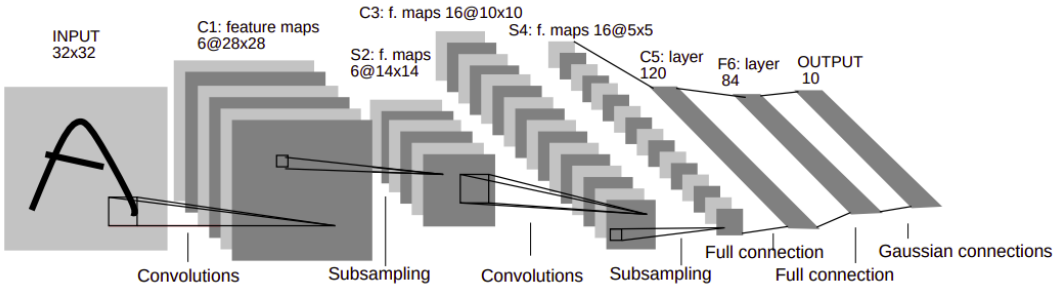


Figure 2.3: Layers in LeNet-5 architecture. Image from [106].

The paper outlines that the number of cells in each layer decreases with the layer's depth. Nevertheless, the experimental section depicts a similar number of neurons in each intermediate layer (24 to be specific). The first successful application of a CNN was presented in a handwritten zip code recognition system [105]. The authors trained a convolutional network, alternating convolutions with subsampling layers and stacking a couple of fully connected layers at the end. This model is the first constructed with the increasing pyramidal pattern, and it was named LeNet-5 (Figure 2.3). It presents all the essential elements of current CNNs. Its design has a set of three convolutional layers (6, 16 and 120 neurons) and two fully connected layers (84 and 10 neurons).

AlexNet [100] builds on LeNet-5 design and introduces several improvements, still in use in many current CNNs, that had been developed separately: ReLu for activation functions [143], max-pooling layers for subsampling [160], dropout for regularisation [79] and training process performed in GPU [185]. This last feature exploits the highly parallelisable nature of neural network operations, drastically reducing the training time. Its design consists of eight layers arranged in an almost pyramidal distribution (96, 256, 384, 384 and 256 units in convolutional layers). This model was the winner of the ImageNet Large-Scale Visual Recognition Challenge [170] in 2012, beating handcrafted feature encoding methods by a considerable margin.

One crucial work that redefined the shape of AlexNet filters distribution to be the most widely used distribution is the one presenting ZFNet model [214]. The paper experimentally shows that using small 3x3 filters and changing the number of filters to 512, 1024 and 512 in the last convolutional layers decrease the validation error on ImageNet by three percentage points.

The most recent models resembling LeNet-5 are the family of VGG architectures [183]. The authors adopted the distribution of filters proposed by Zeyler and Fergus [214] and tested different architectures by changing the depth. Filters were assigned following

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.4: Distribution of filters by layer in VGG architectures. Image from [183].

the incremental pattern 64, 128, 256, 512, 512. To cope with the different depths, they grouped convolutional layers in five blocks, as the number of layers in AlexNet, for each particular architecture and assigned the same number of filters within the block (see Figure 2.4).

2.1.3 Deeper, Wider and Denser

Although layers within blocks in VGG has nothing in particular other than the same number of filters, designers realised this strategy allowed easier development of new and deeper models. It is more flexible to design a small structure and then repetitively stack it to form a complete network [119]. The GoogLeNet [187] model was built using identical Inception modules. The authors state that the network topology tries to approximate locally sparse structures with dense elements to avoid increasing computational resource utilisation while conveniently exploring different-sized patches. With the increase in depth, GoogleNet faced the problem of vanishing gradients [80]. The solution was to add

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 2.5: Distribution of filters by layer in GoogLeNet architecture. Image from [187].

auxiliary outputs to translate gradient signals to lower parts of the architecture. We can observe in Figure 2.5 that the pyramidal distribution of filters remains.

Following this block-style design emerged ResNet architectures [73] with a formulation that enables very deep networks to be more easily trained. Their blocks are constructed with residual connections that help reduce the degradation problem, leading to poorer performance when stacking numerous layers in a neural network model [61]. In Figure 2.6, we can infer that the authors used the same filter distribution adopted in VGG.

Another family of models with a solution for diminishing gradients is FractalNets [102]. The authors argue that residual connections are not necessary, and they proposed drop-path regularisation, which randomly enables a single column subnetwork to be trained at once. They also declare in the FractelNet paper: “we set the number of filter channels within blocks 1 through 5 as (64, 128, 256, 512, 512), mostly matching the convention of doubling the number of channels after halving spatial resolution”.

After ResNet, based on the idea that deeper models are better models [194], researchers started training networks up to more than one thousand layers [74, 85]. Although they utilised varied strategies for enabling convergence, such as an adaptive rectifier and a robust initialisation method [72], very soon they realised that the cost of slightly improving performance came at the expense of meaningfully increasing the number of layers. Very deep models still suffered from diminishing gradients and a very

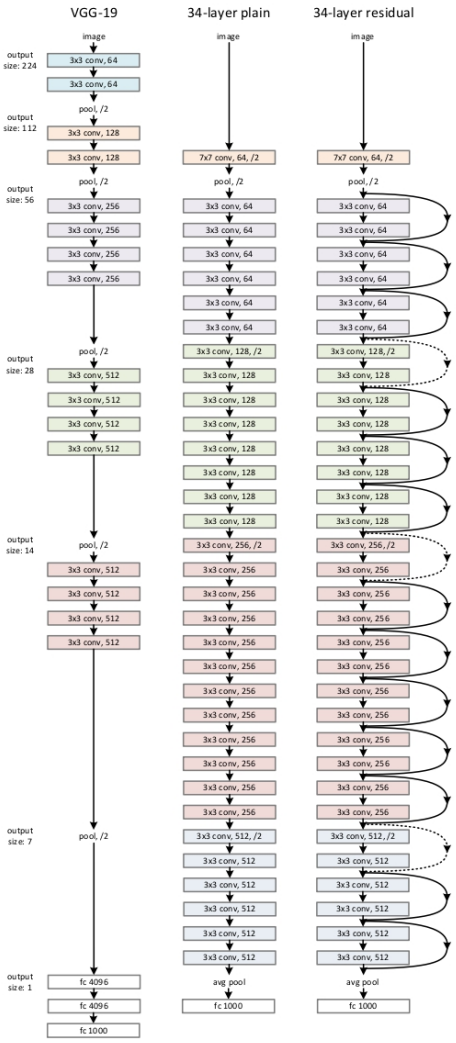


Figure 2.6: Comparison of filters by layer between VGG and ResNet architectures. Image from [73].

long time to train.

A natural alternative to keeping increases in depth was to scale models in width [11]. WideResNets [213] are constructed using residual modules in networks from 16 to 52 layers, proportionally scaling the number of filters from 2X to 12X. Being shallow, wide residual networks are faster than deeper networks, but on the other hand, they count more parameters and therefore are prone to overfitting. For this reason, the authors added dropout layers within residual modules to work as regularisers. Similar to previous architectures, the WideResNet base model adopts an increasing distribution of filters, as shown in Figure 2.7.

Motivated by the success of residual connections, designers created DenseNet [84].

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Figure 2.7: Distribution of filters (16, 16, 32 and 64) in a Wide Residual Network with the original residual block. Image from [213].

The architecture leverages feature reuse by adding short paths to every previous layer inside a block. Because of these dense connections, the model has a low number of parameters but at the cost of high memory utilisation and increasing FLOPs. Feature maps in previous layers must reside in memory to be concatenated and processed again along with the new features. The computational burden restrained the authors from increasing the number of new filters in each block to only twelve. The DenseNet block has a constant number of new filters in each layer, but the resulting number of feature maps in the global design still resembles an increasing distribution.

Since the first CNN models, the incremental distribution of filters has followed the shape of a stepped pyramid. The "steps" are found in the downsampling layers, where the feature map resolution decreases by the pooling operation, and the number of filters increases, doubling the previous ones. The work presented in [198] unveils a particular behaviour in residual networks: removing blocks from the architecture at test time have little impact on the final accuracy. This is because the residual connections make the remaining subnetwork act as if it has been trained isolatedly, and then the final model performs as an ensemble of subnetworks. The only block removals considerably hurting the accuracy are those that happen in the sections next to the downsampling layers. PyramidNet designers build on this outcome and hypothesise that a more reliable ensemble distributes the damage in all the blocks [70]. They propose to smoothly increase the number of filters in each layer within the block as shown in Figure 2.8.

2.1.4 Efficiency-Oriented Architectures

Once designers found hardware limitations for training and running deeper and wider networks, they looked for solutions to reduce computational resource consumption of

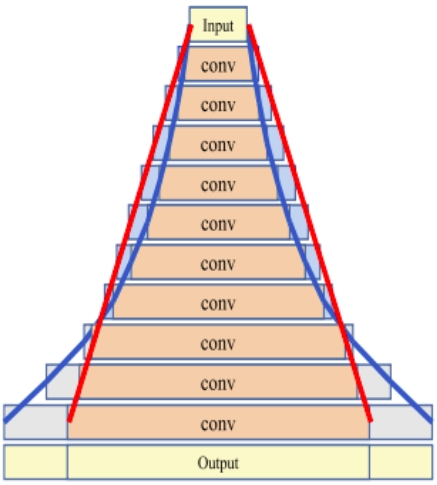


Figure 2.8: Number of filters in additive PyramidNet (red) and multiplicative PyramidNet (blue). In both cases filters are smoothly distributed along layers. Better experimental results were obtained with the additive distribution. Image from [70].

models instead of exclusively improving accuracy. One of the first models to show a dramatic reduction of resource demands was SqueezeNet [86]. Using a module known as Fire block, the model can reach AlexNet comparable accuracy with a much smaller number of parameters. The reduction is achieved by reducing channels entering the 3x3 convolutions within the Fire block with a few 1x1 filters. The authors present an extensive description of the exploration space for the hyperparameters defining the architecture; however, filters in each layer are kept with the incremental distribution without further explanation.

The GoogLeNet model suffered several refinements leading to better accuracy and more computational efficiency. Firstly by adding batch normalisation layers [87]; secondly, by decomposing 5x5 convolutions with two cheaper sequential 3x3 ones; lastly, by factorising nxn convolutions in nx1 followed by 1xn convolutions [188].

The author in [29] took inspiration from Inception modules and combined them with the concepts described in [89, 182] of factorising convolutions to reduce computational demands. This type of convolution, called depthwise separable convolutions, can significantly reduce the number of operations performed in a neural network while keeping similar accuracy. The rationale behind this idea is that the information contained in the channels can be disentangled from spatial information. The distribution of filters in this new model, named Xception, remains similar to the GoogLeNet model.

Almost concurrently with the appearance of Xception, a family of models called MobileNets were proposed aiming to run efficiently in constrained environments [82].

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2.9: Number of filters in the MobileNet base model. Designers can fit variate computational budgets using the architecture with a width multiplier. Image from [82].

MobileNet designers extensively used depthwise separable convolutions composed of 3x3 depthwise convolution filters performed in each input channel, followed by pointwise convolutions made of 1×1 convolutions.

Additionally, designers introduced two extra parameters to control the model's capacity: the width and resolution multipliers. These hyperparameters provide a flexible design that allows the use of MobileNets in a broad range of applications. We emphasise that the width multiplier proportionally increases or decreases the number of filters but follows the base pyramidal distribution shown in Figure 2.9. Similar to GoogLeNet, MobileNet architecture has been refined to reach higher accuracy with fewer resources [175]. MobileNet V2 authors introduced the inverted bottleneck module, which on one side, alleviates the loss of information caused by embedding the representation manifold into a lower subspace with a non-linear transformation. On the other side, it incorporates residual connections between the bottlenecks keeping expansion layers in the middle of the module and saving computational costs.

2.1.5 Grouped Convolutions and Attention

The strategy of the Inception module of splitting the input, processing it in separate branches with different filter sizes, and then merging all the branches again is the inspiration for a new type of architecture. The first of them is the ResNeXt model [205], second place of ILSVRC 2016. The authors built a multipath structure taking parallel ResNet blocks that split the input and merged the output at the end. Instead of having specialised filter dimensions, the ResNeXt block uses similar 1x1 and 3x3 filters in all the paths. The number of paths (known as cardinality) is a new hyperparameter that adjusts the size and, therefore, the accuracy of the network. The macrostructure of ResNeXt is similar to the ResNet model but proportionally counts much more filters. However, the number of parameters and FLOPs are equivalent due to the implementation of multibranch modules as grouped convolutions.

These grouped convolutions, combined with depthwise convolutions, are also present in ShuffleNet, a model designed to be small and efficient [217]. The authors exploit the assumption from [216] that keeping channel information separated in each group (or branch) could limit the network's performance and that randomly shuffling channels could reduce the effect. Similarly, they proposed to mix the channels between groups. In this way, all grouped convolutions can access information contained in the rest of the branches without increasing computational requirements. However, the idea of doubling the number of channels every time the feature map size is reduced still rules this design.

The concept of attention has been studied in several works [18, 88]. It is a mechanism that enables dynamically weighting features produced for convolution layers depending on the input being processed to select the more important ones for the task. Some of the works building effective attention mechanisms are found in dasNet [184], stacked hourglass networks [144], and residual attention networks [199]. But it was the SENet model which won the ILSVRC 2017 using attention. The authors of SENet modified existing block-based architectures such as ResNet and Inception by adding an attention structure called squeeze and excitation block (a.k.a SE block) [83]. The SE block first captures global spatial information into a channel embedding. Then it learns the importance of each channel using a pair of fully connected layers with a non-linear transformation. The output of the original ResNet or Inception block is scaled according to the importance learned by the SE block. The distribution of filters in the final model is not affected by the modifications of the SE block, as reflected in Figure 2.10.

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d fc, softmax		

Figure 2.10: Comparison of the number of filters in ResNet against ResNet and ResNeXt architectures using SE blocks. SE enabled models are identified by the SE prefix. Image from [83].

2.1.6 Recent Developments for Uniform Distribution

Since the start of this Thesis, there have been other works related to revisiting the uniform distribution. We cite in this section state-of-the-art networks created with this distribution of filters. They underline that, by not necessarily following the pyramidal pattern, it is possible to achieve higher performance and structure simplicity, making them easier to implement. We note that the first recent network successful with the uniform distribution was published the same year our filter templates were publicly presented.

In 2019, the first work was introduced following the neocognitron filter distribution. The authors call the new type of networks Isometric Neural Networks (INN) [174]. The research behind INN is not focused on the distribution of filters. Instead, the paper 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 on the final performance of the network. Based on the findings, the authors of INN propose a fixed internal resolution across the architecture. Consequently, they keep a constant number of filters in each layer (see Figure 2.11). As a result, INN not only performs higher than MobileNets, but they count fewer parameters and memory footprint.

Based on the attention mechanism, the Transformer architecture appeared in the natural language processing (NLP) domain. It was created to address the problem of lack of parallelisation, and the subsequent slow training procedure, produced for

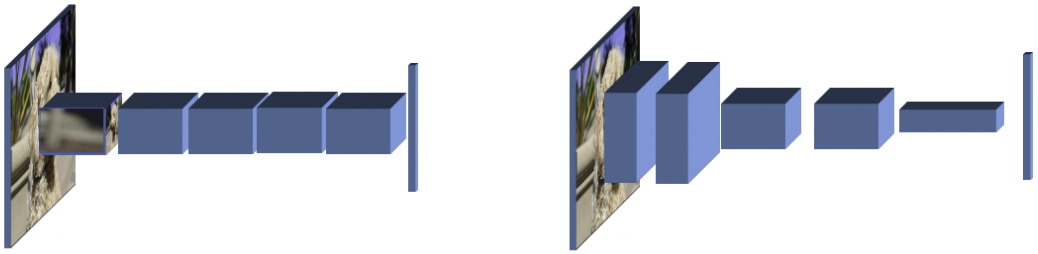


Figure 2.11: Comparison of an isometric neural network (left) versus a standard pyramidal architecture (right). Blue blocks schematise the resolution and number of feature maps. Image from [174].

recurrences in existing networks [197]. Although attention is used in several computer vision-based models, it is combined with convolutional networks. The Vision Transformer (ViT) has been adapted to work with patches of images instead of tokens (words) [46]. Like the tokens in the NLP transformer, the patches are encoded with a linear learnable embedding. The authors removed the convolutional structure entirely and utilised the attention mechanism proposed in the original transformer. Its performance beats CNN based architectures when trained in massive datasets and then fine-tuned to ImageNet. The ViT network is composed of constant size transformer encoders keeping equal resolution throughout all layers in a uniform pattern.

The use of image patches at the input of ViT networks raised the question of whether their success originated from the transformer encoder or from the use of patches. Researchers proposed using patches (previously embedded as in ViT) as inputs of a convolutional network called Convolutional Mixer (ConvMixer) to answer the question [192]. Instead of using the classical pyramidal pattern for assigning filters in each layer, the authors of ConvMixer copied the uniform resolution and number of channels from the ViT network (see Figure 2.12). The resulting very simple architecture outperforms the accuracies of ViT and ResNet on ImageNet using similar parameters.

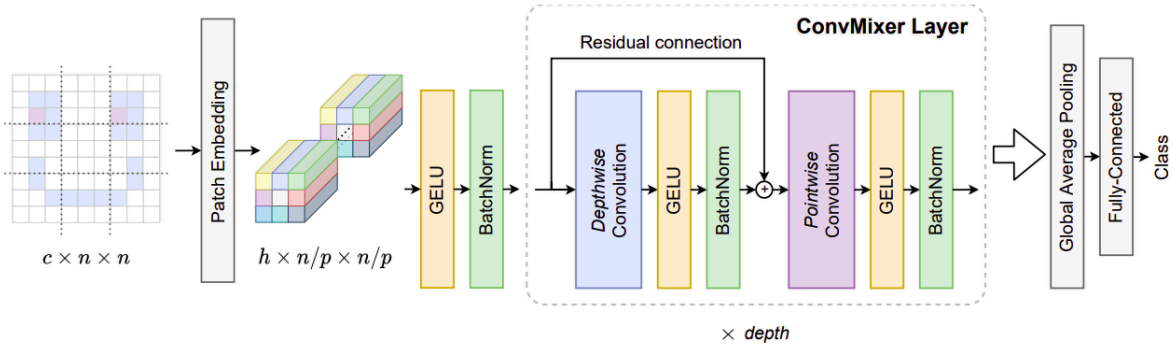


Figure 2.12: ConvMixer structure uses copies with the same number of filters of a simple convolutional block. Image from [192].

2.2 Methods for Neural Network Model Compression

In the previous section, we listed the advances in convolutional neural networks. Despite new improvements having brought more powerful models, they were achieved at the cost of increasing models' computational complexity [94]. Many of these networks, especially the best-performing ones, necessitate massive amounts of computing and memory. These restrictions not only raise infrastructure costs but also complicate network implementation in resource-constrained contexts like mobile devices [138].

The necessity of finding a way to reduce these high demanding networks encouraged the development of techniques to compress models without decreasing their performance. Theoretical findings state that a smaller system shows better generalisation performance [17]. Neural models are over-parameterised [40, 176]. Then, it is considered they are amenable to be reduced in size and, therefore, in resource requirements [39] always that the network dimension surpasses a certain minimum threshold [13].

Compressing a neural network allows multiple advantages. The final model is usually faster to train, with less latency and reduced memory needs. These features enable it to be deployed in low-energy devices used in mobile applications. Moreover, implementing numerous compressed models in big data centres can help significantly reduce the total energy consumption [116].

We revisit in this section two important compression methods for neural networks related to our work: network pruning and knowledge distillation. Although templates are not purposely designed like a model compression technique, many reach comparable resource demand reductions.

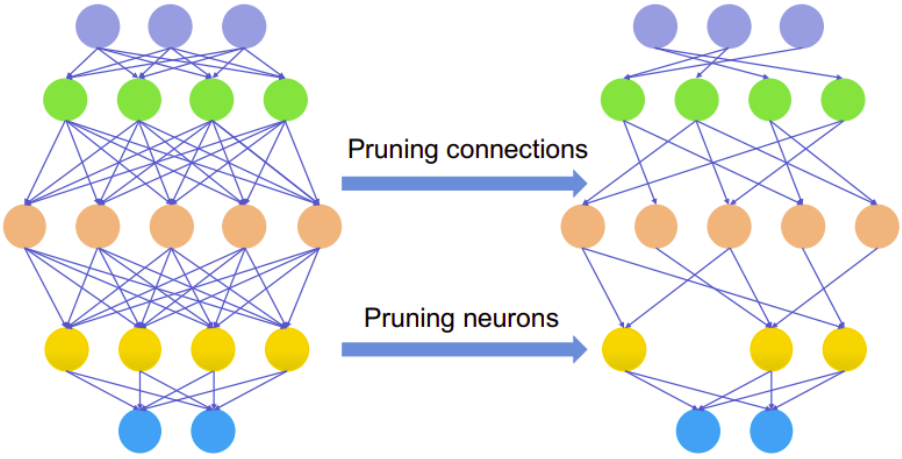


Figure 2.13: Unstructured and structured pruning approaches. The original model (left) is reduced by eliminating some of the weights (top right) or complete neurons (bottom right). Image from [25].

2.2.1 Neural Network Pruning

The purpose of pruning methods is to achieve a reduction of unnecessary elements in neural network models such as connections, filters, or layers [16, 206]. Early pruning works were focused on reducing the number of connections (parameters) [107, 142, 163]. The methods are known as unstructured pruning, and although they have shown to be successful [47, 126, 139], the resulting networks are difficult to implement due to their sparsity, requiring special hardware and libraries [71]. Recent studies claim that the levels of compression achieved by these methods are not reachable by training the reduced model from scratch [57].

On the other hand, structured pruning, which prunes at the levels of channels, layers or even blocks, produces dense models that harvest the characteristics of actual GPU devices. Figure 2.13 schematise the difference between the two approaches. Among all the levels of structured pruning, the most popular is carried out by removing complete neurons [75, 76, 104, 128, 208] using some heuristic. Thus, the final model ends up with a different distribution of filters. We interpret this definition as if it is implicitly assumed in pruning methods that the original distribution of filters is not optimal for the architecture being reduced. Pruning methods assume there exists one distribution that uses fewer filters but is still capable of reaching similar or even superior accuracy. In that sense, our work is related to neural network pruning because we explore some radically different distributions of filters for existing models.

We describe below the three main beliefs that have been under discussion in the last years regarding the importance of trained weights in a neural network:

1. Final weights are important, and then, neural networks can be reduced after/during training by cutting less relevant elements.

2. Initial and final weights are important, therefore, neural networks can be reduced after/during training by removing less relevant elements and then training the subnetwork again from the original initialisation.

3. Only structure is essential, so neural networks can be reduced before training by removing less relevant elements out of budget.

2.2.1.1 Pruning Based on Final Weights

Like the earliest pruning methods, the first group of works relies on the final weights of the model. The network has to be fully trained, and then, metrics about the importance of each element are used to decide whether to delete any of them. By far, this is the predominant belief [10, 163], and it was adopted since the Optimal Brain Damage (OBD) work [107]. It is commonly acknowledged that trained weights with high values are more critical than those with low values. Furthermore, it is possible to group weights using the $\ell 1 - norm$ to remove complete filters [110]. This removal of weights with zero or nearly zero values is the purpose of norm-based pruning algorithms. This means that the distribution of weight magnitudes should be sufficiently wide to include enough weights near zero to produce a small network. In some cases, however, this doesn't happen, making pruning based on a threshold problematic [75].

Pruning the final weights eliminates network redundancies while lowering the number of calculations without compromising accuracy. However, because the criterion to select elements to remove is not always precise, certain critical elements may be left out, resulting in a reduction in accuracy. To compensate for the loss of precision, the model has to be retrained for some epochs using lower learning rates than in regular training in a process known as fine-tuning [165] (see Figure 2.14).

2.2.1.2 Pruning Based on Initial and Final Weights

The idea of retaining initial weights was first introduced in the lottery ticket paper [51]. The authors hypothesise the existence of particular subnetworks (a.k.a. winning tickets) hidden in an entire network model with such characteristics that, when trained isolated,

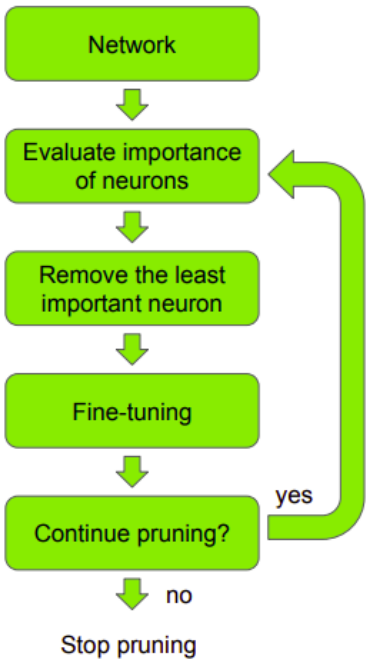


Figure 2.14: Typical prunning pipeline to reduce a neural network model with minimal diminishing in accuracy. Image from [140].

they can compete in accuracy with the original network. To be able to discover these winning tickets, the whole network is fully trained from random initialisation. Next, the network is pruned with some existing pruning methods. Then, the remaining model needs to be initialised to their original random weights and be retrained. Experiments indicate that dense, randomly initialised sub-networks may be trained effectively and with the same training iterations as the original network.

According to a follow-up study [220], the winning tickets perform better than randomly on some datasets even without any training. In light of this, they propose a technique for locating a good subnetwork inside a randomly initialised network with high accuracy.

There have been several improvements over the original lottery ticket paper extending the hypothesis with a new conjecture[129] to find winning tickets without training [159] or providing more efficient methods for winning tickets discovery at early training stages [52]. In addition, some authors have tested the lottery ticket hypothesis in new architectures [26] and tasks [59].

2.2.1.3 Pruning Based On Structure

These methods challenge two central assumptions in the field of neural network pruning [124]. The first one assumes that starting with a large network is necessary because it provides high accuracy. Then because of the over-parameterisation of the model, one can remove redundant weights safely with negligible reductions in performance. The second assumption is that after completing the model training, the final weights are relevant for selecting the portions of the network that should be pruned.

Experimental evidence of the superior performance of large models over pruned models trained from scratch has been presented in several works [110, 212] however, authors of [124] claim that superiority is not undoubtedly true for structured pruning methods. They found that a small model randomly initialised can perform similar or better than the original model from which the small model was obtained. Therefore, the model can be reduced directly to the required target size and trained from scratch. This finding suggests that the reduced architecture may be more relevant for these pruning strategies than the conserved weights. The approach presented in our research about templates relies on the previous result. We claim that we can change the filter distributions in a model straight away without doing any pretraining and still obtain competitive accuracies.

2.2.2 Knowledge Distillation

While neural network compression is performed in pruning by removing less important elements, knowledge distillation (KD) methods follow a different approach of replacing the complete neural network (Teacher) with a smaller one (Student) [78] (see Figure 2.15). KD methods are based on the same premise used in pruning methods that models are over-parameterised and therefore amenable to be reduced. However, KD methods rely on a second assumption with a broad application than neural networks: it is easier to train a small model not on the original data but in the mapping function the big model has learned [22].

Although the first work focused on obtaining a small model from a pretrained cumbersome model [78, 112], recent works have proposed to perform the training process of the teacher and the student concurrently [8, 203]. Furthermore, KD techniques are not restricted to distil knowledge from a single big model but multiple ones combined in an ensemble. Training the student network requires not the original labels of the data but the processed labels from the ensemble (soft labels). The standard approach to

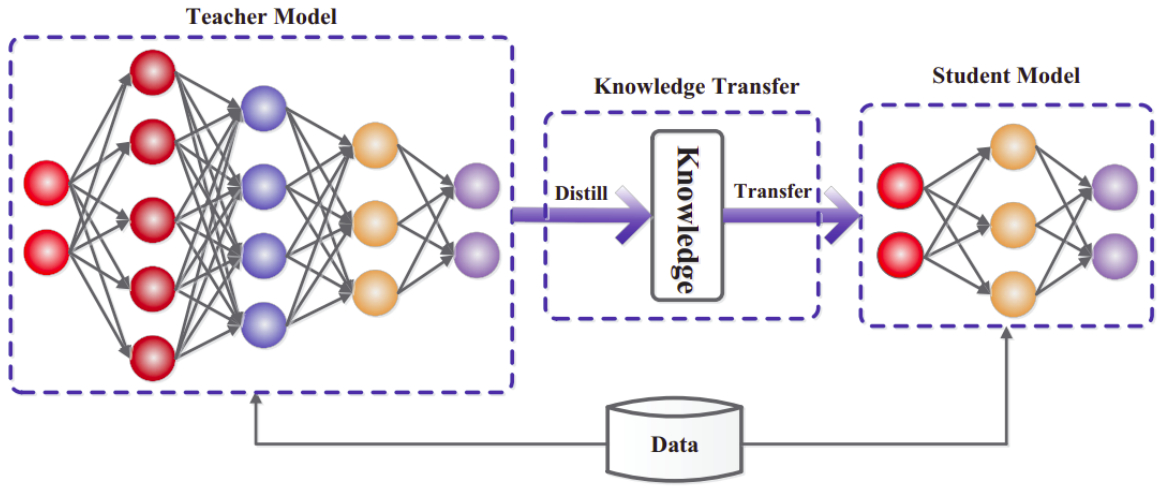


Figure 2.15: The knowledge distillation framework is composed of teacher and student models. Image from [65].

combining multiple teacher models is averaging the outputs [23, 130]. Some other works have proposed to combine the output distributions [54, 145].

The area of Knowledge distillation continues to be very active [4] with applications in numerous domains such as image classification [168], semantic segmentation [122], medical data mining [134] or video captioning [148]. Currently, several efforts are made to have a theoretical understanding of the distillation process [5, 150, 153].

2.3 Neural Architecture Search

The process of designing a neural network is a task mainly based on experience and experimentation that consumes a lot of time and computational resources. With the increase in the use of neural networks, particularly convolutional networks for computer vision problems, a mechanism to automatically find the best architecture has become a requirement in the field of Deep Learning. Some works were published several years ago [101, 155] trying to solve the topic of automatic architecture generation, but they did not provide competitive results compared to handcrafted architectures. Modern algorithms for neural architecture search (NAS) have rapidly reduced the gap, and recently [221], they are capable of producing some of the highest performing models with minimal human interaction [49].

Some approaches to NAS design are reported in [113]. The authors propose a classification of methods based on three elements: search space, search strategy and perfor-

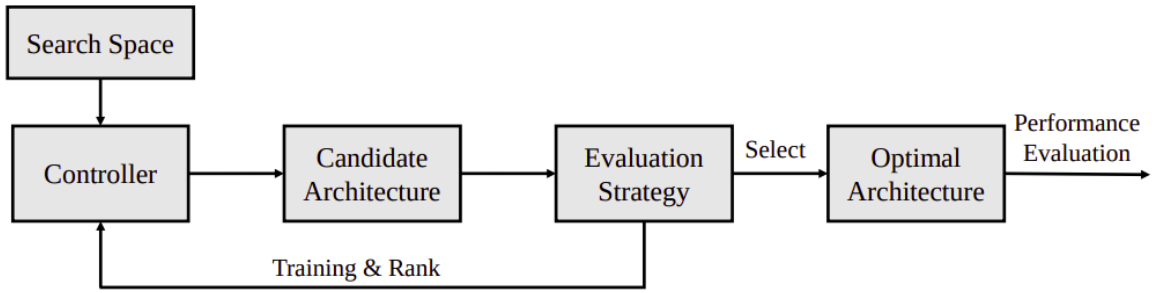


Figure 2.16: Modern framework of NAS methods. Image from [164].

mance estimation strategy. These aspects remain present in modern frameworks of NAS as depicted in Figure 2.16.

Generally speaking, methods for automatic architecture discovery operate similarly. A search strategy chooses an network from a predefined search space using a controller. Next, the candidate architecture is passed to a evaluation strategy, which reports back the true or estimated performance of the sample to the search strategy. Knowing the network's performance, the controller iteratively enhances its ability to sample selection. At the final step, the NAS framework fully trains the best models choosing the highest performing one.

One of the biggest challenges in automatic architecture design is that the search space for CNN architectures is infinite. 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 is still complex [31, 50, 81] and many approaches have decided to borrow previously published search spaces [120, 189] (see Figure 2.17 for some examples of search space). Interestingly, the search space for most NAS methods is restricted to different sets of layers and their connections. However, the distribution of the number of filters in each layer follows an incremental pattern similar to the ones found in the models described in section 2.1.

The exceptions to the rule 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 [64, 109]. CNS methods usually reduce the computational burden caused for the exploration of the search space by parameter sharing [15, 44, 200, 209]. However, most of the architectures used as base models to initialise the automatic search in CNS methods share the practice of increasing filters resembling LeNet design[106].

The network architecture construction process iteratively adds simple blocks based on the experience of the controller acquired by training [120], by analytical methods

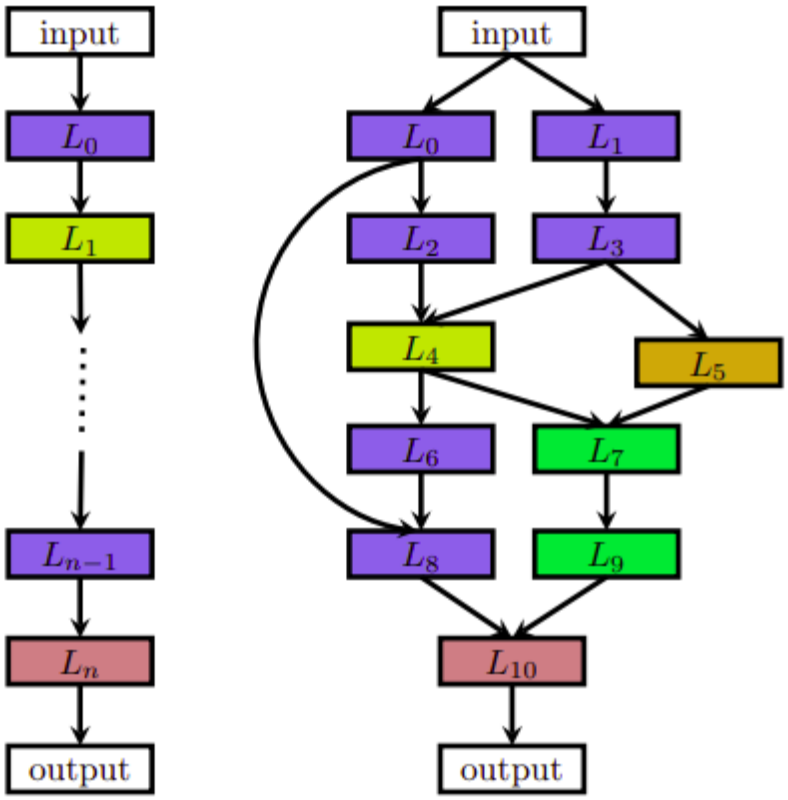


Figure 2.17: Two different architecture search spaces for NAS methods. A network structure is built following a sequential design (left) or a more complex pattern (right). Image from [49].

[36] or by the use of genetic algorithms [123, 127]. Another approach that reduces the exploration time dramatically is to have a graph defining all networks in the search space. Then, the edges of the graph defining this super network with shared weights are reduced by a controller [152], or with stochastic gradient descent [121] alternatively while training the model's weights.

The weight sharing paradigm presented in [121] is leading most of the current NAS techniques [204]. The method uses a relaxation condition to transform the selection of layers in the architecture into a continuous space using a *softmax* function. The relaxation allows performing a simultaneous search of weights and architecture using gradient descent. The method converges after one day of GPU time, surpassing the time-consuming previous methods. Most of them with computational demands for the search process normally in the order of thousands of GPU days (see Figure 2.18).

Initially, a dataset was absent with examples of the best architectures for a particular

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
AmoebaNet-A (Real et al., 2019)	3.34(0.06)	3.2	3150	evolution
PNAS (Liu et al., 2018a)*	3.41(0.09)	3.2	225	SMBO
ENAS (Pham et al., 2018)	2.89	4.6	0.5	RL
NASNet-A (Zoph et al., 2018)	2.65	3.3	2000	RL
DARTS (1st) (Liu et al., 2018b)	3.00(0.14)	3.3	0.4	gradient
DARTS (2nd) (Liu et al., 2018b)	2.76(0.09)	3.3	1.0	gradient
SNAS (Xie et al., 2018)	2.85(0.02)	2.8	1.5	gradient
GDAS (Dong & Yang, 2019)	2.82	2.5	0.17	gradient
BayesNAS (Zhou et al., 2019)	2.81(0.04)	3.4	0.2	gradient
ProxylessNAS (Cai et al., 2018)†	2.08	5.7	4.0	gradient
P-DARTS (Chen et al., 2019)	2.50	3.4	0.3	gradient
PC-DARTS (Xu et al., 2019)	2.57(0.07)	3.6	0.1	gradient
SDARTS-ADV (Chen & Hsieh, 2020)	2.61(0.02)	3.3	1.3	gradient
TE-NAS (ours)	2.63(0.064)	3.8	0.05‡	training-free

* No cutout augmentation.
† Different space: PyramidNet (Han et al., 2017) as the backbone.
‡ Recorded on a single GTX 1080Ti GPU.

Figure 2.18: Performances of models and resources consumed by NAS methods on the CIFAR-10 dataset. Image from [27].

problem to feed the controller network to acquire experience in designing and selecting. Therefore, one popular alternative for training was reinforcement learning (RL) [12]. But the evaluation process of the predicted architecture is carried on a standard manner by training it with a large number of iterations, and it can only generate classical architectures composed of sequential layers of convolutional, pooling and fully connected blocks.

Researchers have proposed several improvements to build better controllers. In [21] they present a mechanism to rank a group of CNN architectures by generating initial weights with an auxiliary network. Those weights provide enough information to accurately sort the architectures based on the validation performance of each configuration with a few iterations. Recently several works released datasets to facilitate the evaluation of controllers. In particular, the NASBench-101 dataset [207] contains the validation accuracy of all the fully trained models from the typical search space for NAS (see Figure 2.19). New datasets have been extended to wider search spaces but use a surrogate model to approximate the true validation accuracies [45, 181].

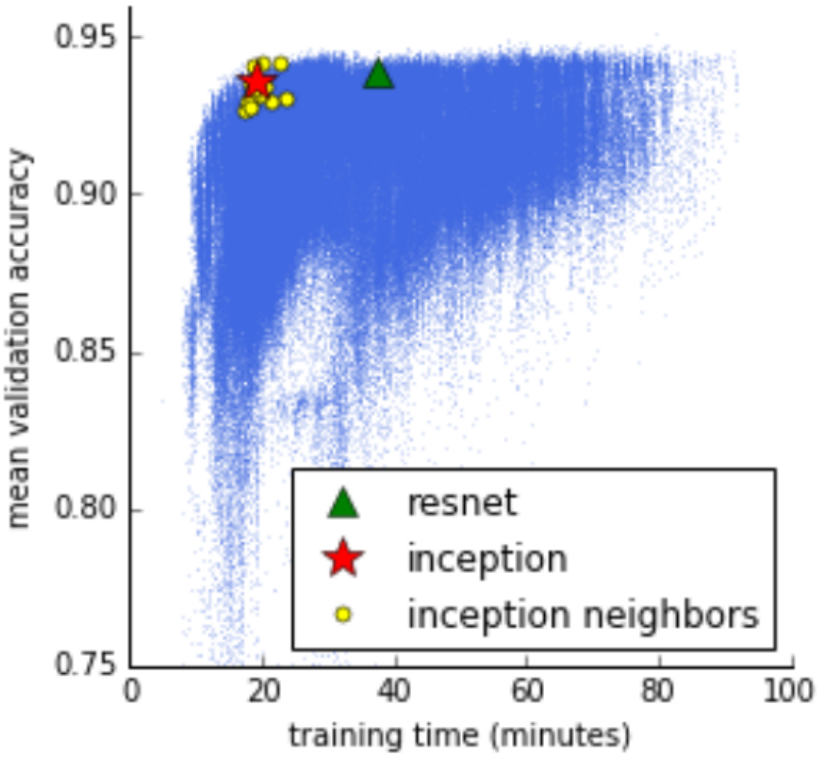


Figure 2.19: Validation accuracy of fully trained models in NASBench-101 dataset. Every blue point represents a trained model. Image from [207].

2.4 Conclusions

Since their early origins, this chapter revisited convolutional neural network architectures, starting with the neocognitron. It showed that the distribution of filters began with a uniform pattern but switched to a pyramidal pattern since the LeNet introduction in 1989. Since then, the pyramidal design has been predominant 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 overfitting ImageNet.

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 fasters 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 aim to open the search for other filter distributions to benefit the deep learning community by obtaining more efficient models.

CHAPTER 3

FILTER DISTRIBUTION TEMPLATES FOR IMAGE CLASSIFICATION

This chapter challenges the widely used design of increasing filters in neural convolutional models by applying a small subset of diverse filter distributions, called templates, to existing neural network designs. Experimental evidence shows that simple changes to the pyramidal distribution of filters in convolutional network models lead to improvements in accuracy, number of parameters or memory footprint. We highlight that many recent models, which have had a more detailed tuning in the filter distribution, present resiliency in accuracy to filter distribution changes, which requires further research and explanation.

Experiments in this chapter use an equal number of filters in all templates without constraining the redistribution effects. We extend these experiments in chapter 5 where templates are evaluated with more rigorous experiments keeping FLOPs to similar values as in the original model and then comparing resource consumption.

3.1 Convolutional Neural Networks

A CNN consists of a set of layers. The first (input) layer is the one that directly takes an image for further processing through the following multiple hidden layers, typically including convolutional layers, pooling layers, fully connected layers and normalisation layers. Finally, the final layer (output) produces a prediction relative to the sample

introduced in the input layer.

Each convolutional layer L^n in a network of n layers apply M filters (we will use the terms filters, kernels and neurons similarly) through overlapping regions of the input image. Its functionality is defined by the number of input and output feature maps, filter sizes, and skipping factors (strides). A convolutional layer produces M equal-size feature maps with resolution (M_x, M_y). All filters of size (K_x, K_y) are convolved over the input image using steps S_x and S_y pixels in x and y directions. The M output map resolutions (M_x^n, M_y^n) for each layer are defined by the number and size of the filters in the equations:

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n} + 1$$

$$M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n} + 1$$

Each feature map created in layer L^{n-1} is processed as input by layer L^n producing M^{n-1} output feature maps. Filters convolving different sections of an input feature map share their weights but have different receptive fields [32].

Pooling layers combine the outputs of neuron clusters at one layer into a single neuron in the next layer using one sample (e.g. the maximum or the average value) from each cluster. Samples are taken from non-overlapping sections (patches) of $P_x \times P_y$ pixels. The benefit of pooling layers is to allow position invariance over wide local areas and reduce the processing of high resolution feature maps by a factor of $P_x \times P_y$ [19].

Fully connected layers have all their neurons connected to all activations in the previous layer, as seen in regular Neural Networks. This type of layer usually forms the last layers in a CNN architecture. As they are in charge of producing the final prediction, changing their size and activation function gives the CNN the flexibility to work in different tasks. Traditionally, a CNN ends with a Softmax activation function [20] for classification tasks, while a ReLu activation [2] is used in regression problems. Changing the last layer according to the required output is a way to build a task-dependent map using deep networks.

3.2 Popular CNN Architectures

The state-of-the-art networks evaluated represent some of the highest performing CNNs on the ImageNet challenge in the previous years[170]. They have been primarily tested

on classification tasks and also have demonstrated a solid ability to generalise to images outside the ImageNet dataset by transfer learning. Therefore, they are expected to perform well in regression tasks, as PoseNet is based on the GoogleNet network. Additionally, we present some state-of-the-art models designed to maintain a low number of parameters. We merely describe those architectures since we are not interested in the architectures themselves. We used them only as a map representation and as a tool for visual features exploration.

The VGG network architecture [183] is recognised for its simplicity (see Figure 2.4). It is composed of sequential convolutional layers followed by max-pooling reduction layers. Finally, two fully-connected layers and a softmax classifier manage the final classification. The main disadvantage of these networks is the size of their parameters, being the biggest of all the architectures evaluated.

PoseNet [93] is also an adaptation of GoogleNet to be used as a regressor (see Figure 2.5). The intermediate classifiers are discarded at test time, and the softmax classifiers are replaced with regressors. The authors placed a fully connected layer before the final regressor acting as a localisation feature vector.

Authors of ResNet [73] succeed on the problem of training very deep CNNs by reformulating the assumption that the network blocks are modelling a function closer to an identity mapping than to a zero mapping. Therefore, it should be easier to find differences with reference to an identity rather than a zero mapping. This assumption is carried out by adding additional references at the end of building blocks (see Figure 2.6).

The MobileNet network [82] is built on depthwise separable convolutions except for the first layer, which is a full convolution. All layers are followed by a batch normalisation layer and a relu nonlinearity except the final fully connected layer, which consists of a softmax layer for classification.

3.3 Convolutional Neural Networks and Their Default Filter Distribution

An important consideration to create a convolutional neural network (CNN) model is the number of filters required at every layer. The Neocognitron implementation, for example, keeps an equal number of filters for each layer in the model [56]. A very common practice has been to use a bipyramidal architecture. The number of filters across the different layers is usually increased as the size of the feature maps decreases. This pattern was first proposed in [106] with the introduction of LeNet and can be observed in a diverse

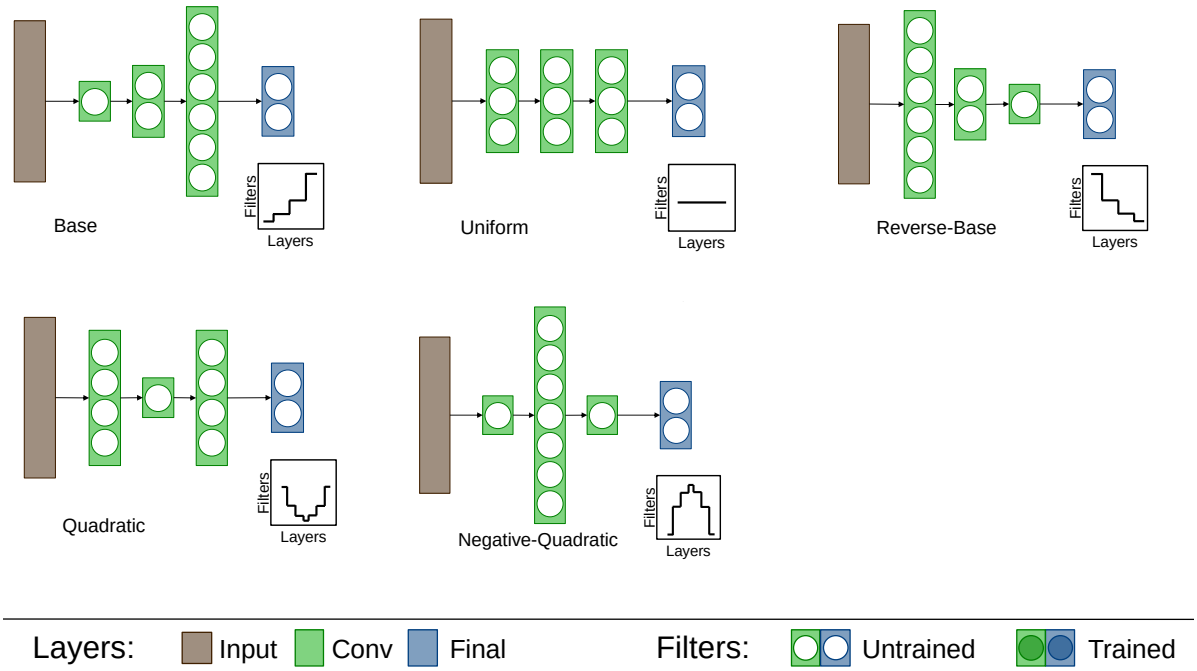


Figure 3.1: Filters per layer using the proposed templates for filter redistribution in a toy VGG style model. Base distribution, which is the original distribution, shows the common design of growing the filters when resolution of feature maps decreases in deeper layers. Although the total number of filters is kept constant after templates, changes in filter distribution induce different effects in performance and resource consumption.

set of models such as VGG[183], ResNet[73] and MobileNet[82]. Even models obtained from automatic model discovery, like NASNet [222], follow this principle since neural architecture search methods are mainly formulated to search for layers and connections. In contrast, the number of filters in each layer remains fixed. The motivation behind this progressive increase in the number of kernels is to compensate for a possible loss of the representation caused by the spatial resolution reduction [106]. In practice, it improves performance by keeping a constant number of operations in each layer [30]. It remains unknown if this pyramidal distribution of filters is also beneficial to different aspects of model performances other than the number of operations.

3.4 Why Not A Learned Function?

We have mentioned in section 2.3 that the main disadvantage of NAS and CNS methods is the high computational cost (some in the order of thousands GPU days) produced for searching the space of possible solutions, not taking into account such space is

incomplete and chosen based on experience. Trying to learn the function defining the filter distribution could face identical drawbacks caused by searching any arbitrary function. One solution is to reduce the set of possible functions (e.g. by searching only linear functions). We go further by reducing the number of points that the searched function should fit and constraining the number of FLOPs that the model with the new distribution should count. However, the remaining space is still big enough to require a considerable computational budget. Alternatively, we propose a small set of different but straightforward distributions that can be exhaustively evaluated.

While our templates are heuristically chosen, they open avenues for insight and alternatives for designing models. While searching the entire space may sound intractable, the small number of proposed templates offer a fast and iteration-less alternative to architecture search or optimisation, which can deliver better performance straight away.

3.5 Filter Distribution Templates

While most of the neural network architectures show an incremental distribution of filters, recent pruning methods such as [64, 104], have yielded different filter distribution patterns emerging when reducing models like VGG that defy the notion of pyramidal design as the best distribution for a model. These results are a motivational insight into what other distributions can and should be considered when designing models. On one side, the combinatorial space of distributions makes this a challenging exploration. On the other, however, it emphasises the need to pursue such exploration if resulting networks can make gains in accuracy and overall performance.

In this work, rather than attempting to find the optimal filter distribution with expensive automatic pruning or growing techniques, we propose first adjusting the filters of a convolutional network model via a small number of pre-defined templates. These templates, such as those depicted in Figure 3.1, are inspired by existing models that have already been found to perform well and thus candidates that could be beneficial for model performance improvement beyond the number of operations. In addition, performance criteria such as accuracy, memory footprint and inference time are arguably as important as the number of operations required.

In particular, we adopt as one template a distribution with a fixed number of filters as with the original neocognitron design, but also other templates inspired by the patterns discovered in [64]. Some of the proposed distributions can be found in different blocks from the resulting ResNet101 model: 1) filters agglomerate in the centre, and 2) filters

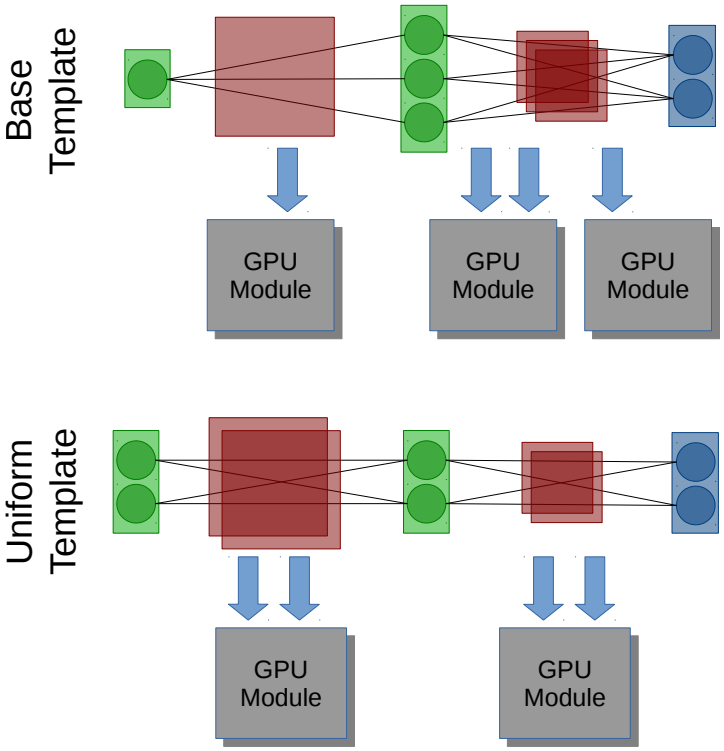


Figure 3.2: A toy example to show how two different templates with the same number of filters produce a variety of effects in parameters, memory, inference time and FLOPs. Layers (rectangles) contain, in total, an equal number of filters (circles) for both templates. Lines between filters represent parameters. Red squares are by-channel feature maps that reside in memory jointly with parameters. Flops are produced by shifting filters along with feature maps. Inference time is affected by flops and number of transfers, indicated by blue arrows and limited to two simultaneously, between memory and GPU modules. The diagram assumes filters of equal sizes and pooling between layers. Differences are scaled up in real models, counting thousand of filters.

are reduced in the centre of the block. In [104, 208] it is also shown a pattern with more filters in the centre of a VGG model. We define the templates we use in this work based on these observations.

Different distributions with the same number of filters can lead to different parameters (e.g. weights) and different memory or computational requirements (e.g. GPU modules). In the toy example in Figure 3.2, both models have the same number of filters, whereas the one on the right has fewer parameters and fewer compute requirements at the cost of more memory footprint. This example highlights the compromises that filter distributions can offer for the design and operation of network models.

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 $l \in \{1,...,D\}$. $D+1$ is the final classification

layer. The total number of filters that can be redistributed is given by

$$F = \sum_{l=1}^{D} f_l. \tag{3.1}$$

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

The number of filters in the final layer $D+1$ depends on the task and remains the same for all the templates. Therefore, it is not taken into account for computing the number of filters to redistribute. For architectures composed of blocks (e.g. Inception) we consider blocks as single layers and keep the number of filters within a block the same. As a result, a final Inception module marked with f_l filters is set to that number of filters in each layer inside the module.

3.5.1 Uniform Template

The most simple distribution to evaluate is, as the original Neocognitron, a uniform distribution of filters. Computing the number of filters in an uniform distribution is straightforward, the new number in each layer is given by

$$f'_l = F/D \quad \forall l \in \{1,...,D\}. \tag{3.2}$$

3.5.2 Reverse Template

Another straightforward transformation for the filter distribution adopted in this work is reversing the number of filters in every layer. Our final model with this template is defined by the filters

$$f'_l = f_{D-l+1} \forall l \in \{1,...,D\}. \tag{3.3}$$

3.5.3 Quadratic Template

This distribution is characterised by a quadratic equation $f'_l = al^2 + bl + c$ and consequently, has a parabolic shape with the vertex in the middle layer. We set this layer to the minimal number of filters in the base model

$$f_{min} = min(f_l) \quad l \in \{1,...,D\} \tag{3.4}$$

so, the new number of filters in the middle layer is described by

$$f'_{D/2} = f_{min}. \tag{3.5}$$

Also, we find the maximum value in both the initial and final convolutional layers, thus

$$f'_1 = f'_D. \tag{3.6}$$

To compute the new number of filters in each layer we solve the system of three linear equations given by *a)* the restriction of the total number of filters in equation 3.1

$$\sum_{l=1}^{D} \left(f'_l\right) = \sum_{l=1}^{D} \left(al^2 + bl + c\right) = F, \tag{3.7}$$

that can be reduced to

$$\left(\frac{D^3}{3} + \frac{D^2}{2} + \frac{D}{6}\right)a + \left(\frac{D^2}{2} + \frac{D}{2}\right)b + Dc = F, \tag{3.8}$$

b) the equation 3.5 produced by the value in the vertex

$$f'_{D/2} = \left(\frac{D}{2}\right)^2 a + \frac{D}{2}b + c = f_{min} \tag{3.9}$$

and *c)* the equality from the maximum values in equation 3.6, which reduces to

$$(D^2 - 1)a + (D - 1)b = 0. \tag{3.10}$$

3.5.4 Negative Quadratic Template

It is a parabola with the vertex in a maximum, that is, a negative quadratic curve. The equation is the same as the previous template, but the constraints change. Instead of defining a value in the vertex, f'_l at the initial and final convolutional layers are set to the minimal number of filters in the base model

$$f'_l = f_{min} \quad l \in \{1, D\}. \tag{3.11}$$

The number of filters in each layer is computed again with a system of equations specified by the restriction of the total number of filters as in the quadratic template (equation 3.8), and the two points already known in the first and last convolutional layers (equation 3.11) defined by

$$a + b + c = f_{min} \tag{3.12}$$

and

$$D^2 a + Db + c = f_{min}. \tag{3.13}$$

3.6 Model Comparison With Similar Neurons

This section investigates the effects of applying different templates to the distribution of kernels in convolutional neural network models (VGG, ResNet, Inception and MobileNet). We compare models based on size, memory, and speed in three popular datasets for classification tasks with models produced from the same architectures with the same number of neurons.

3.6.1 Datasets and Models

We trained over three datasets traditionally used for convolutional network evaluation: CIFAR-10, CIFAR-100 [99] and Tiny-Imagenet [103]. The first two datasets contain sets of 50,000 and 10,000 colour images for train and validation, respectively, with a resolution of 32x32. Tiny-Imagenet is a reduced version of the original Imagenet dataset with only 200 classes and images with 64 x 64 pixels resolution.

We evaluate some of the most popular CNN models: VGG [183], ResNet [73], Inception [187] and MobileNet [82]; which represent some of the highest performing CNNs on the ImageNet challenge in previous years [170].

3.6.2 Implementation Details

Experiments have models fed with images with the standard augmentation techniques of padding, random cropping and horizontal flipping. Our experiments were run in an NVidia Titan X Pascal 12GB GPU adjusting the batch size to 128. All convolutional models, with and without templates, were trained for 160 epochs using the same conditions. Therefore, there is some margin for improving accuracy for each distribution by performing individual hyperparameter [111, 137]. We used stochastic gradient descent (SGD) with weight decay of 1e-4, momentum of 0.9 and a scheduled learning rate starting in 0.1 for the first 80 epochs, 0.01 for the successive 40 epochs, and 0.001 for the remaining epochs.

3.6.3 Template Effect Over Baseline Models

We conducted an experiment to test our proposed templates on the selected architectures. Table 3.1 shows VGG, Inception and MobileNet accuracies improving in all datasets when templates are applied. Being complex architectures, ResNet and Inception present the highest accuracy in general. A surprising finding is that in both models difference

Table 3.1: Model performances with the original distribution and four templates with the same number of filters evaluated on CIFAR-10, CIFAR-100 and Tiny-Imagenet datasets. After filter redistribution models surpass the base accuracy. Results show average of three repetitions.

		Redistribution Templates			
Model	Base	Rev Base	Unif	Quad	Neg Quad
CIFAR-10					
VGG-19	93.52 ± 0.2029	**94.40 ± 0.1609**	94.24 ± 0.3080	94.18 ± 0.1501	94.21± 0.1096
ResNet-50	94.70 ± 0.0907	95.17 ± 0.2828	95.08 ± 0.2042	94.41 ± 0.0424	**95.23 ± 0.2514**
Inception	94.84 ± 0.2787	94.60 ± 0.3089	94.82 ± 0.2300	**94.86 ± 0.3031**	94.77 ± 0.2145
MobileNet	89.52 ± 3.2479	**91.35 ± 3.5317**	91.28 ± 3.1284	89.98 ± 2.9819	91.04 ± 3.2207
CIFAR-100					
VGG-19	71.92 ± 0.5519	**74.65 ± 0.4091**	74.03 ± 0.8072	73.55 ± 0.5100	74.05 ± 0.6091
ResNet-50	**77.09 ± 1.2008**	74.80 ± 0.2969	76.65 ± 1.2796	75.71 ± 0.4808	76.76 ± 0.7595
Inception	78.03 ± 0.7239	77.78 ± 0.8005	**78.12 ± 0.4454**	77.67 ± 0.8310	76.65 ± 0.1184
MobileNet	65.08 ± 3.7613	66.39 ± 7.7449	**68.71 ± 5.2817**	63.89 ± 3.8005	67.05 ± 7.5938
Tiny-Imagenet					
VGG-19	54.62 ± 1.5897	57.73 ± 0.7783	56.68 ± 1.2042	54.73 ± 0.9091	**59.50 ± 1.4912**
ResNet-50	**61.52 ± 0.9634**	53.67 ± 5.6167	60.97 ± 0.5379	59.77 ± 0.7864	60.12 ± 0.7239
Inception	54.80 ± 0.9814	55.24 ± 0.7143	55.78 ± 0.4315	54.97 ± 0.0500	**55.87 ± 0.4935**
MobileNet	56.29 ± 2.0687	51.40 ± 0.5246	**58.11 ± 2.0120**	53.37 ± 1.1472	55.76 ± 2.4712

in accuracy between templates is less than 2.3% despite the drastic modifications that models are suffering after the change of filter distribution. Models that share a sequential classical architecture, such as VGG and MobileNet, show a better improvement when using templates in Tiny-Imagenet. A remarkable accuracy improvement of 4.88 percentage points is achieved in VGG.

When analysing resource consumption (Table 3.2), we find models are affected differently with each template and model. *Reverse-Base*, *Uniform* and *Quadratic* templates show some reductions in the number of parameters, while *Negative Quadratic* template reduces the memory usage. Inference Time is affected negatively for most of the templates. This is an expected result as original models are designed to perform well in the GPU. Inception model shows an improvement in speed with reductions of 14% over inference time with respect to the base model while maintaining comparable accuracy. ResNet can reduce inference time by 49% at the cost of having slightly less accuracy than the base model.

Table 3.2: Resource consumption of selected models when applying our templates keeping the same number of filters evaluated on CIFAR-10 dataset. Models are normally optimised to fast GPU operation, therefore the original base distribution has a good effect in speed but the redistribution of filters induced by our templates makes models capabilities improve on the other metrics. Memory footprint reported by CUDA.

			Redistribution Templates			
Resource	Model	Base	Reverse Base	Uniform	Quadratic	Neg Quad
Parameters (Millions)	VGG-19	20.0	20.0	16.0	**15.8**	20.0
	ResNet-50	23.5	23.1	**12.9**	19.0	33.0
	Inception	**6.2**	6.7	**6.2**	7.2	7.0
	MobileNet	3.2	**2.2**	**2.2**	3.2	2.4
Memory Footprint (GB/batch)	VGG-19	**1.3**	2.6	4.4	2.0	1.4
	ResNet-50	3.1	11.5	4.1	7.9	**3.0**
	Inception	**1.5**	3.1	1.7	2.2	1.6
	MobileNet	2.5	5.1	1.5	6.0	**1.0**
Inference Time (ms/batch)	VGG-19	**3.0**	8.2	5.3	7.5	7.3
	ResNet-50	46.4	61.0	**23.4**	59.0	47.6
	Inception	28.5	54.9	34.3	25.2	**24.3**
	MobileNet	**3.8**	6.8	4.3	7.4	4.9
FLOPs (Millions)	VGG-19	**399**	4296	2006	3060	1058
	ResNet-50	**1304**	16326	3515	9710	4741
	Inception	**1544**	3734	2263	3469	2093
	MobileNet	**47**	760	277	797	132

Table 3.3: Resource consumption of selected models when applying our templates keeping the same number of filters evaluated on Tiny-Imagenet dataset. Best improvements on metrics differ from those obtained on CIFAR-10 suggesting there is not universal best template. Memory footprint reported by CUDA.

			Redistribution Templates			
Resource	Model	Base	Reverse Base	Uniform	Quadratic	Neg Quad
Parameters (Millions)	VGG-19	25.0	20.6	**19.3**	20.7	20.6
	ResNet-50	23.9	23.1	**13.0**	19.3	33.0
	Inception	19.2	**10.0**	12.7	18.7	10.1
	MobileNet	3.4	**2.4**	**2.4**	3.3	2.6
Memory Footprint (GB/batch)	VGG-19	**1.5**	10.0	4.8	6.8	3.8
	ResNet-50	**5.0**	10.1	9.6	7.5	9.8
	Inception	**5.8**	10.8	6.7	8.6	5.9
	MobileNet	2.5	5.1	5.9	4.8	**1.9**
Inference Time (ms/batch)	VGG-19	4.9	4.1	4.2	4.6	**3.5**
	ResNet-50	13.3	12.8	12.8	**11.0**	29.9
	Inception	24.0	21.4	28.3	**18.3**	31.4
	MobileNet	5.8	6.7	9.7	7.3	**5.3**
FLOPs (Millions)	VGG-19	**1716**	17215	8112	12360	4265
	ResNet-50	**335**	4157	908	2489	1195
	Inception	**1568**	3837	2279	3497	2100
	MobileNet	**51**	886	324	913	136

3.7 Template Effect With Similar Resources

It can be argued that models obtained with templates make use of more resources such as memory or number of operations in the GPU (reflected in the low inference speed). So, we formulated a second experiment that makes proportional changes in the models after applying the templates. We reduce the models and increase them to observe if the actual total number of filters is adequate for the task the model is performing or if the model accuracy could improve by adding more filters. Thus, we create curves for each template performing proportional reductions using a width multiplier with values of 1.6, 1.3, 1.0, 0.8, 0.5, 0.25, 0.1 and 0.05. These curves of reduction allow comparison under the same amount of resources and compare the use of resources under the same accuracy. The experiment also shows the level of reduction that our models can tolerate without a significant loss in accuracy for the evaluated datasets.

We add dashed lines to every plot to be used as a reference for the model with the original distribution and no reductions, which is the point where both vertical and horizontal dashed lines cross. In general, any arbitrary vertical line in the plot compares accuracy between models with the number of resources (parameters, memory or speed). On the other side, any arbitrary horizontal line compares the resources taken for each model under each template to produce similar accuracy.

3.7.1 Parameters Count

Evaluating a model's performance using accuracy and parameters is, by far, the default approach. We show models performances with these metrics in Figure 3.3. VGG and MobileNet models improve accuracy almost with any template in CIFAR-10 and CIFAR-100. By using reductions with width multipliers, the original accuracy can be reached with less than 25% of the original parameters in the two models. ResNet shows less improvement when compared with networks with similar resources, yet templates can reduce the model further before accuracy drops. Inception behaviour considering the same resources remains similar no matter the template used. In general, for this test, the uniform template seems to get the best parameter efficiency for all the models.

3.7.2 Memory Footprint

Results in Table 3.2 have shown that comparing parameters is not the best option for practical implementations. Models with a small number of parameters are not necessarily

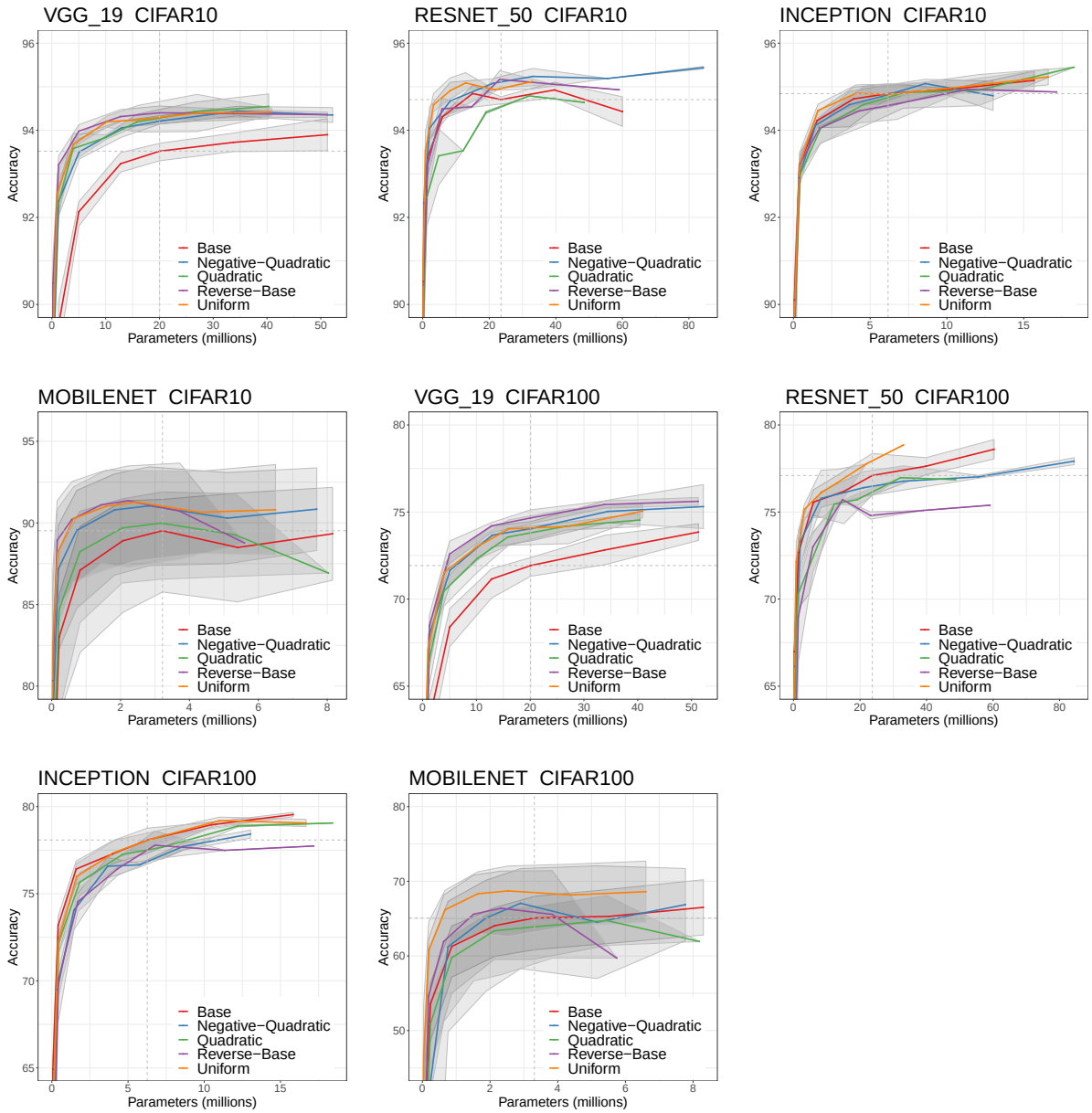


Figure 3.3: Average Accuracy versus Parameters in CIFAR-10 and CIFAR-100 datasets using templates with VGG, ResNet, Inception and MobileNet. Curves are created by changing models using width multiplier scaling. An arbitrary vertical line in the plot compares templates effects using the same amount of parameters. An arbitrary horizontal line compares parameters for reaching the same accuracy. Shadows cover maximum and minimum of three repetitions. Dashed lines indicate base model accuracy and parameters.

related with a small memory footprint or greater speed, and more results are presented in Figure 3.4. We observe again that templates enhance VGG and MobileNet accuracy. More than 50% of memory consumption can be reduced in both models while producing the same accuracy. With this metric, ResNet and Inception improve slightly in CIFAR-10 with the Negative-Quadratic template but perform lower with the rest of the templates. We attribute the lower memory efficiency the fact that in all the templates except *Negative-Quadratic*, the number of filters is increased in the initial layers. At these layers, the size of feature maps produced for each filter is more significant and, therefore, more memory costly.

3.7.3 Inference Time

One final comparison also crucial for practical issues is inference time. Our experiments show the patter of improvement for VGG and MobileNet and degradation of inference time when adopting templates in ResNet and Inception (See Figure 3.5 in Appendix). In particular, Inception shows an improvement with the Negative-Quadratic template in CIFAR-10.

Looking at results in inference time, it seems unpromising to use templates. However, we can take a different perspective. It is possible to obtain models with better accuracy by sacrificing inference speed. This could be an undesirable decision, but in practice, it is frequently chosen. It is clearly reflected in the inference time between different original models. For example, by using ResNet the accuracy improves compared to the obtained by VGG in the two datasets tested, but at the cost of increasing the time for inference. On the contrary, looking for speed, enhancement MobileNet has sacrificed accuracy. In this sense, our templates are still competitive compared to searching for a totally different model to improve accuracy.

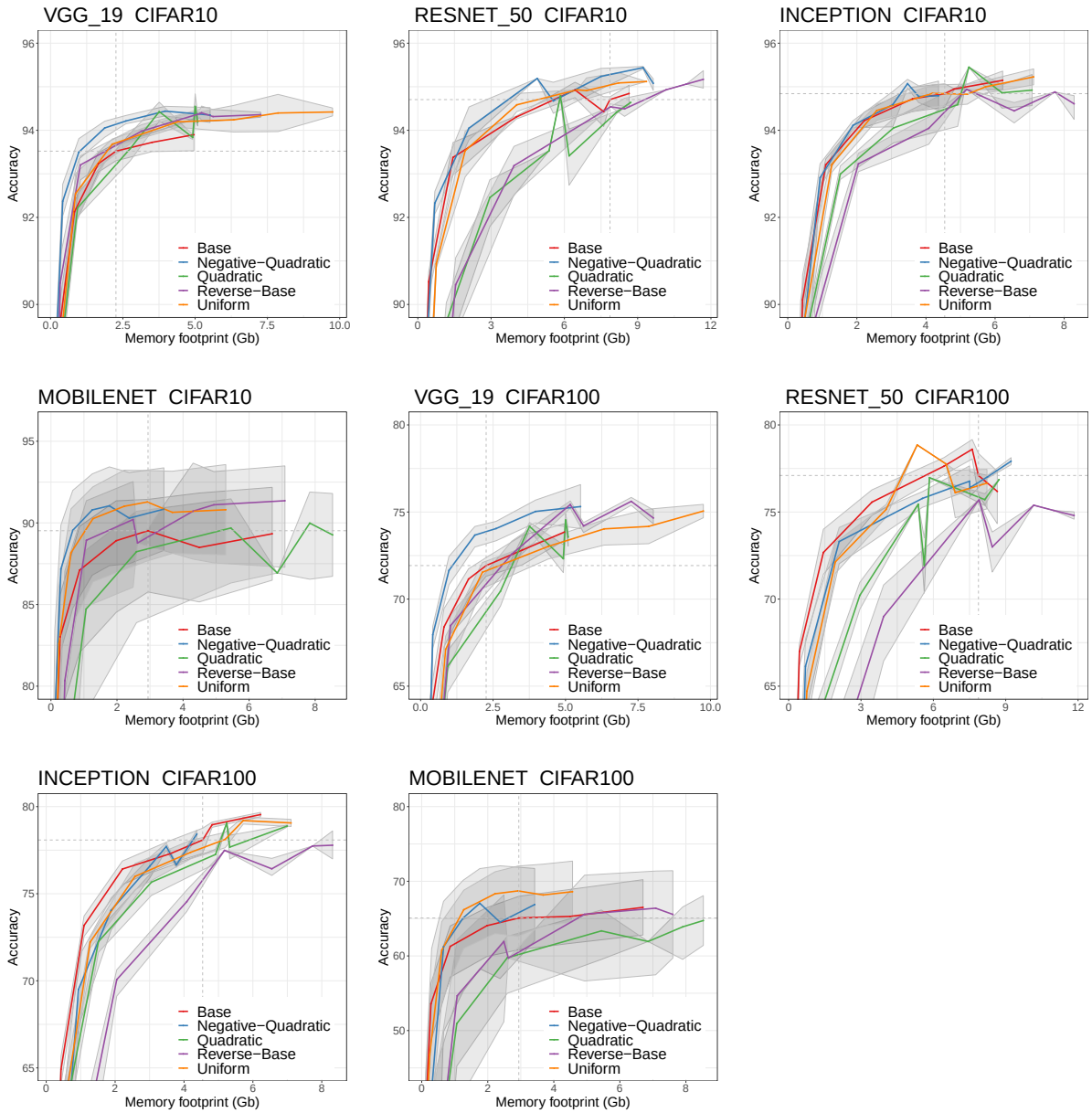


Figure 3.4: Accuracy versus Memory Footprint in CIFAR-10 and CIFAR-100 datasets using templates with VGG, ResNet, Inception and MobileNet. Curves are created by reducing models using width multiplier scaling. An arbitrary vertical line in the plot compares templates effects in accuracy using the same amount of memory. An arbitrary horizontal line compares memory consumption for reaching the same accuracy. Shadows cover maximum and minimum accuracy of three repetitions. Dashed lines indicate base model accuracy and memory footprint.

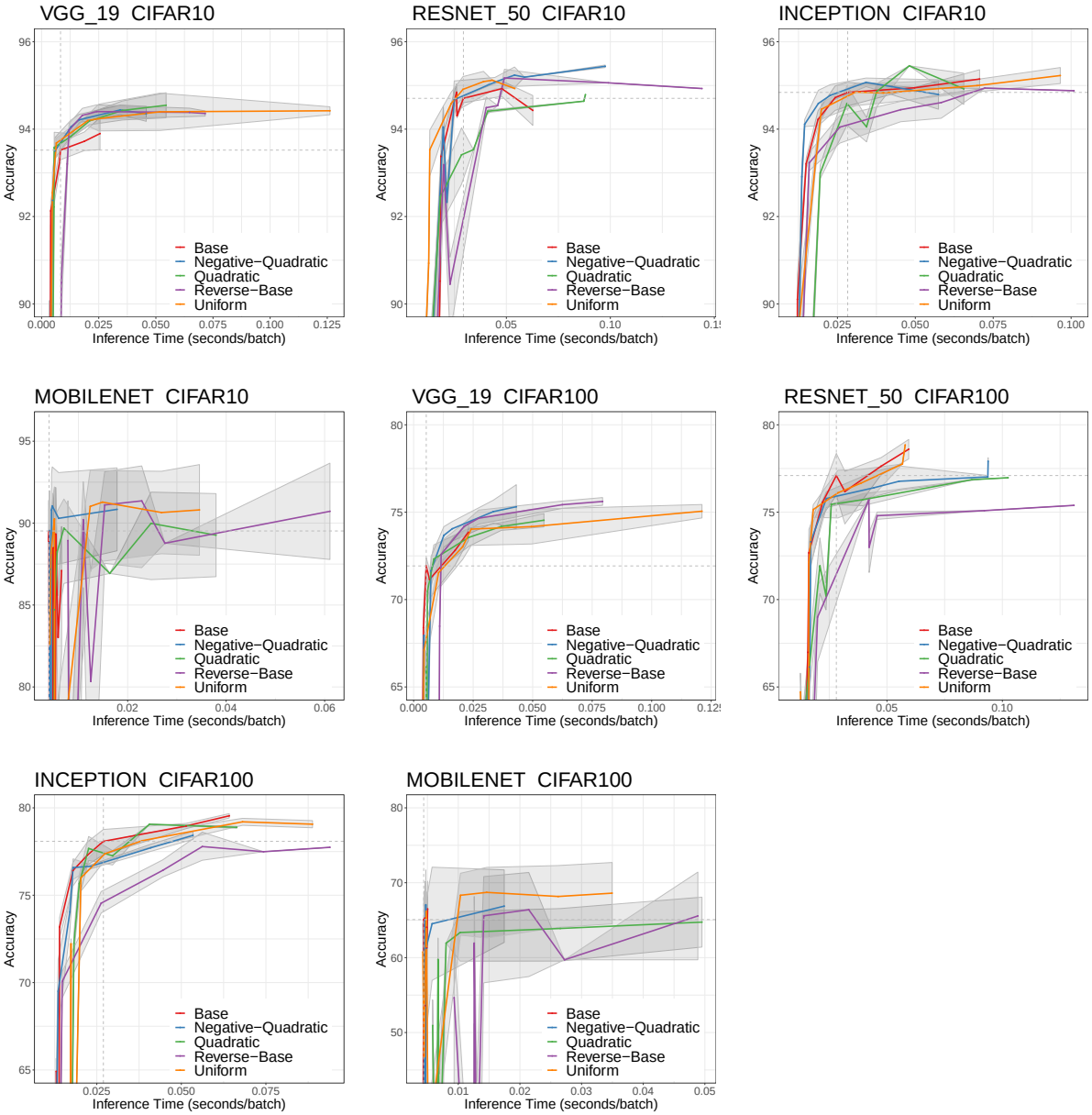


Figure 3.5: Accuracy versus Inference Time in CIFAR-10 and CIFAR-100 datasets using templates with VGG, ResNet, Inception and MobileNet. Curves are created by reducing models using width multiplier scaling. An arbitrary vertical line in the plot compares templates effects in accuracy between models with same inference speed. An arbitrary horizontal line compares inference time of models with the same accuracy. Grey errors bands denote maximum and minimum of three repetitions.

3.8 Conclusion

The most common design of convolutional neural networks when choosing the distribution of the number of filters is to start with a few and then increase the number in deeper layers. We challenged this design by evaluating some architectures with various distributions on the CIFAR and Tiny-Imagenet datasets. Our results suggest that this pyramidal distribution is not necessarily the best option for obtaining the highest accuracy or parameter efficiency.

Our experiments show that models with the same number of filters but different distributions produced by our templates improve accuracy by up to 4.8 points for some model-task pairs. Moreover, in terms of resource consumption, templates can obtain a competitive accuracy compared to the original models using fewer resources with up to 56% fewer parameters and a memory footprint up to 60% smaller. Results also reveal an interesting behaviour in evaluated models: a strong resilience to changes in filter distribution. After modifying models with templates, variation in accuracy for all models is less than 5% despite the considerable modifications in filter distributions and, therefore, in the original filter design.

It is important to notice that a reduced number of parameters does not correspond to low consumption of memory, not even a small inference time. Some of the causes are the difference in feature map resolution for filters in different layers, the need to keep early feature maps in memory for late layers and the restrictions for improving parallelisation in the computational graph of the model.

CHAPTER 4

BEYOND CLASSIFICATION TASKS: TESTING TEMPLATES IN OTHER DOMAINS

Image Classification experiments in the previous chapter showed the advantage of templates to find models with better accuracy by redistributing the same neurons or filters in unexplored manners. The changes produce new models with different resource demands than the original models. Compared with similar resources, these new distributions, called templates, are more effective than the standard incremental distribution present in most neural network models.

This chapter examines whether neural network models in other domains are amenable to being improved by using the templates. Previous results showed that templates help reduce parameters in final models. Therefore the chapter compares templates against pruning methods. Furthermore, it describes and evaluates how templates can be used jointly with CNS methods such as MorphNet, to produce more efficient models.

4.1 Introduction

We have found in chapter 3 that template resulting CNNs are more advantageous than the original models from which they were obtained, particularly for the task of image classification. We designed experiments in this chapter to explore the scope where templates can help to improve existing models. We tested the templates using images as input in two new domains. Firstly, we evaluated templates in a regression task, which is

generally considered more difficult than classification. Secondly, we evaluated templates in image super-resolution. Many models in the latter task possess a very particular feature: they do not contain subsampling layers. Therefore the feature maps remain of the same resolution along with the architecture, and then the effects of redistributing templates can differ from other domains.

The proposed distributions in chapter 3 also showed reductions in parameters and memory usage. We argue that using templates can be seen as a tool for obtaining better performances. On the other hand, they can also be thought of as a tool to reduce the resource consumption of models. With this in mind, we compared the accuracy and resources of models obtained from templates with models obtained from three popular pruning methods. Nonetheless, our primary goal is to see if pruning methods converge to the same results starting from different distributions. We also showed that the templates could work jointly with pruning methods to benefit further. Finally, we presented in this chapter an evaluation of the templates working with MorphNet, a method searching for the best number of filters of an existing architecture.

We investigated the effects of applying different templates to the distribution of filters in some well known convolutional neural network models (VGG, PoseNet and MobileNetV1) for the mentioned domains. Where possible, the resulting models produced with templates have similar FLOPs to the original model from which they were obtained. This constraint facilitates further comparison of models on the basis of size, memory and speed.

4.2 Global Localisation

Classification is the task where a predictive model approximates a mapping function taking input values to predict discrete output values. Classification algorithms can process any type of input variables, either categorical or numeric, but the examples must be allocated into one of two or more classes. Regression tasks differ from classification tasks in the type of output variables. Instead of predicting a discrete class, a model performing regression predicts continuous values.

Neural networks have been used in both classification and regression tasks. However, it is believed that neural network models perform better in classification than in regression tasks for some types of domains [151], possibly because regression tasks require a more accurate prediction of output values.

Among many regression problems, one that has attracted plenty of attention is

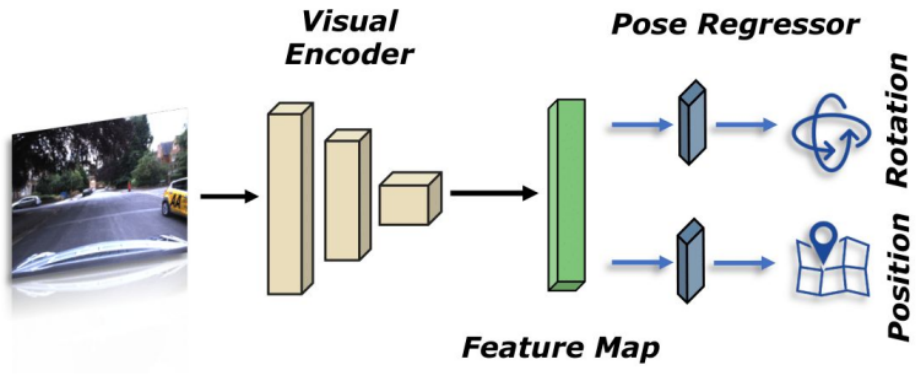


Figure 4.1: The typical design of architectures for localisation. The pattern was first presented in PoseNet. Image from [24].

localisation. For decades, localisation has been studied, and a range of complicated hand-designed models and algorithms have been devised [190]. The task's purpose is to provide an agent with its position (a continuous value) within the surrounding environment by perceiving scenes with sensors such as a camera.

Recently, researchers have been adopting data-driven methods to directly compute the camera's position by using only the input image. The first implementation of a neural network in global localisation was presented in the PoseNet paper [93].

Since then, researchers have proposed several improvements such as modelling uncertainty [92] and inducing temporal [33] and spatial coherence [77] but normally following a similar architectural design shown in Figure 4.1. The idea of PoseNet borrowing a model from classification and then modifying the last layers to produce the desired localisation values has been the trend in many of the subsequent works [24].

A singular model derived from PoseNet is the hourglass architecture [133]. It consists of a two-section network that first encodes coarse information and then extracts fine-grained details. The network structure follows an increasing number of filters in the first half of layers and then a decreasing pattern in the other half, resembling one of our templates.

The neural networks used in localisation tasks create an implicit map during training. They are fed with the camera pose and the image captured in that position creating a hidden representation of the world. We decided to test the templates in this task to show if this implicit representation is significatively different when we change the distribution of the filters. In the experiments with classification, we found that the models are affected differently by each template in their parameter count. If we take the parameters as the place the implicit map is stored, one could expect that templates

Figure 4.2: Examples of the datasets used in the experiment. The first row corresponds to samples from 7-Scenes dataset. The second row shows examples taken from the Cambridge Landmarks dataset. The second dataset presents much more variations in spatial dimension and illumination conditions.

producing networks with fewer parameters are less capable of performing well in this localisation task. However, some works on map compression have shown that reductions in the size of neural networks do not significantly affect the localisation error [35]. As if the network was capable of automatically removing the less useful visual features to build the map as made in hand-crafted descriptor approaches [34].

4.2.1 Models and Datasets

We create an experiment using three classical neural networks with several templates. We took the PoseNet network as the first option. It was originally trained end-to-end to compute the 6-DoF camera pose from a single RGB image. The architecture is based mainly on GoogleNet [187]. Still, it changes the last softmax layer with a fully connected layer to output the global pose, consisting of position and orientation vector. We chose VGG and MobileNet as the second and third models to be tested, adapting the last layer in the same fashion as in PoseNet.

All CNNs were trained over three 2D image datasets traditionally used for visual metric localisation evaluation: the 7-scenes dataset[60] composed by small indoor settings with very few illumination condition variations; and the Cambridge Landmarks[93] with large scale outdoor scenes under varied lighting conditions (see samples in Figure 4.2). This selection provides our experiment with diverse environments and differences in the number and density of samples. The 7-scenes dataset contains scenes with between 500 and 1000 frames recorded from a Kinect RGB-D camera using a resolution of 640×480 pixels. The Cambridge Landmark dataset consists of outdoor urban scenes populated with people and vehicles. Both datasets generated the camera pose with structure from motion (SfM) methods [178].

4.2.2 Implementation Details

In principle, PoseNet produces a 6-dimensional pose, but given that the orientation is provided in quaternions, the final pose requires a 7-element vector. A drawback in PoseNet is that the authors introduced a new hyperparameter β to tune the relevance of both orientation ($\mathbf{q}$) and position ($\mathbf{x}$) within the loss function given by

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|$$

In this section experiments, we decided only to predict the position to avoid tuning the parameter, which could have different optimal values depending on the template to be applied.

Our experiments in localisation follow a similar procedure to the classification experiments. They run in an NVidia Titan X Pascal 12GB GPU adjusting the batch size to 16. With and without templates, all models were trained for 160 epochs using stochastic gradient descent with weight decay of 1e-4, momentum of 0.9 and a learning rate of 0.1. Localisation entirely relies upon the camera viewpoint, so augmentation was limited to colour space transformations.

Our implementation is comparable to that in PoseNet with random initialisation, which is lower than that using ImageNet and Places pretraining. Moreover, we used the whole training set instead of fractions of the data like in the PoseNet paper because we performed from scratch training.

4.2.3 Results

These experiments evaluate models using the mean absolute error (MAE). According to the authors of 7-Scenes and Cambridge Landmarks datasets, the units for the camera pose are given in metres. We present in Table 4.1 the results of evaluating VGG19 on the 7-Scenes datasets. We observe that in more than half of the scenes tested, the original distribution gets the lower error while the *negative-quadratic* template obtains the lower error in two cases. The difference between the best and the second model is less than 10% of the best value. The PoseNet architecture obtains lower errors (Table 4.2 compared to those of VGG19. Besides, the templates produce the best models for five of the seven scenes, mainly made with the *negative-quadratic* distribution.

For the Cambridge Landmark dataset, templates produce a similar pattern with VGG19 and PoseNet models but with bigger improvements. For VGG19, Table 4.3 shows that the base incremental distribution of filters is better in only three scenes while

Table 4.1: MAE for 7-Scenes dataset for the VGG19 model with its original filter distribution and four templates. Table shows one-repetition results.

7-Scenes (VGG19)		Redistribution Templates			
Scene	Base	Reverse Base	Uniform	Quadratic	Neg Quad
chess	**0.2517**	0.2941	0.2936	0.2719	0.2773
fire	**0.3672**	0.4406	0.3958	0.4153	0.4057
heads	0.2178	0.2608	0.2190	0.3383	**0.2013**
office	**0.3522**	0.3821	0.3681	0.3572	0.3652
pumpkin	0.4423	0.5036	0.4472	0.4589	**0.4220**
redkitchen	**0.4303**	0.5131	0.5078	0.4859	0.4733
stairs	0.4181	0.4852	**0.4038**	0.4632	0.4295

Table 4.2: MAE for 7-Scenes dataset for the PoseNet model with its original filter distribution and four templates. Table shows one-repetition results.

7-Scenes (PoseNet)		Redistribution Templates			
Scene	Base	Reverse Base	Uniform	Quadratic	Neg Quad
chess	**0.1868**	0.2185	0.1979	0.1916	0.1975
fire	0.3120	0.3672	0.3179	0.3293	**0.3117**
heads	0.1986	0.2608	0.2056	0.1937	**0.1810**
office	0.3256	0.3471	0.3218	0.3220	**0.3135**
pumpkin	0.3202	0.3373	0.3086	**0.3069**	0.3247
redkitchen	0.3780	0.4587	0.3792	0.3884	**0.3746**
stairs	**0.3741**	0.4711	0.3822	0.4236	0.3787

for PoseNet (Table 4.4), the base model only performs better in one scene. Again, the *negative-quadratic* distribution seems like the best suited for the dataset.

Localisation is expected to be performed for a mobile agent. While the original PoseNet is considered a more efficient architecture than VGG, it is clear that its resource consumption is enough to surpass the capacity of a mobile system. We extend our experiment with the Cambridge Landmarks dataset to include MobileNet, a more appropriate model for mobile applications optimised to that end. MobileNet performance is shown in Table 4.5. The model obtains higher error than the performance-oriented PoseNet and VGG. The differences in the three models' results are compatible with the reported in the literature for other datasets such as ImageNet. For MobileNet, the most successful model is found with the *uniform* template with the lowest error for four out of six scenes.

Comparing performances among models gives a partial perspective of the model's capacity. For this reason, we also show in Table 4.6 the resources required for the templates with the VGG19, PoseNet and MobileNet models. We found that, while memory,

Table 4.3: MAE for Cambridge Landmark dataset for the VGG19 model with its original filter distribution and four templates. Table shows one-repetition results.

Cambridge Landmarks (VGG19)		Redistribution Templates			
Scene	Base	Reverse Base	Uniform	Quadratic	Neg Quad
KingsCollege	8.0613	**7.5204**	8.3575	8.8226	7.6481
OldHospital	**5.2201**	5.8798	5.4487	5.9931	5.5314
ShopFacade	**2.8282**	2.9202	4.3013	3.7109	3.3459
StMarysChurch	**7.0244**	8.5696	7.9538	8.4453	7.4600
Street	80.8015	83.3418	85.8898	86.5996	**77.5769**
GreatCourt	24.8433	25.5597	26.2833	**24.6635**	26.6305

Table 4.4: MAE for Cambridge Landmark dataset for the PoseNet model with its original filter distribution and four templates. Table shows one-repetition results.

Cambridge Landmarks (PoseNet)		Redistribution Templates			
Scene	Base	Reverse Base	Uniform	Quadratic	Neg Quad
KingsCollege	6.8787	9.1456	6.0554	6.2638	**5.4428**
OldHospital	5.0916	6.3190	**4.6366**	4.6476	4.8044
ShopFacade	3.2472	3.3165	2.9183	3.1887	**2.7564**
StMarysChurch	8.1209	11.1730	7.9718	8.6005	**7.9059**
Street	72.0053	131.9090	71.8444	78.5514	**71.2381**
GreatCourt	**23.9980**	25.2407	25.6946	24.6157	25.0797

Table 4.5: MAE for Cambridge Landmark dataset for the MobileNet model with its original filter distribution and four templates. Table shows one-repetition results.

Cambridge Landmarks (MobileNet)		Redistribution Templates			
Scene	Base	Reverse Base	Uniform	Quadratic	Neg Quad
KingsCollege	11.2847	12.4568	**10.2853**	13.5246	12.4949
OldHospital	**7.7933**	14.6661	8.5886	13.0720	9.2989
ShopFacade	**5.2755**	7.5576	5.6626	6.7993	6.1695
StMarysChurch	12.3526	15.3554	**13.3396**	16.4533	13.3917
Street	99.4402	100.3203	**97.9171**	108.3730	101.9091
GreatCourt	27.1893	28.5328	**25.7820**	28.8231	26.7306

Table 4.6: Parameters, memory, inference time and FLOPs for selected models when applying our templates keeping the same number of filters in localisation tasks.

			Redistribution Templates			
Resource	Model	Base	Reverse Base	Uniform	Quadratic	Neg Quad
Parameters	VGG19	20.33	**1.80**	3.30	2.02	7.19
(Millions)	PoseNet	6.16	**3.28**	4.99	3.48	6.29
	MobileNet	3.21	**0.12**	0.33	0.17	0.80
Memory	VGG19	0.44	0.53	0.55	0.54	**0.43**
Footprint	PoseNet	5.02	5.57	5.08	4.77	**4.62**
(GB/batch)	MobileNet	**0.11**	0.16	0.15	0.15	0.12
Inference	VGG19	**0.053**	0.068	0.065	0.056	0.057
Time	PoseNet	0.614	0.763	0.613	**0.612**	0.614
(ms/batch)	MobileNet	**0.012**	0.025	0.023	0.025	0.017
FLOPs	VGG19	19.55	**18.99**	19.86	18.45	18.63
(Gigas)	PoseNet	99.88	**92.29**	99.92	95.13	98.23
	MobileNet	0.57	**0.54**	0.54	0.56	0.53

inference time and FLOPs remains similar with all templates, using these different distributions reduce original parameters significatively for all models. Furthermore, the reduction allows storing a modified PoseNet model (*reverse* template), with lower prediction error on average, in the same space of an original MobileNet.

4.3 Single Image Super-Resolution

The technique of recovering high-resolution (HR) images from low-resolution (LR) photos is known as image super-resolution (SR) [201]. It is a subset of image processing techniques used in computer vision and image processing with a vast scope of applications, including enhancing medical, satellite, surveillance, and astronomical imaging.

The super-resolution task is particularly interesting for our work because of its differences in architecture design. Firstly, models in SR do not reduce the size of the feature maps in deeper stages of the architecture, but they keep the same size. Secondly, most models observe not the incremental design found in almost all computer vision tasks but one that follows a uniform distribution, keeping constant the number of filters for all layers.

The first work proposing to use a CNN to produce high-resolution images from low-resolution ones is presented in [43]. The model named Super-Resolution Convolutional Neural Network (SRCNN) is a shallow network with two layers using double of filters

in the first (64) than in the second layer (32). The distribution is contrary to what is the typical neural network building pattern in other computer vision tasks consisting of doubling filters in deeper layers. The justification behind this selection on the number of filters relies on the analogy of sparse-coding based SR methods and SRCNN, so the authors assume the model will "rely more on the central part of the high resolution patch" [43] like typically happens in sparse-coding HR reconstruction. SRCNN uses large filter size of up to 9x9.

VDSR is a deep model following SRCNN and inspired by the VGG architecture [95]. The network requires a preprocessing step to grow the LR image to the desired final resolution using a classical interpolation technique. The model is composed of 20 layers of 64 channels produced by 3x3 filters. This is the first architecture adopting a constant number of filters in super-resolution which later became the standard distribution for model design in this task. The authors realised that a deeper network produced a higher performance, but it was more challenging to train, taking longer training time and failing to converge. Their solution was to add a residual connection from the first to the final layer.

Other approaches propose using a generative adversarial network (GAN) framework for image super-resolution [108]. The trained generative network keeps the uniform distribution of filter of VSDR but changes VGG layers for residual modules [73]. The resulting network, called SRResNet, preserves high-frequency details producing perceptually satisfactory images at 4x resolution.

The EDSR model builds on SRResNet by transforming the multiple residual blocks [118]. The proposed block in EDSR does not include batch normalisation layers, resulting in improved performance and reduced memory consumption. Previous models were capable of resolving HR images at several scales (2x, 3x and 4x) but creating independent networks for each particular scale. The EDSR paper presents an extension of the network that takes advantage of the inter-scale correlation training. The authors propose a network with a single backbone sharing weights that connect with several branches producing each scale as described in Figure 4.3.

We decide to take the EDSR architecture to test our templates in the super-resolution task for several reasons: 1) it is one of the state-of-the-art models in SR, 2) its design is modular and easily adaptable to templates, 3) the backbone is built with the well known residual modules already in use in this work but it can fit other architectures, and 4) its training code is publicly available.

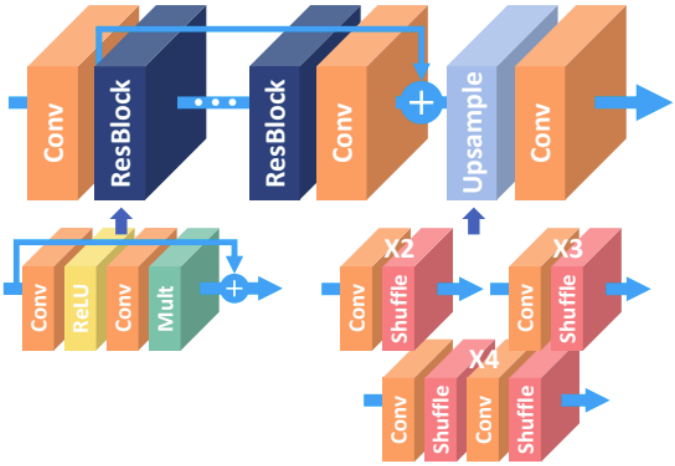


Figure 4.3: EDSR design for super-resolution. Residual modules count equal number of filters. The final branch depends on the HR scale of the final image but they can be combined to produce a multi-scale network. Image from [118].

4.3.1 Implementation Details

We used the code for training and evaluation of the models provided by [118]. We changed the number of filters in the EDSR backbone according to the definitions of the templates in section 3.5. Additionally, we tested a VGG-like backbone comparable to the VDSR architecture.

For evaluation, we use the popular DIV2K benchmark dataset [3] consisting of a set of 800 images for training, 100 for validation, and 100 for testing. As in previous works, we present performances on the validation dataset, given the test set is not available. Models in SR are typically trained with the mean squared error (MSE) as the loss function. However, they are evaluated with peak signal-to-noise ratio (PSNR) given by the equation

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

MAX_I is the maximum value for a pixel, usually 255. MSE is computed with pixel-by-pixel differences between the resolved and the ground-truth images. Although other metrics have been proposed considering the perceptual quality of the images [90], PSNR is still the most popular in the field of SR.

In the experiment, we do not follow the common practice in SR of only processing the image's luminance channel (Y channel in YCbCr colour space) and computing the rest of

Table 4.7: Performance at 4x and resource consumption of the EDSR method (VGG backbone) using templates tested on the DIV2K dataset. Table shows one-repetition results. *Unlike other tasks, original base template for super resolution uses an uniform distribution.

DIV2K (EDSR-VGG19)		Redistribution Templates			
Metric	Base*	Increasing	Reverse	Quadratic	Neg Quad
PNSR	28.623	28.516	28.404	**28.720**	28.495
Best epoch	191	172	**151**	199	171
Parameters (Millions)	**21.5**	25.9	24.9	21.6	24.8
Memory (GB)	7.36	7.71	7.18	**7.16**	7.70
GFLOPs	**264.5**	305.3	295.8	266.0	295.3
Inference time (ms)	**8.11**	8.98	8.71	8.32	8.82
File size (MB)	**86.1**	103.8	99.7	86.8	99.5

the channels with bicubic interpolation. Instead, we produce the full RGB channels with the network and evaluate PSNR in all of them.

For training, we use the default configuration of the EDSR paper with a 4x scale and a patch size of 192x192 for 300 epochs. We use an Adam optimiser with a learning rate of 0.0001.

4.3.2 Results

We developed two experiments for the task of SR. The first one tests two architectures, VGG and ResNet, as backbones of the EDSR framework using the different templates. The second experiment explores reductions in the number of filters using a width multiplier to find if the models are amenable to being compressed without significant loss in quality.

EDSR performance using VGG as backbone is presented in Table 4.7. The *quadratic* template shows benefits in the quality of the images evaluated with PSNR and lower memory footprint compared with the base model following a uniform distribution. Moreover, the models produced by templates seem to converge earlier.

The same results are found with the ResNet backbone shown in Table 4.8. The performances are higher compared to the VGG backbone which explains some of the improvements from VDSR to EDSR papers. The *negative-quadratic* template gives the best performance in this case, but this result is expected as the *negative-quadratic* template counts twice the parameters of the original distribution.

We show a resolved sample using VGG and ResNet backbones in Figure 4.4. It can be

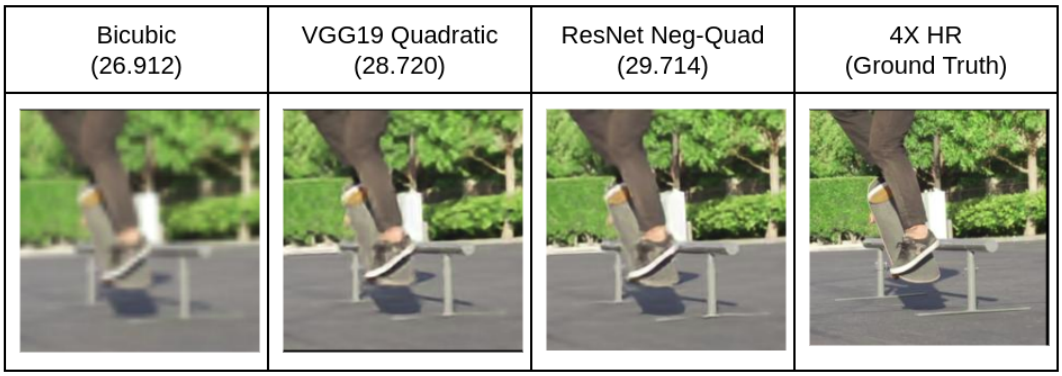


Figure 4.4: A resolved image using bicubic interpolation compared with the best performing models using templates with VGG and ResNet backbones in EDSR.

Table 4.8: Performance at 4x and resource consumption of the EDSR method (ResNet backbone) using templates tested on the DIV2K dataset. Table shows one-repetition results. *Unlike other tasks, original base template for super resolution uses an uniform distribution.

DIV2K (EDSR-ResNet)		Redistribution Templates			
Metric	Base*	Increasing	Reverse	Quadratic	Neg Quad
PNSR	29.709	29.712	29.669	29.693	**29.714**
Best epoch	299	296	**270**	271	285
Parameters (Millions)	**16.4**	26.3	26.8	22.3	35.9
Memory (GB)	4.86	5.44	5.53	**4.60**	6.97
GFLOPs	**217.9**	308.6	313.8	272.1	397.2
Inference time (ms)	**8.02**	9.84	9.89	8.89	12.73
File size (MB)	**65.9**	105.2	107.5	89.4	143.7

appreciated a significant difference between the classical method of bicubic interpolation and deep learning methods. However, no big gainings are obtained between different neural network models.

It has been found empirically in [43] that increasing the wide of the network leads to an increase in performance. The improvement, however, is small. Using a 4x width multiplier, the network improves in 0.34 points of PSNR. We show in tables 4.9 and 4.10 that the same effect happens in the opposite direction. We reduced VGG and ResNet backbones to 0.5x and 0.25x to find if there is a significant loss. We found that by using one quarter of the filters, the *quadratic* template reduces its performance by 0.38 points for the VGG backbone.

The loss using the *negative-quadratic* template with the ResNet backbone is lower: only 0.22 points of PSNR. Edge devices could benefit from this level of compression by

Table 4.9: Change of performance (PSNR) with different sizes of the EDSR method (VGG19 backbone) using templates tested on the DIV2K dataset. Models are reduced by using width multipliers. Table shows one-repetition results. *Unlike other tasks, original base template for super resolution uses an uniform distribution.

DIV2K (EDSR-VGG19)		Redistribution Templates			
Width multiplier	Base*	Increasing	Reverse	Quadratic	Neg Quad
1.00	28.623	28.516	28.404	**28.720**	28.495
0.50	28.464	**28.540**	28.088	28.484	28.329
0.25	28.291	28.249	27.874	**28.340**	28.089

Table 4.10: Change of performance (PSNR) with different sizes of the EDSR method (ResNet backbone) using templates tested on the DIV2K dataset. Models are reduced by using width multipliers. Table shows one-repetition results. *Unlike other tasks, original base template for super resolution uses an uniform distribution.

DIV2K (EDSR-ResNet)		Redistribution Templates			
Width multiplier	Base*	Increasing	Reverse	Quadratic	Neg Quad
1.00	29.709	29.712	29.669	29.693	**29.714**
0.50	29.539	29.580	29.539	29.479	**29.610**
0.25	29.416	29.449	29.394	29.339	**29.497**

receiving low-resolution images and producing high-resolution ones on the user side, with savings in the communication channel and the processing capacity of the device.

The experimental results show that models with the already in-use uniform distribution are more efficient in resource consumption for most metrics, including parameters, inference time and FLOPs. However, the *quadratic* template offers models with a smaller memory footprint. For VGG and ResNet backbones, templates produce models with more resource requirements in general. This disagrees with the reductions found in the classification and localisation tasks described before. This can be explained by the architectural design of the networks in the SR task that does not reduce the feature maps in the whole architecture. Templates indeed increase the total number of filters thus requiring more parameters and FLOPs. We found in this experiment that the incremental filter distribution is not the best option for the SR task. Yet variations in the performances and the resource demands are minimal across the templates. We believe this is intriguing, considering the drastic changes in filter distributions.

4.4 Templates and Neural Network Pruning

As we have shown in this and previous chapters, templates exhibit an ability to produce networks with reduced resource demands such as parameters in classification and localisation tasks. The reduction is conducted by simply redistributing the number of filters in an existing network before any training. The resource network reduction is analogous to that achieved by neural network pruning methods cited in section 2.2.1, so a required comparison has to be done. In this section, we compare the models resulting from templates against three different techniques for neural network pruning. The first two methods prune filters by evaluating their importance with different approaches. The first one, called Filter Pruning via Geometric Median (FPGM) remove filters close to the geometric median. The second one, known as Gate Decorator, uses Taylor expansions to evaluate the importance of filters on the loss function. The last one of the three methods resembles more a knowledge distillation [78] technique but at the level of layers. It tries to perform filter decomposition to reduce the number of parameters and FLOPs.

One common feature of these three methods is that they all fall in the category of methods where learning occurs late or after training. We consider the utilisation of the proposed templates as a pruning-at-initialisation method [53]. Templates do not evaluate the magnitude of weights nor their repercussions in the loss function. We agree with [124] that a simple reduction (or as in templates, a change) of filters and then training from scratch with randomly initialised weights is enough to compress a model without a noticeable decrease in performance.

4.4.1 Pruning Filters Based On Their Norm

Pruning weights with the smallest absolute magnitude is an obvious and surprisingly efficient criterion. This heuristic is widely applied in pruning methods doing structured and unstructured pruning. The common belief is that weights or filters with smaller norms are less critical for the network and then can be safely removed. The problem with that approach is that the remaining weights can not be used to reconstruct the values of those pruned, which might represent subtle features for the performing task.

Geometric Median [75] considers, unlike other methods, removing filters with values not close to zeroes but near to the geometric median of the set of filters in every layer. By pruning these filters, the remaining ones can represent the ones missing. As a result, reducing such filters doesn't significantly impact model performance.

Computing the geometric median is challenging. There is no algorithm nor explicit

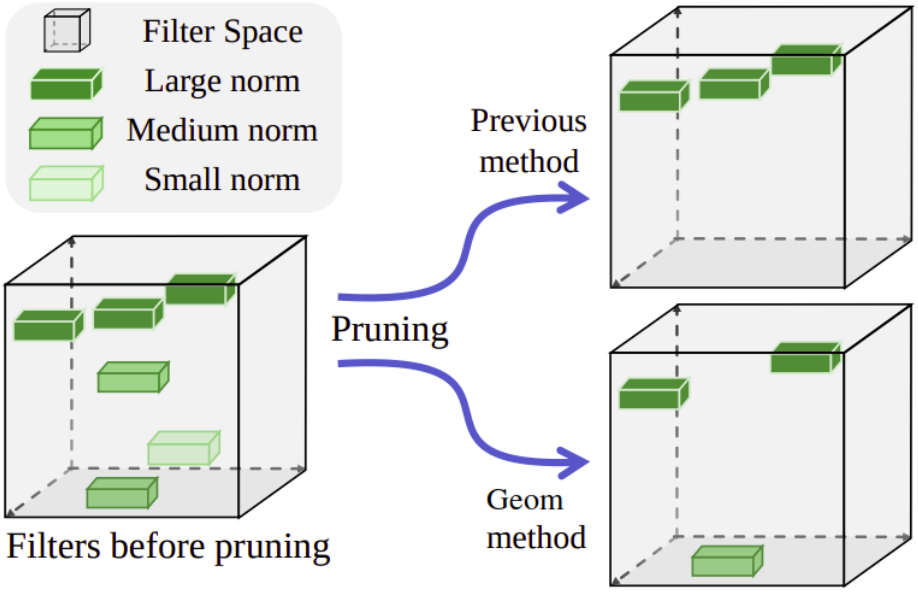


Figure 4.5: Instead of removing filters with norm close to zero, FPGM prunes filters close to the geometric media and therefore, keeping those with more diversity and able to represent the ones missing. Image from [75].

formula to produce exact results, but approximations can be computed iteratively with certain error margins depending on the running time. The authors of FPGM use a simple trick of reducing the search space to the list of filters in each layer. In this way, they find which filter minimises the sum of the distances to the rest of the filters. Once sorted, the set of distances is used to select the filters to be pruned.

4.4.2 Pruning Filters Based On The Importance Over The Loss Function

Another widespread criterion to reduce the number of filters in a model is to compute the effects of the filters on the loss function. Using a Taylor decomposition has been the standard approach to compute the importance of a filter. During the backpropagation step, the method evaluates the influence of a particular parameter when it is eliminated from the loss.

Gate Decorator [208] solves the problem of importance filter ranking by using the approach mentioned above and removing the less essential filters iteratively until the pruning effect reaches a certain amount of FLOPs in the reduced model. The method typically modifies the batch normalisation layers (BN) [87] to compute the importance of the features maps in the previous convolution layer using the scaling factor of BN.

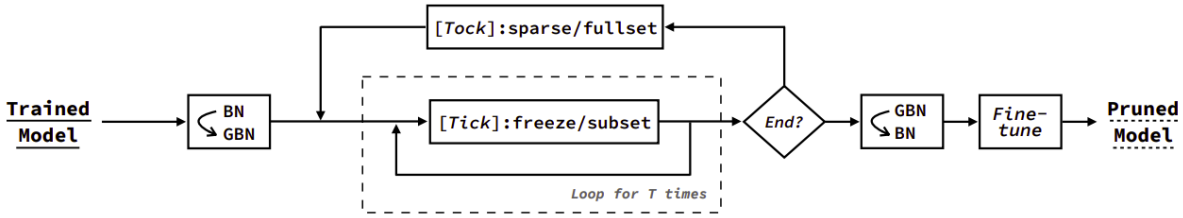


Figure 4.6: Gate Decorator framework divides pruning into two phases. The tick phase freeze weights and compute filter importance in the loss using a small subset of the data. The tock phase prunes less important weights and fine-tunes the model. Image from [208].

Gate Decorator framework splits the pruning process into the two phases shown in Figure 4.6. In the first phase, called *Tick*, all the weights in the model are set to non-updatable. Nevertheless, the modified BN layers remain trainable to compute the filter importance using a small subset of the data. After one epoch of training, a portion of the less important filters is removed. The second step named *Tock* uses the whole dataset to perform fine-tuning of the model with an added constraint to the loss function that helps to find unimportant filters.

4.4.3 Reducing Filters With Filter Decomposition

Filter Basis [114] uses filter decomposition replacing the original filters with a set of small ones that are combined linearly to produce the feature maps of the original layer while minimising reconstruction error. Filter Basis is better classified as a model compression method rather than a pruning method, given that it does not remove specific filters. Instead, it replaces them with a set of low-rank matrices that operates with the whole set of feature maps.

The authors implement the decomposition by doing convolutional operations exploiting the existing forward pass during the model training. All the feature maps are processed individually with the same basis and then the appropriate number of channels is obtained with a 1x1 convolution as presented in Figure 4.7. Learning the parameters of the low-rank matrices is done jointly by adding the feature maps restoration error in the loss function given by equation

$$loss = \left\| y - f_{\mathbf{B},\mathbf{A}|\Theta}(x) \right\|_F^2 + \gamma \sum_{l=1}^{L} \left\| \mathbf{W}^l - \mathbf{B}^l \cdot \mathbf{A}^l \right\|_F^2$$

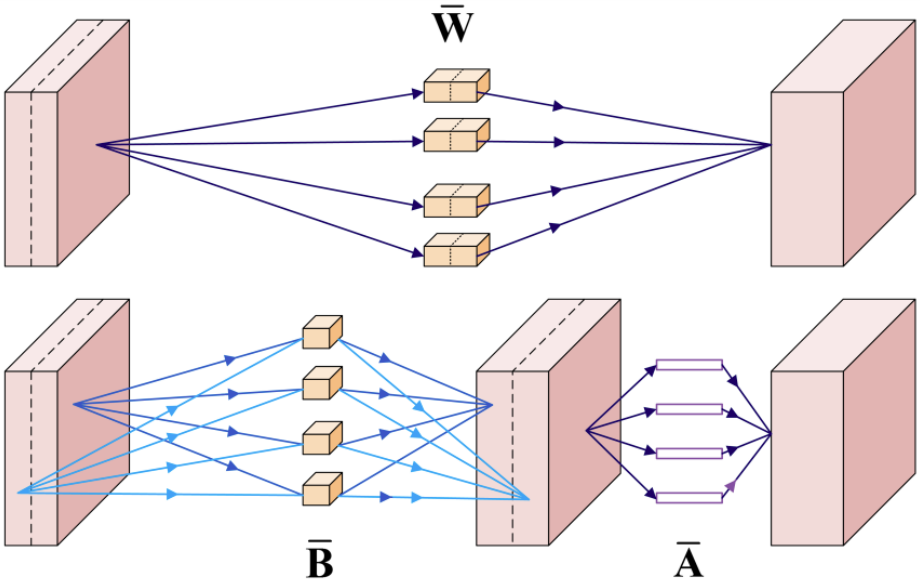


Figure 4.7: Filter Basis uses decomposition to approximate feature maps produced by original filters with low-rank matrices. Image from [114].

where $f_{\mathbf{B},\mathbf{A}|\Theta}(x)$ denotes the network with parameters $\mathbf{B},\mathbf{A}$ conditioned that the rest of the parameters Θ are known. The feature maps reconstruction error F is evaluated along all of the L layers of the network.

4.4.4 Experiments and Results

We conducted an experiment to show the effects of templates with three different recent compression methods using the CIFAR-100 dataset. We tested templates in two different ways using a VGG19 model. In the first one, templates are compared with the pruning methods applied to the original base model. In the second approach, the networks are first changed with the templates, trained and then pruned with the different compression techniques. Once neural network architectures have been adjusted with our templates and trained, pruning methods are still able to reduce models.

We present results for VGG19 model in CIFAR-100 dataset comparing accuracy versus the number of parameters in Table 4.11 for the three methods using code and configurations referenced in their respective papers. The purpose of the table is to observe improvements in each method when applying templates and a comparison between methods should be taken with care. There are differences in the number of epochs for training performing 160 for Geometric Mean, 300 for Filter Basis and variable epochs in Gate Decorator (final fine-tuning step is not performed in our experiments with

Table 4.11: Accuracies and parameters (millions) of VGG19 with the original distribution and four templates for the same number of filters evaluated on CIFAR-100 after being compressed using three different methods. When compared on similar parameters templates produce higher performances. Geometric Median reduces models based on percentage of pruned filters, Gate Decorator on the percentage of FLOPs and Filters Basis on the number of basis created. Comparisons between compression methods should consider differences between published training configurations as number of epochs. Results show average of three repetitions.

		Base		Reverse		Uniform		Quadratic		Neg-Quad	
Method	Size	Acc.	Par.	Acc.	Par.	Acc.	Par.	Acc.	Par.	Acc.	Par.
Baseline	100%	71.98	20.34	**74.54**	20.12	73.64	16.22	73.14	**16.08**	74.05	20.06
Geometric Median[75]	80%	70.83	12.93	**73.81**	12.80	72.85	10.33	72.35	**10.23**	73.56	12.76
	50%	67.96	5.11	**71.83**	5.02	70.67	4.07	69.39	**4.03**	71.05	5.01
	25%	61.39	1.30	**67.91**	1.26	65.01	**1.03**	64.38	**1.03**	66.61	1.25
	10%	48.00	0.21	**54.28**	0.20	52.89	**0.17**	48.80	**0.17**	51.90	0.19
Gate Decorator[208]	80%	71.63	11.91	**74.15**	18.69	72.96	15.01	72.66	14.70	74.11	**10.87**
	50%	71.07	**2.63**	**74.28**	15.66	73.46	8.72	72.96	10.20	73.08	5.70
	25%	66.30	**1.12**	**74.63**	10.14	72.98	4.57	73.39	5.35	71.72	2.70
	10%	59.58	**0.47**	**73.15**	4.10	68.73	1.84	71.13	2.57	67.01	1.02
Learning Filter Basis[114]	64	73.59	9.54	75.42	12.07	74.63	**7.73**	73.76	9.33	**75.62**	9.33
	32	69.74	5.58	75.13	9.13	73.20	**5.53**	72.28	7.23	**75.04**	5.67

Gate Decorator). Baseline accuracies are produced by applying only templates, and they differ slightly from results obtained in the classification task in the previous chapter (Table 3.1) as they were obtained with the Gate Decorator code. We found that models with templates perform better than the original model using similar parameters for the first two methods. For Gate Decorator, we can find at least one template with better accuracy than the pruned base model. For example, keeping parameters close to 1.10 million, a negative-quadratic model with 10% of FLOPs performs better than a base model with 25% of FLOPs.

We additionally compare accuracy versus FLOPs in Table 4.12 using the three previous methods with the same model/dataset. It is found that a reduction in FLOPs causes a degradation in accuracy for templates excepting Negative-Quadratic which helps to reduce the number of flops. Using similar FLOPs that the baseline original model, Negative-Quadratic template in addition to pruning methods, produces networks that surpass the original accuracy as with Filter Basis that improves 3.06%.

Table 4.12: Accuracies and FLOPs (billions) of VGG19 with the original distribution and four templates for the same number of filters evaluated on CIFAR-100 after being compressed using three different methods. Geometric Median reduces models based on percentage of pruned filters, Filters basis on the number of decompositions and Gate decorator on the percentage of FLOPs. Comparisons between compression methods should consider differences between published training configurations as number of epochs. Results show average of three repetitions.

		Base		Reverse		Uniform		Quadratic		Neg-Quad	
Method	Size	Acc.	Flops	Acc.	Flops	Acc.	Flops	Acc.	Flops	Acc.	Flops
Baseline	100%	71.98	**0.40**	**74.54**	4.29	73.64	2.00	73.14	3.06	74.05	1.05
Geometric Median[75]	80%	70.83	**0.25**	**73.81**	2.74	72.85	1.28	72.35	1.95	73.56	0.67
	50%	67.96	**0.10**	**71.83**	1.07	70.67	0.50	69.39	0.76	71.05	0.26
	25%	61.39	**0.02**	**67.91**	0.27	65.01	0.12	64.38	0.19	66.61	0.06
	10%	48.00	**0.004**	**54.28**	0.04	52.89	0.02	48.80	0.03	51.90	0.01
Gate Decorator[208]	80%	71.63	**0.31**	**74.15**	3.41	72.96	1.59	72.66	2.42	74.11	0.84
	50%	71.07	**0.19**	**74.28**	2.14	73.46	1.00	72.96	1.52	73.08	0.52
	25%	66.30	**0.09**	**74.63**	1.07	72.98	0.50	73.39	0.76	71.72	0.26
	10%	59.58	**0.04**	**73.15**	0.43	68.73	0.20	71.13	0.30	67.01	0.10
Learning Filter Basis[114]	64	73.59	**0.20**	75.42	3.59	74.63	1.43	73.76	2.75	**75.62**	0.57
	32	69.74	**0.13**	75.13	3.31	73.20	1.31	72.28	2.63	**75.04**	0.40

4.5 Templates + MorphNet: Improving the Search of Filter Distribution

In the previous section, we compared some pruning methods against the ability of templates for reducing parameters in a neural network. Pruning methods' goal is to take an architecture and remove all unnecessary elements such as weights or filters.

MorphNet was one of the first techniques which were not based on only cutting off filters of a previously defined neural network but searching for an optimal distribution of them [64]. Furthermore, MorphNet is configurable to reduce a particular resource while still increasing the accuracy of the model. For example, when targeting FLOPs, higher-resolution neurons in the lower layers of the CNN tend to be sacrificed more than lower-resolution neurons in the upper layers of the CNN. The situation is the exact opposite when the targeted resource is model size rather than FLOPs.

We chose to test our templates with MorphNet looking for the answer regarding MorphNet converging to the same filter distribution when starting from the different templates. An affirmative answer would suggest that there is an optimal distribution of filters. Even if the convergency was specific to a dataset and model. The result could provide more insight to neural network designers.

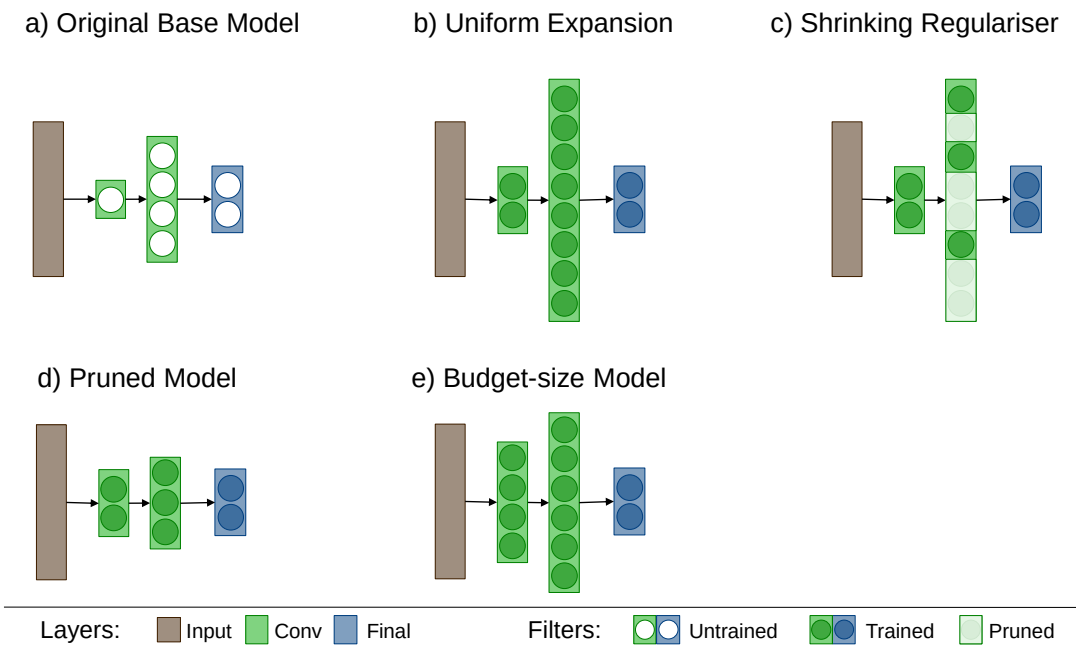


Figure 4.8: MorphNet iterative steps for finding the best filter distribution matching a particular resource in a toy example. All the filters in an untrained base model (a) are expanded with an arbitrary global width multiplier (b). During training, a regulariser shrinks the model removing less important filters (c). The resulting pruned model (d) is then expanded again using a width multiplier to match a resource budget (e).

4.5.1 MorphNet Steps for Optimising the Filters Distribution

The searching process performed in MorphNet is divided into two phases. During expansion, a simple method is used only, which uniformly expands the number of filters in each layer with a width multiplier to reach the resource constraint. In the shrinking phase, MorphNet identifies inefficient neurons and prunes them from the network by applying a sparsifying regulariser such that the total loss function of the network includes a cost for each neuron (see Figure 4.8).

The above steps completed one cycle of improving the network architecture. The process continues iteratively until the performance is acceptable or until the architecture converges, leading to similar filters per layer after several iterations. Yet, a single iteration of expansion and shrinking is enough to deliver a model with appreciable improvement over the naive solution of just using a uniform width multiplier. When a layer has zero neurons, this effectively changes the topology of the network by cutting the affected branch from the architecture. If the final model doesn't fit the predefined budget, MorphNet adjusts it with a width multiplier.

4.5.2 Experiments

We tested MorphNet with three of the most used architectures: VGG, ResNet and MobileNet. We modified the original models with four templates. Then, we run MorphNet with the resulting network and trained the final design from scratch. We compared the results with the performances obtained in the classification task section. We empirically adjusted the additional hyperparameter γ following the recommendations described in the MorphNet paper. The iterative process to learn the architecture was repeated until convergence defined by the suggested threshold in MorphNet.

Similar to the experiments with pruning methods, we compared the templates in two ways for each of the three models and three datasets tested. The first way compares directly templates versus MorphNet. The second way compares templates cooperatively working with MorphNet. We initially changed the networks with the templates and next we run the process of learning the distribution with MorphNet. Finally, we trained the neural network architectures using the hyperparameters described in [64]. All models produced from the same original architecture, either directly produced by templates or by MorphNet are adjusted to the same amount of FLOPs to make a fair comparison. We noticed that the MorphNet learning procedure was performed in CIFAR-10 and transferred to the rest of the datasets.

Table 4.13 show results for the VGG19 model counting around 400 MFLOPs. MorphNet was able to perform well in CIFAR-10 and CINIC datasets but the baseline model surpassed the accuracies of MorphNet with all the templates. On the other hand, models changed only by templates obtained the highest accuracy with the *decreasing* (a.k.a. *reverse*) and *negative quadratic* distributions.

For ResNet results were lower than the VGG ones. We see in Table 4.14 that only for CIFAR-100 MorphNet delivered a better accuracy than the base model but used a uniform distribution as seed. Like with VGG, ResNet plus templates produced the highest accuracy in two datasets with the *negative quadratic* distribution. All ResNet models were set to 1307 MFLOPs.

Figures 4.9 and 4.10 depict the final distributions of filters obtained by MorphNet using each template in VGG19 and ResNet50. We observe a trend in the final VGG models with more filters in the near initial layers as well as in the final layers. A reduction of filters can be seen in the middle of almost all VGG architectures. In ResNet, the pattern remains but is less noticeable. It looks improbable that MorphNet is able to converge to a unique distribution, but we can not conclude that an optimal distribution is non-existent. We believe that the lack of convergence is caused by the definition of the

Table 4.13: Accuracy of models produced by combining VGG19 + Templates + MorphNet + Width Multiplier compared to only using Templates + Width Multiplier. All models are adjusted to ~399 MFLOPs. MorphNet filter distribution search is performed on CIFAR10. Table shows one-repetition results. *Increasing uses the original base filter distribution therefore it is not applicable for templates only.

VGG19		Templates + MorphNet (~399 MFLOPs)				
Dataset	Base	Increasing*	Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	92.98	93.19	92.58	91.92	93.06	**93.54**
CIFAR100	**71.22**	71.03	69.61	66.23	70.17	67.56
CINIC10	83.33	**83.77**	82.74	81.83	83.66	83.76
VGG19		Templates Only (~399 MFLOPs)				
Dataset	Base		Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	92.98		**93.43**	**93.43**	93.23	93.16
CIFAR100	71.22	NA	69.52	70.56	67.37	**71.91**
CINIC10	83.33		83.90	83.92	83.51	**84.17**

Table 4.14: Accuracy of models produced by combining ResNet50 + Templates + MorphNet + Width Multiplier compared to only using Templates + Width Multiplier. All models are adjusted to ~1307 MFLOPs. MorphNet filter distribution search is performed on CIFAR10. Table shows one-repetition results. *Increasing uses the original base filter distribution therefore it is not applicable for templates only.

ResNet50		Templates + MorphNet (~1307 MFLOPs)				
Dataset	Base	Increasing*	Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	**94.31**	92.96	92.91	93.83	93.36	93.36
CIFAR100	73.25	73.42	71.51	**74.06**	66.55	70.25
CINIC10	**87.04**	84.09	84.67	85.60	86.38	83.77
ResNet50		Templates Only (~1307 MFLOPs)				
Dataset	Base		Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	94.31		92.90	94.33	92.57	**94.46**
CIFAR100	73.25	NA	70.03	73.84	68.37	**73.86**
CINIC10	**87.04**		85.77	86.73	85.12	86.94

Table 4.15: Accuracy of models produced by combining MobileNetV1 + Templates + MorphNet + Width Multiplier compared to only using Templates + Width Multiplier. All models are adjusted to ~47 MFLOPs. MorphNet filter distribution search is performed on CIFAR10. Table shows one-repetition results. *Increasing uses the original base filter distribution therefore it is not applicable for templates only.

MobileNetV1		Templates + MorphNet (~47 MFLOPs)				
Dataset	Base	Increasing*	Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	91.45	**91.70**	89.06	88.90	86.73	82.38
CIFAR100	62.58	**65.02**	56.31	57.85	46.79	43.80
CINIC10	80.99	**82.09**	77.09	76.64	73.22	66.73
MobileNetV1		Templates Only (~47 MFLOPs)				
Dataset	Base		Decreasing	Uniform	Quadratic	Neg-Quad
CIFAR10	91.45		91.14	92.02	87.44	**92.08**
CIFAR100	62.58	NA	63.99	64.76	53.47	**68.81**
CINIC10	80.99		81.72	**82.85**	77.75	82.67

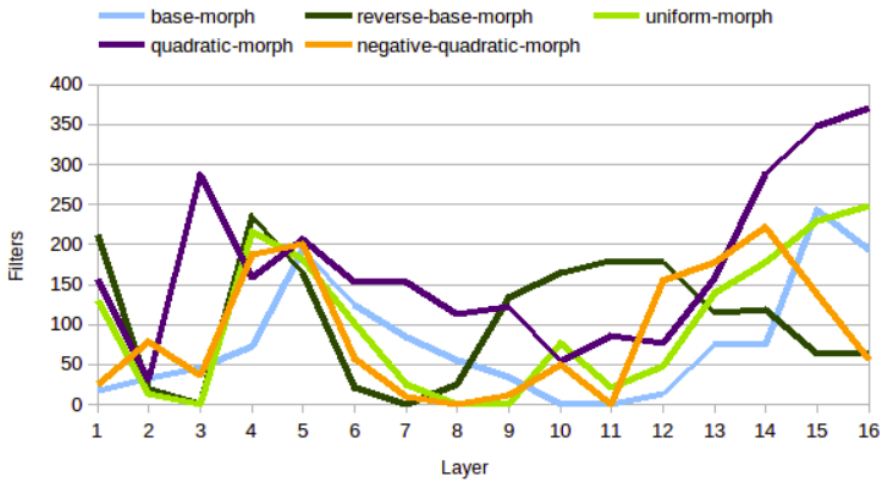


Figure 4.9: Final filters in VGG19 architecture using Templates + MorphNet. The plot shows MorphNet final filter distributions tending to converge to the same distribution.

constraint in MorphNet.

It can be argued that using only templates to find a good performing model can demand excessive computational resources to train each of the different templates. Nevertheless, we have observed the lack of convergence of MorphNet in the resulting filter distributions when the process initiates from different filter distributions. The implication is that, in order to find a good model, it would be recommendable to explore several initial distributions. This process could add more resources than the required for only training the different templates, not adding the superior performance obtained by

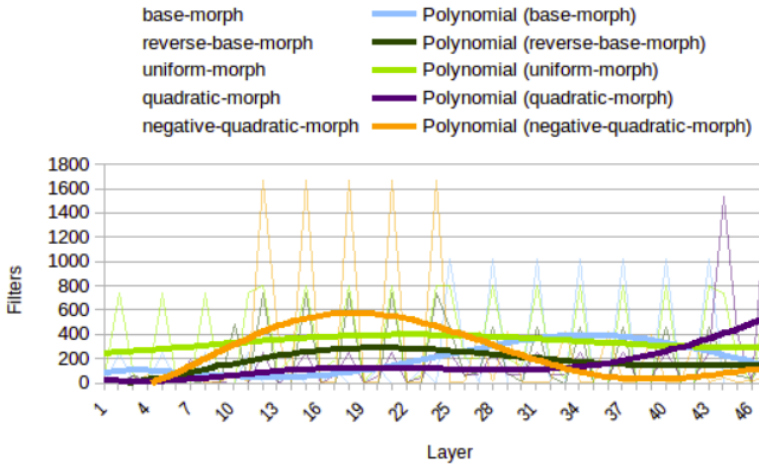


Figure 4.10: Final filters in ResNet50 architecture using Templates + MorphNet. Continuous lines are polynomial aproximations for smoothing the fluctuations in the number of layers caused by the bottlenecks modules.

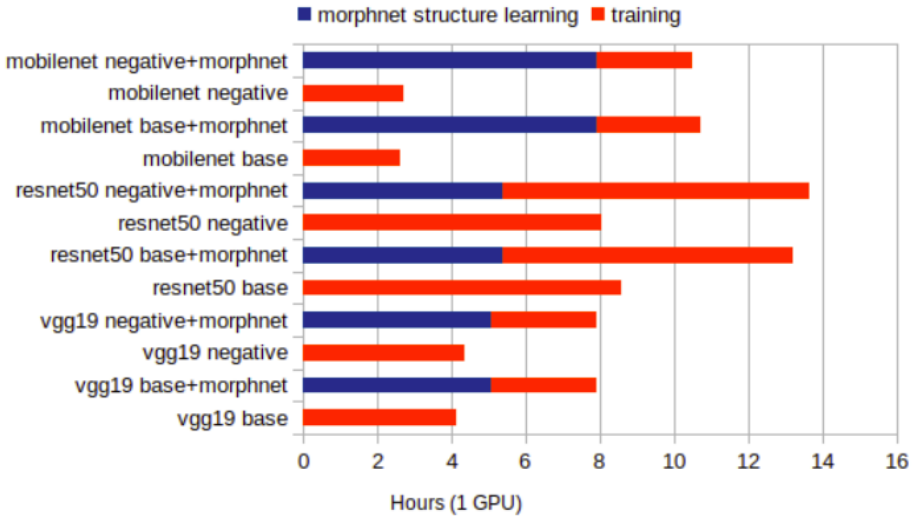


Figure 4.11: Comparison of individual and combined training times of templates and MorphNet for several architectures.

only using the templates. We show in Figure 4.11 the time taken for training a template compared to the time for searching a model with MorphNet. It is observed that the time needed for MorphNet, combined with the training time for the resulting model, surpasses the time of applying a template and training the model from scratch.

The overall results are summarised in Table 4.16. Templates were able to produce models with higher accuracy than the networks produced by MorphNet in six out of the nine combinations of datasets and models. MorphNet produced two of the best networks

Table 4.16: Best accuracy and pruning method by dataset compared to the original base model.

Model	Dataset	Base	Best	Method
VGG19	CIFAR10	92.98	93.54	MorphNet (Negative)
~399 Mflops	CIFAR100	71.22	71.91	Templates (Negative)
	CINIC10	83.33	84.17	Templates (Negative)
ResNet50	CIFAR10	94.31	94.46	Templates (Negative)
~1307 Mflops	CIFAR100	73.25	74.06	MorphNet (Uniform)
	CINIC10	87.04	87.04	Base
MobileNetV1	CIFAR10	91.45	92.08	Templates (Negative)
~47.2 Mflops	CIFAR100	62.58	68.81	Templates (Negative)
	CINIC10	80.99	82.85	Templates (Reverse)

but not starting from the original (increasing) distribution of filters. Instead, the initial models were modified with the *Negative Quadratic* and *Uniform* templates. The results highlight again the need for exploring new and diverse distributions of filters like the ones proposed in our templates which can offer more efficient neural network designs.

4.6 Conclusion

The experimental results in this chapter suggest that other domains beyond image classification are also prone to the benefits produced with templates. For example, in the localisation task, we found the templates successful in more than half of the locations tested in 7-scenes with PoseNet. Furthermore, the modified PoseNet achieved lower MSE using 8% less memory. In the same dataset, VGG only performed better in two locations. However, the second-best performing model, produced with a template, counted only 35% of the original parameters. The template-changed PoseNet and MobileNetV1 obtained lower errors than the original models in six out of seven locations for the Cambridge Landmarks dataset. The best template in MobileNetV1 achieved better performance while using 10% fewer parameters.

Experiments in SR showed that the correct distribution is important. While templates may not have been able to show improvements on the tested datasets, the experiments in classification leave open the possibility that different distributions could make a difference when the dataset changes. Even though one template achieved a lower reconstruction error (measured with PNSR) than the original model, it was done at the cost of more computational resource demands in all aspects. It is worth noticing that in SR, the adopted distribution is not the incremental one. Models designed for the task follow a

uniform distribution of filters. In other words, the incremental distribution was not the best for this task.

When we studied how the templates can collaborate with pruning methods, we found that the results of VGG models on CIFAR100 improved when the pruning process started with models modified with templates. The Geometric Median method was able to increase one point of accuracy in a model with a reduction of 50% of parameters. The Gate Decorator method, which focuses on reducing FLOPs, found a model performing more than one point higher in accuracy with approximately the original FLOPs. Filter Basis, which replaces filters with low-rank matrix operations, found a model with 5.3 points higher in accuracy compared to the model obtained with the same method but starting from an incremental distribution. These results highlight the advantages of using templates together with pruning methods to get more efficient models.

Finally, the chapter includes research about how templates change the behaviour of a CNS method. Using three classical neural network models and three datasets, we evaluated MorphNet and templates in two ways: competing and collaborating. We restricted MorphNet and templates to produce models with similar FLOPs to the base model. There were improvements in accuracy in eight out of nine scenarios caused by using only templates or some combination of MorphNet and templates. For example, for VGG, MorphNet produced the best model in CIFAR10 with 0.56 points more accuracy but started its process with a template-changed VGG model. Something similar happened with ResNet and CIFAR100, improving accuracy by 0.81 points. For the rest of the cases, templates showed superior performance to MorphNet. In particular, MobileNetV1 model improves significantly with templates with up to 6.2 points in accuracy for CIFAR100.

In general templates were effective for the task and models we use for evaluation. The benefits of exploring new distributions of filters extend beyond classification tasks. In some areas like super-resolution, researchers already use a different pattern than the incremental design. Furthermore, we found that pruning and CNS methods can produce better results if they initiate the exploration process from another distribution from the incremental one. An important outcome is that there is no absolute best template. Each pair model-dataset seems to be particularly and differently affected for each template. So the proposed templates are a simple tool to rapidly obtain increases in performance without the burden of automatic neural network design methods.

CHAPTER 5

TEMPLATES 2.0

In the previous chapter, we have shown the effectiveness of templates in various domains. The strategy of changing the pattern of filter design in convolutional neural networks following mainly square functions has been found to be successful considering its simplicity. This chapter improves templates with new functions to define filters that replace "abrupt" quadratic changes in the number of filters in each layer for "smoother" linear segments. We complement the new definition with a fast method to match the number of FLOPs of the original design or any FLOPs budget, allowing a fair comparison of models. Next, we extend the experiments with templates to domains that differ significantly from the previously tested and where the internal representation of models with templates is more relevant than the final outputs. The chapter also illustrates the importance of exploring new filter distributions by applying templates to the NASBench-101 dataset. Finally, it describes the exploration with a similarity metric, trying to correlate the models' accuracies and their internal representation space as a proxy to find the best template.

5.1 Templates Redefinition to Match Similar Resources

Chapter 3 presented a first approach to producing variations in the filter distribution of existing architectures with different templates. However, comparing the resulting

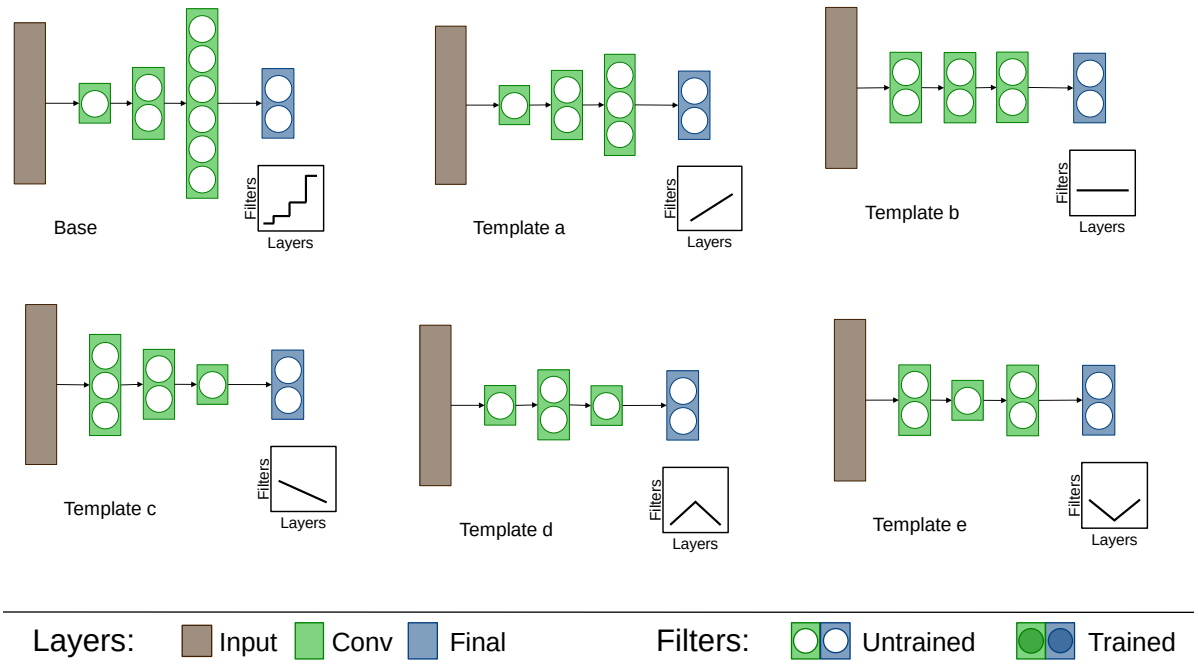


Figure 5.1: Schematic distribution of filters per layer in our templates. These templates are created with simple and intuitive but aggressive variations of filter distributions. Base distribution, which is the original pyramidal distribution, shows the common design of drastically growing the filters when the resolution of feature maps decreases in deeper layers. New templates follow a smoother transition in the number of filters between layers. We match the same number of filters in the thinnest layers and adjust the maximum to meet the original number of FLOPS for a fair comparison.

models is difficult because templates change the computational requirements of each model. So, it is desirable to reduce the effects of variations and match the number of resources. One naive solution is reducing filters proportionally across all layers with a width multiplier. Unfortunately, some models end with different sizes in the layer with the lowest number of filters and, therefore, in the minimal representation power. We want to avoid these differences in the representational bottlenecks favouring unbiased comparisons.

This section presents an improved set of template definitions using the finding of [70] related to the benefits of using additive increases in the number of filters. The new set of templates, named Templates 2.0, allows matching a predefined number of FLOPs. These templates are depicted in figure 5.1 and have been found to perform well and are thus candidates for model performance improvement beyond accuracy. Performance criteria such as parameters, memory footprint and inference time are arguably as important.

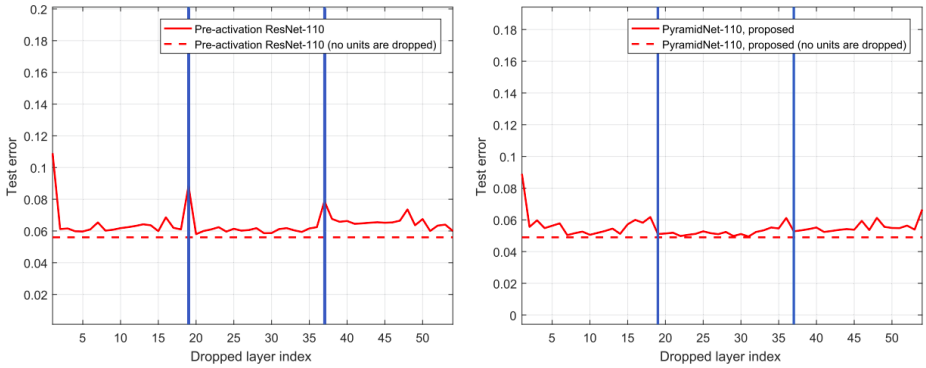


Figure 5.2: Impact of deleting one residual unit a time (solid red lines) in a sharply increasing pattern (left) and in a smooth pattern (right). Dashed lines represent test errors when no units are deleted. Blue lines point the place of downsampling layers. Taken from [70].

5.1.1 Defining a New Set of Filter Distribution Templates

To define the new set of templates, we strongly rely on the findings of the PyramidNet paper [70]. The article presents two variations of the incremental pattern for the distribution of filters in a modified ResNet architecture that brings increases in accuracy. 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. The motivation to change the distribution is the harmful effects of sharply increasing the filters between blocks (see figure 5.2), a phenomenon that has only been studied in residual networks. However, we have found that the effect extends to plain architectures. Thus, to define the new 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 adopt as the first template, one with the same incremental distribution but with a smooth step **(a)**. The second template is a distribution with a fixed number of filters **(b)** as in the original Neocognitron and Isometric Neural Networks [174]. Another immediate option, contrary to the increasing distribution, is a decreasing distribution of filters **(c)**. Finally, inspired by the distributions of blocks from the resulting ResNet101 and VGG models found in [64] and [104, 208], we 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. We believed that using one metric is insufficient for a fair comparison. As an example, 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 $F_{1:D} = \{f_1, \ldots, f_D\}$ and the total number of FLOPs, the resource to be matched between templates, is given by some function $\mathscr{R}(F_{1:D})$. We want to find a new distribution of filters $F'_{1:D}$ in which

$$\mathscr{R}(F_{1:D}) \approx \mathscr{R}(F'_{1:D}) \tag{5.1}$$

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 constraint (5.1).

5.1.2 Similar FLOPs Optimisation

CNS methods aim to find individual values for the number of filters in each layer, and thus, the exploration space becomes huge. For our method, the optimisation of the values is eased by the constraint in (5.1) 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 layer of the original model. To find the second ($n'_{max} = n$), we rely on the monotonic relationship between n and the model's FLOPs, valid for all templates. We use a modified binary search starting with the maximum number of filters (n_{max}) 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 (See Algorithm 1).

Algorithm 1: Producing a CNN model using templates with FLOPs similar to an existing base model.

Input:
CNN_{base}
n_{max}
$template \leftarrow [a,b,c,d,e]$
Output:
$CNN_{template}$
n'_{max}
begin
  $F \leftarrow compute\_flops(CNN_{base})$
  $n \leftarrow n_{max}$
  repeat
    $CNN_{template} \leftarrow change\_filters(CNN_{base}, template, n)$
    $F' \leftarrow compute\_flops(CNN_{template})$
    $factor \leftarrow F'/F$
    $n_{old} \leftarrow n$
    $n \leftarrow int(n/factor)$
  until $n = n_{old}$;
  $n'_{max} \leftarrow n$
end

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 in all models and templates tested in this work in A.1 (appendix).

5.2 Templates 2.0 on Image Classification

We investigated the effects of applying different templates to the distribution of kernels in well known convolutional neural network models (VGG, ResNet, MobileNet and Mnasnet). We highlight that the resulting models obtained from templates have similar FLOPs to the original model from which they are obtained. This constraint facilitates

further comparison of models under the basis of size, memory and speed tested in several popular datasets for classification tasks.

5.2.1 Datasets and Models

We selected six datasets with diverse domains, number of samples, and classes to test our templates but allowed a relatively fast training process. Each model is evaluated with a set of five templates. Therefore, we use MNIST, FashionMNIST, CIFAR-10, CIFAR-100 [99], CINIC-10 [38] and Tiny-Imagenet [103]. The first four datasets contain sets of 50,000 and 10,000 samples for train and validation, respectively. MNIST and FashionMNIST 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. 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[183] and ResNet[73] models, which represent some of the most influential CNN architectures on the ImageNet challenge in previous years [37, 170] as well as MobileNetV2[175], one highly optimised model, and MnasNet[189], an automatically produced architecture from a NAS method.

5.2.2 Implementation Details

Experiments have models fed with images with the standard augmentation techniques of padding, random cropping and horizontal flipping and additionally, with cutout [41] 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.

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-10, CIFAR-100 and CINIC-10, all 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 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.

5.2.3 Effects of Templates on Classical Models

We conducted an experiment to test our proposed templates on the selected architectures. Table 5.1 show properties of resulting models for each architecture after using templates. Parameters, memory footprint and inference time are reduced in almost all cases. This result is in some way surprising given that template patterns were only selected following simplicity and diversity but not precisely efficiency.

In particular, for classical models, we observe in Figure 5.3 increases in accuracy up to 2.11 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 es 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 found that there is a frequent behaviour related to each template that is clearly observed in Figure 5.3. 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 detailed results of our experiments with classical models in Table 5.2.

5.2.4 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 5.1. Although it is expected that the margin of improvement on these highly optimised models be considerably lower, we found that templates can reduce 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. On the other hand, we keep similar FLOPs for all the templates in our experiments. We think the differences in inference

Table 5.1: Resource consumption on CIFAR-10 for original architectures and resulting models after applying templates. FLOPs remain similar in comparison to base models. Classical models show bigger reductions in all aspects, while optimised models benefit more in parameters. Negative values represent increases in resources.

Template	Param (Millions)		Mem (MB)		Inference Time (ms)		FLOPs (Millions)
vgg19 base	20.03	% ↓	87.0	% ↓	1.85	% ↓	399.2
vgg19 a	17.23	13.9	76.5	12.0	1.85	0.0	396.9
vgg19 b	3.17	84.1	23.0	73.5	**1.40**	**24.3**	400.2
vgg19 c	**1.89**	**90.5**	**17.8**	**79.5**	1.43	22.7	399.5
vgg19 d	8.07	59.7	39.8	54.2	1.46	21.0	399.8
vgg19 e	2.06	89.7	**17.8**	**79.5**	1.43	22.7	399.0

Template	Param (Millions)		Mem (MB)		Inference Time (ms)		FLOPs (Millions)
resnet50 base	23.52	% ↓	185.5	% ↓	5.35	% ↓	1307.7
resnet50 a	14.17	39.7	146.8	20.8	4.83	9.7	1301.9
resnet50 b	4.85	79.3	132.1	28.7	**4.29**	**19.8**	1299.0
resnet50 c	**3.48**	**85.2**	128.9	30.5	4.30	19.6	1293.5
resnet50 d	8.36	64.4	**125.8**	**32.1**	4.32	19.2	1307.3
resnet50 e	3.68	84.3	132.1	28.7	4.34	18.8	1297.7

Template	Param (Millions)		Mem (MB)		Inference Time (ms)		FLOPs (Millions)
mobilenet base	2.23	% ↓	28.3	% ↓	3.81	% ↓	68.9
mobilenet a	1.42	36.3	27.2	3.8	3.86	-1.3	68.4
mobilenet b	0.80	64.1	28.3	0.0	3.91	-2.6	67.7
mobilenet c	0.59	73.5	27.2	3.8	3.71	2.6	68.1
mobilenet d	1.12	49.7	27.2	3.8	**3.68**	**3.4**	68.1
mobilenet e	**0.51**	**77.1**	**25.1**	**11.3**	3.92	-2.8	68.8

Template	Param (Millions)		Mem (MB)		Inference Time (ms)		FLOPs (Millions)
mnasnet base	3.11	% ↓	**82.8**	% ↓	3.79	% ↓	314.6
mnasnet a	1.57	49.5	98.5	-18.9	3.68	2.9	311.8
mnasnet b	1.05	66.2	101.7	-22.8	3.85	-1.5	309.6
mnasnet c	**0.79**	**74.5**	101.7	-22.8	3.82	-0.7	312.1
mnasnet d	1.14	63.3	98.5	-18.9	**3.61**	**4.7**	313.3
mnasnet e	0.93	70.0	100.6	-21.4	3.75	1.0	314.5

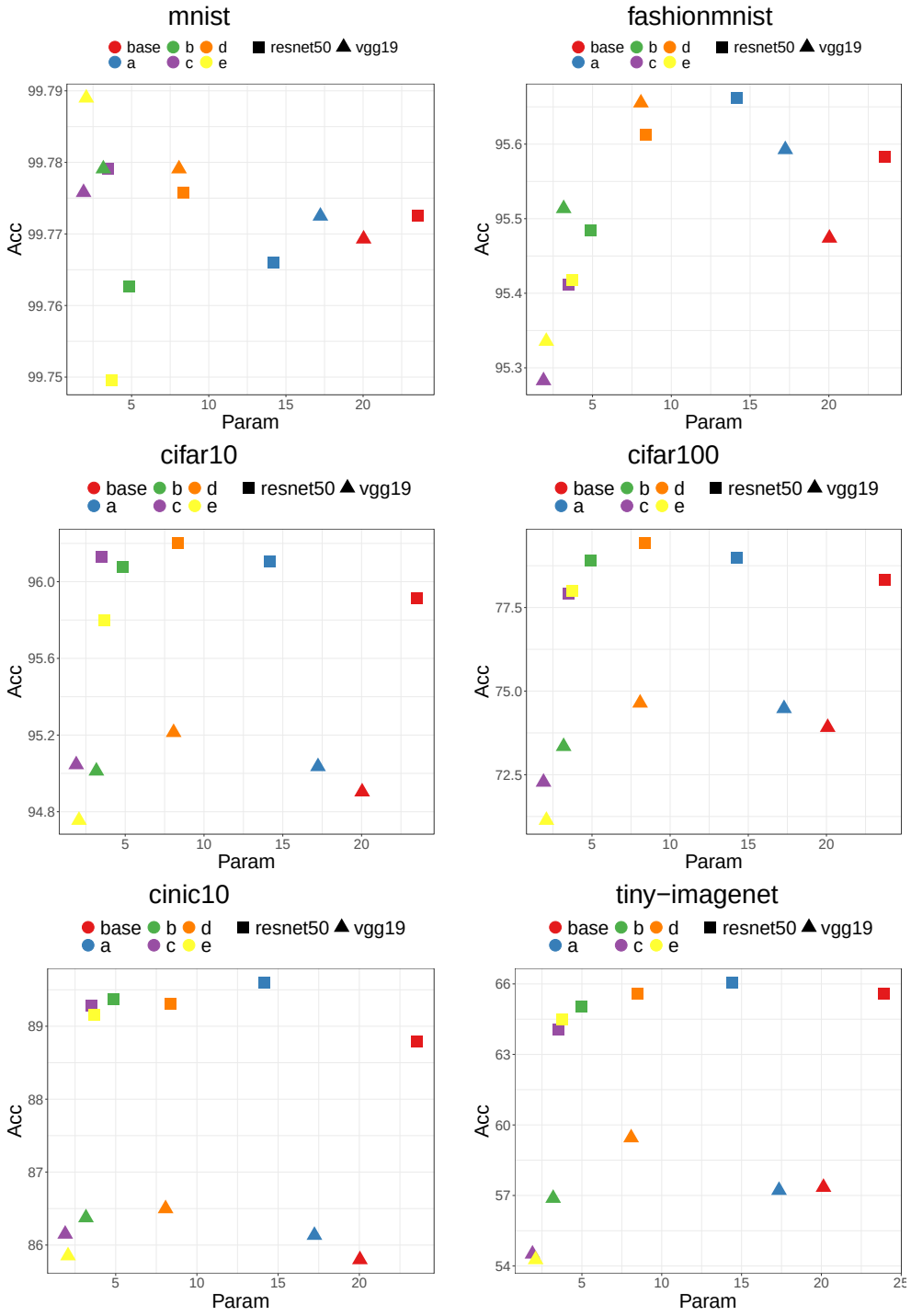


Figure 5.3: Accuracy of VGG and ResNet models after applying templates reported for several datasets. Base is the original distribution of filters. In many cases, templates outperform the base architecture. However, all of them use far fewer parameters than the base model. Note that models produced with templates from VGG have less than a third of FLOPs of those produced from ResNet.

Table 5.2: 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

time come in how well the sizes and the number of feature maps fit better in GPU memory. Since MobileNet and MNASNet more optimised, templates struggle to produce significant improvements in this metric but still, models reach almost 5% of reduction in inference time.

From Figure 5.4, 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 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 think the reason for this effect is that tiny-Imagenet is strongly related to Imagenet, and models' designs have been overfitted 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 detailed results for these models in Table 5.3.

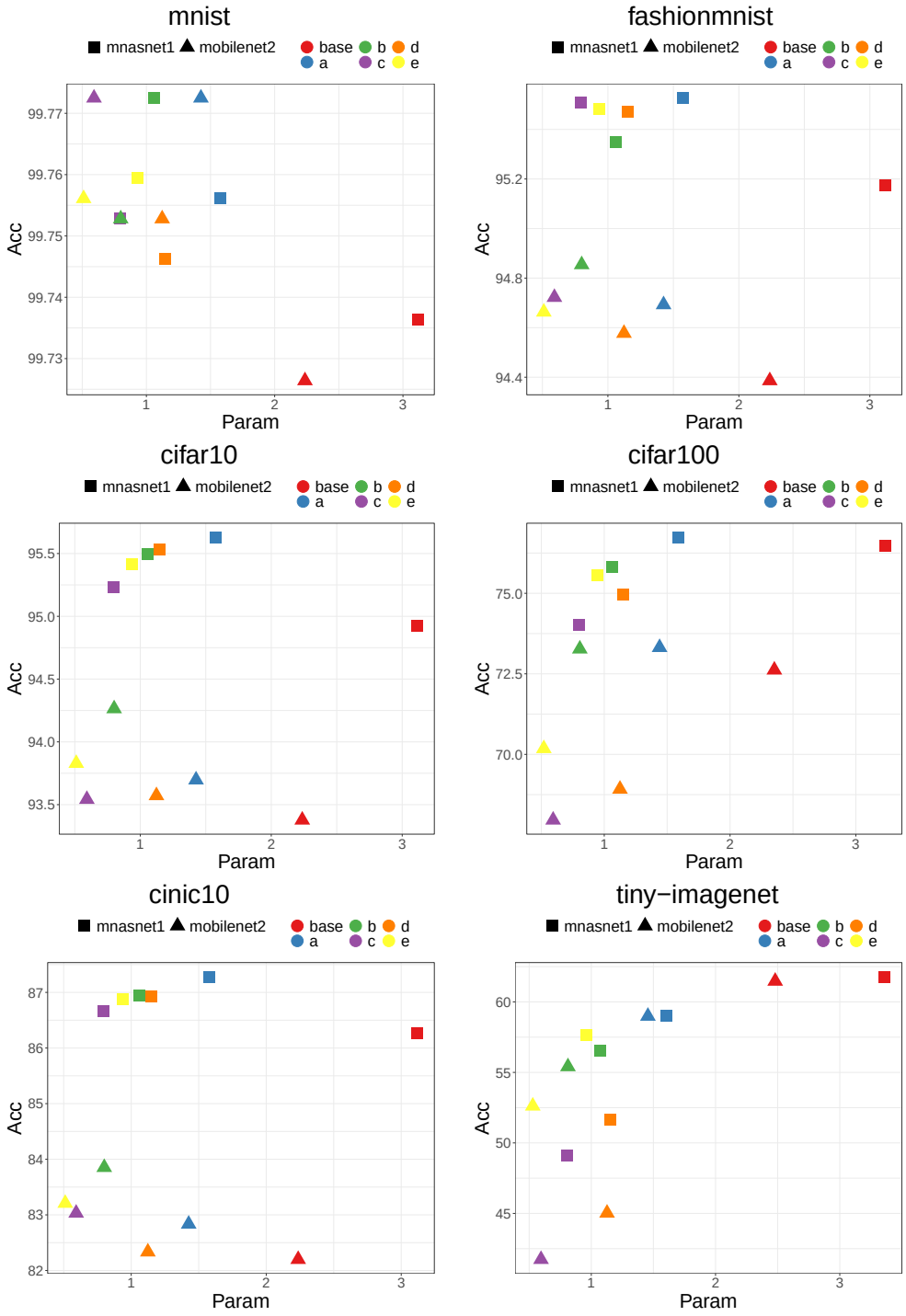


Figure 5.4: Parameter efficiency of MobileNetV2 and MnasNet models with templates reported for several datasets. Base is the original distribution of filters. In many cases, templates outperform the base architecture. However, all of them use far fewer parameters than the base model. Note that models produced with templates from MnasNet have more than 4X FLOPs of those produced from MobileNetV2.

Table 5.3: 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

5.3 Templates 2.0 on Audio Classification

One of the areas where research in deep learning has enabled many applications is signal processing. After evolving in computer vision tasks, neural networks have been widely adopted in many domains. In particular, they have been applied to audio processing, including speech, music and environmental sound processing. Despite there being significant differences between computer vision and audio domains, many of the existing methods in the latter have been borrowed from the former [156]. However, researchers have started to develop audio-focused techniques [62, 97, 172].

Raw audio samples come from a one-dimensional signal indexed in time [156]. 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 [58, 125, 218]. 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 [68].

Based on the differences mentioned above with computer vision tasks, we decided to test our templates in two popular datasets from the audio classification domain [147]. We explore the accuracy and resource consumption of our proposed improved versions for

filter distributions and compare them against using the traditional pyramidal pattern.

5.3.1 Audio Datasets

The GTZAN dataset [193] is the most-used public dataset for evaluation in machine listening research for music genre recognition (MGR) [186]. 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: Blues, Classical, Country, Disco, Hip Hop, Jazz, Metal, Popular, Reggae, and Rock.

Another popular audio dataset is the ESC-50 dataset, designed to provide a benchmark for environmental sound classification [154]. Analysis of environmental sounds is considered a different task from other everyday audio events such as speech or music (laughter, cat meowing, glass breaking or brushing teeth are some examples of environmental sounds). 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. Some examples of MFCCs for the two datasets used in our experiments are presented in Figure 5.5.

5.3.2 Implementation Details

When convolutional neural networks are applied to raw waveform input, it is normally done with 1-d convolutions. However, we use spectral input features for which 2-d time-frequency convolution is commonly adopted. Neural models in audio use common elements and follow similar pattern designs as computer vision models. They are composed of a series of convolutional layers with pooling layers at the end of each sequence to downsample the learned feature maps, followed by one or more dense layers to produce the desired output according to the task to be solved.

According to [156], 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 most, if not all, 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. Instead, they use classical archi-

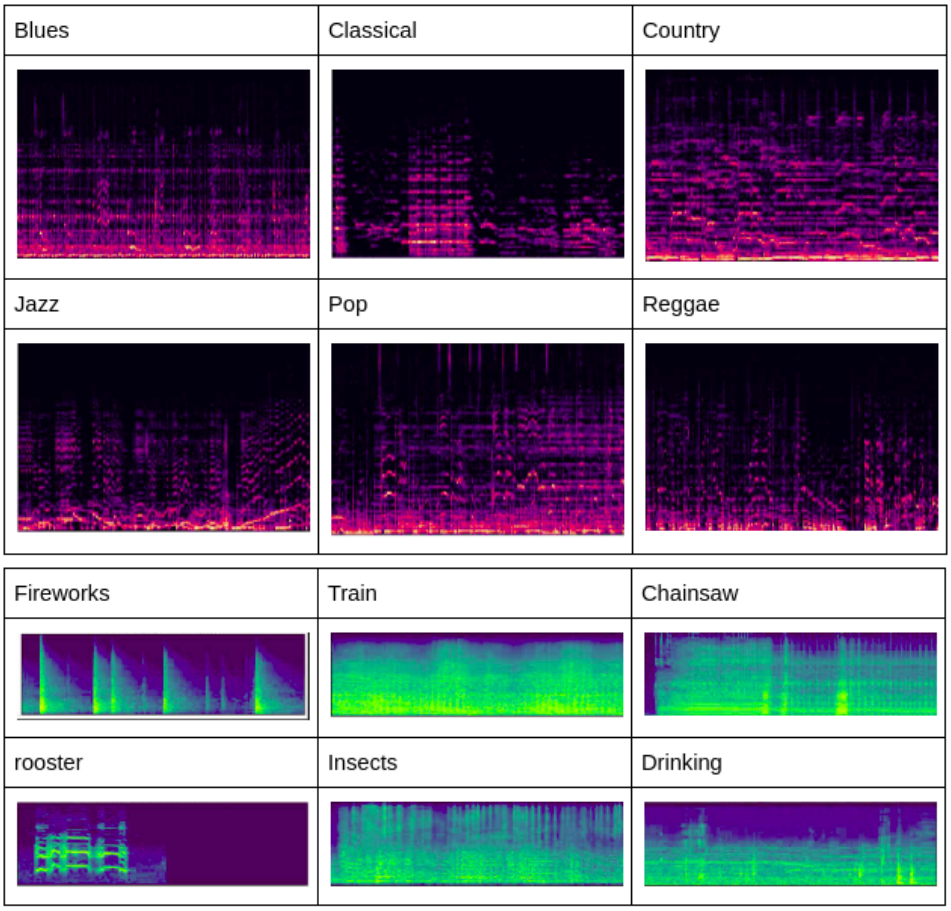


Figure 5.5: MFCC spectrograms for some samples in the GTZAN (purple lines) and ESC-50 (green lines) datasets.

tectures as a backbone, of which ResNet is one of the most popular [67]. 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.

For training, we use the code and hyperparameters provided by [147], which were found with grid search [117]. 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.

5.3.3 Results

The results of this experiment are shown in Table 5.4. They show the average accuracy of three runs. By using templates on ResNet50 we can see marginal improvements

Table 5.4: Accuracy and resource utilisation of ResNet50 with templates on GTZAN and ESC-50 audio classification datasets. Results show average of three repetitions.

Metric	base	Filter Templates				
		a	b	c	d	e
GTZAN Accuracy	85.59 ± 1.04	85.92 ± 0.57	87.26 ± 2.52	85.75 ± 2.03	**87.43** ± 1.51	85.08 ± 0.76
ESC-50 Accuracy	69.66 ± 0.87	**70.33** ± 2.26	68.16 ± 0.72	65.75 ± 1.88	69.91 ± 1.89	67.25 ± 0.75
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**
GFLOPs	4.119	4.102	4.093	4.076	4.118	4.089

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 chapter, 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.

Something worth noticing is that, again, there is no absolute winner template in this task of audio classification. Instead, each particular template seems to provide some advantage even between datasets.

5.4 Templates 2.0 on NASBench 101 Dataset

The aim of neural architecture search (NAS) is to find the best possible CNN model on a specific dataset by iterating over a set of model candidates looking for the best performance [164]. The size of the search space is the most challenging aspect of architecture search. An extensive evaluation of all networks is generally costly. Therefore, most existing methods use some heuristic to predict the final performance of each model with just a few epochs (or none if possible) of training.

The process is described in Figure 5.6. It starts with defining the space of all possible architectures and follows by sampling architectures from that space to estimate their accuracies which can update the exploration strategy.

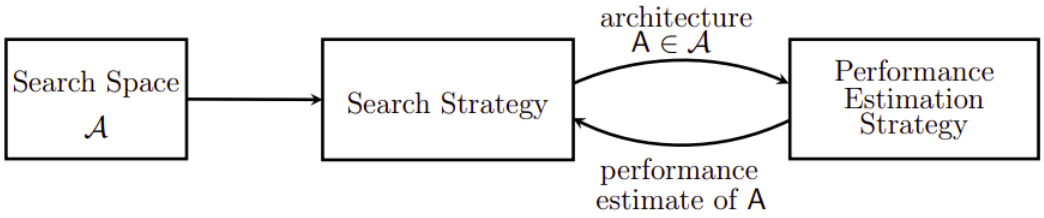


Figure 5.6: Schematics of Neural Architecture Search methods. From a pre-defined search space an exploration strategy selects an architecture. An heuristic computes an estimate of the model performance. Then, the exploration strategy can be updated depending on the estimations. Image from [49].

Research in NAS is divided into the three areas described in the boxes of Figure 5.6: search space definition, exploration strategy and performance estimation strategy. Despite numerous improvements in efficiency and performance, empirical evaluations in NAS continue to be a challenge. Different NAS studies frequently employ different training pipelines, search spaces, and hyperparameters. Thus they do not compare other approaches in similar conditions [210, 211].

NASBench-101 [207] was introduced to provide a common framework to evaluate new proposed exploration and performance estimation strategies. 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.

5.4.1 Implementation Details

More than a dataset, 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 our 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.

5.4.2 Results

It would be inaccessible to us, as well as to many researchers, to test our templates with each architecture within the exploration space of NASBench. We instead evaluated and compared the best model in the dataset. Other singular model mentioned in the NASBench paper is a ResNet-like network. So we decide to test the templates on these two architectures.

Models in the 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 5.5. 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 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 an impressive reduction of computational costs. Template *b* with both models uses one-fifth of the original parameters. Template *c* obtain a higher accuracy than the best performing model with one-third of the parameters. We are not aiming for our method to outperform any NAS method. We state that our method can be used in combination with NAS methods to obtain further improvements 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.

Table 5.5: 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 similar FLOPs. Results show average of three repetitions.

			Templates				
Models		base	a	b	c	d	e
Best architecture	Acc (avg)	95.35	95.20	95.02	**95.44**	95.06	95.26
	Acc (std)	0.1855	0.2688	0.1552	0.3847	0.1345	0.5902
	Param	32.42	27.49	**6.44**	10.38	17.26	8.73
	GFLOPs	3664	3629	3662	3545	3562	3567
ResNet like architecture	Acc (avg)	92.64	**93.85**	91.80	92.65	91.81	92.67
	Acc (std)	0.3807	0.7145	0.2804	0.2502	0.5727	0.3442
	Param	6.04	5.18	**1.24**	1.79	3.30	1.63
	GFLOPs	687	684	685	602	679	665

5.5 Templates 2.0 on Representation and Localisation

Until now we have implemented and tested our templates on domains where we evaluate the final output of the network. We want to know if the internal representation of the models using templates is as good as the representation delivered by the original pyramidal distribution. This original distribution puts more filters on the last layer so, the representation encodes a big number of features. By using templates the representation varies from a few features at the end (templates c and d) to medium range (b and e). Despite template a follows a similar distribution to the original one, the number of filters in the final layer is lower. We propose to evaluate templates in a task that uses the internal representation and furthermore, applies the representation to a different domain than the classification task. We have chosen to replicate experiments published in [173] on localisation using embeddings.

5.5.1 Geolocalisation Embedding Maps and Images

This particular geolocalisation task consists of matching panoramic images with points in a high-level 2-D map in order to find the geographic coordinates where the images were taken. The task assumes no GPS service is available, so the localisation relies only on visual elements in the input images.

The method proposed in [173] finds the best match by linking the semantic informa-

Figure 5.7: Sample location image and map. A georeferenced panorama (left) is divided in four viewing directions (centre) and processed with the correspondant location tile from the map (right). The arrow in the map indicates the heading direction. Image from [173].

tion in both the map and the image in the same fashion as humans do. This is a different approach from most geolocalisation methods where an image is encoded in a vector and then compared to a large dataset of already georeferenced images. Maps are divided into tiles of equal size, and the process jointly learns a low dimensional embedded vector for the corresponding image and map tile pairs.

A training example is shown in Figure 5.7. A panorama is divided into four images with a predefined heading direction. A tile of a configurable range is generated from the whole map centred in the same location matching the orientation of the image. On inference, an agent trying to self-localise produces an input image that is embedded in some vector space. A search process looks in the dataset of all available locations, which are also encoded in the same embedding space. Given that encoded locations are georeferenced, the system is able to retrieve the absolute geographical coordinates of the input image.

5.5.2 Dataset and Model

Georeferenced images for training are taken from the StreetLearn dataset [136] which contains 113,767 panoramic images from Manhattan and Pittsburgh while maps tiles are obtained from Open Street Map [69]. With the geographic coordinates from images, authors generate corresponding map tiles using Mapnik [149]. We used for testing the same three subsets from locations in Hudson River (HR), Union Square (US) and Wall Street (WS), each including 5,000 points within areas of $3.25km^2$, $2.77km^2$ and $2.33km^2$, respectively. We train with the same set of the original work obtained from the remaining locations of the dataset. The total size of the training set included 98,767 images associated with two map tiles each of different scales ($152x152m^2$ and $76x76m^2$).

The proposed architecture is built with two ResNet subnetworks following a siamese-

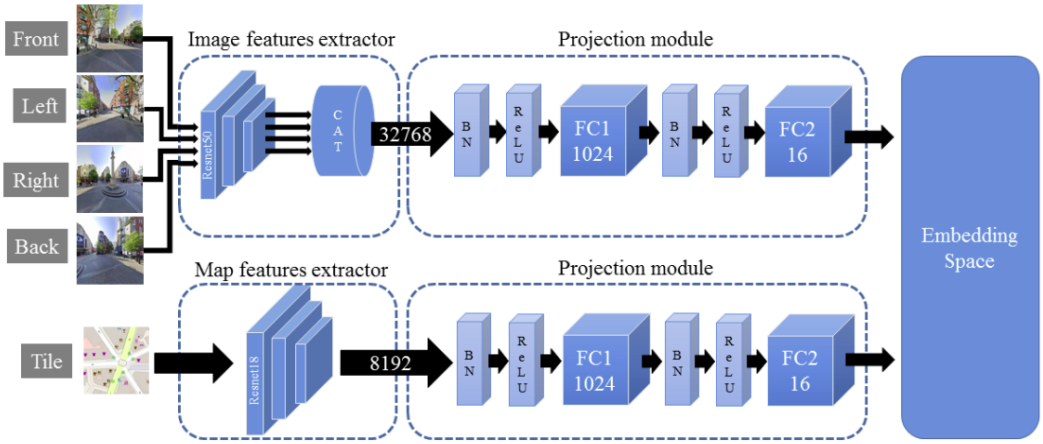


Figure 5.8: Network architecture for embedded space learning. Input panoramic images are divided into four images and then fed to a ResNet50 network. Map tiles are encoded by a ResNet18 network. Both networks are forced to produce close representation embeddings. Image from [173].

like pattern. We illustrate the block design in Figure 5.8. The authors argue that encoding images is a more complex task (and a different domain) than encoding maps. Consequently, they decided to use two independent networks, a ResNet50 model to process images into the embedding space and a ResNet18 model to process maps. Both models produce 512 local 4x4 descriptors that enter a projection module consisting of two fully connected layers. These projection modules reduce the descriptors dimension to 16 and couple the domains in the same embedding space.

We modified the original ResNet subnetworks with each of our five templates, adjusting the last layers to match the number of descriptors required for the projection models. As in each experiment in this chapter, all models produced by templates matched the FLOPs of the original subnetworks.

5.5.3 Implementation Details

Similarly to the work in [173], we trained the whole model end-to-end, updating parameters from all modules at the same time using the provided triplet loss. The network was trained 10 epochs with a learning rate of 0.00004 for the Adam optimiser. Again, augmentation was limited to small variations on the scale of map tiles and the orientation of images.

Differences to the original work lie in the datasets used to pre-train ResNet subnetworks. ResNet50 was originally pre-trained on Places365 [219], but our available

Table 5.6: Accuracy and resource utilisation of ResNet50 with templates on geolocalisation embedding datasets. Table shows one-repetition results.

		Filter Templates				
Metric	base	a	b	c	d	e
Top 1% Recall HR	77.28	**79.92**	76.12	76.64	78.82	74.00
Top 1% Recall US	79.80	79.38	75.16	79.08	**80.24**	74.22
Top 1% Recall WS	69.88	69.70	66.28	68.80	**70.28**	64.28
Param (Millions)	11.17	6.60	2.14	**1.60**	3.90	1.76
Param Change (%)	-	-40.9	-80.8	**-85.6**	-65.0	-84.2
Memory (MB)	66.46	49.28	34.78	**32.88**	38.91	33.52
Mem Change (%)	-	-25.8	-47.6	**-50.5**	-41.4	-49.5
Inference (ms)	8.07	6.42	4.63	4.36	5.39	**3.93**
Inf Change (%)	-	-20.4	-42.5	-45.9	-33.2	**-51.2**

ResNet50 models were pre-trained on Imagenet, so we used those. ResNet18 was pre-trained on ImageNet, but we noticed that starting it from random weights yielded similar results to the published values.

5.5.4 Results

We compared the system using the base ResNet architectures versus our template-generated networks evaluating the quality of the embedded space with recall@k as in information retrieval systems [9]. Particularly, we adopt the same recall at 1% metric to the baseline work.

Results are shown in Table 5.6. Base performances are comparable to the published work and present similar variations due to different degrees of complexity in the sets. Templates show an improved top-1% recall for the three evaluation sets and still produce greater benefits in the resource demands. With a higher recall, template d produces savings in parameters up to 65% and template a up to 40%. Although template c slightly compromises the performance, the reductions in parameters (85%) and memory (50%) can widely compensate for the loss. Template e provides an alternative for applications optimising speed with a reduction in inference time of more than 50 percentage points.

This experimental result in embedding spaces suggests that the internal representation differs in models using distinct distributions of filters. Moreover, models with the pyramidal pattern of filters cause overuse of resources that can be alleviated by simply changing the distribution. We revisit embedding spaces further in the next section to provide more insights into the quality of the representation produced by templates.

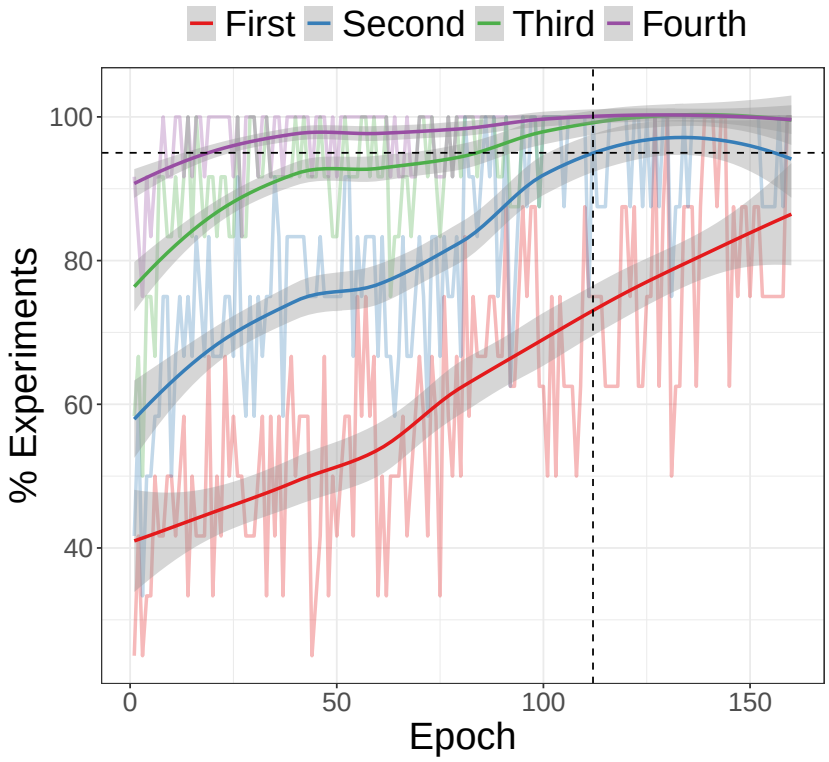


Figure 5.9: Percentage of runs in which the best final template, considering accuracy, was at least in the first positions at each epoch. A horizontal dotted line marks 95% of all experiments. The vertical dotted line shows the epoch in which 95% percent of cases were in the first two positions.

5.6 Finding the Best Template

Templates have obtained better performances than base pyramidal models in many previously presented tasks. Even though some templates tend to produce high results or significantly reduce computational resources, there is no single template that improves in every case. Training the five templates might still be cheaper than performing an exhaustive model search [223]. However, for more extensive datasets, it is desirable to have a way to predict which would be a good candidate template before full training or even with no training. Methods try to achieve this estimation by using lower fidelities (a.k.a proxy metrics) of the actual full training performance [49]. One naive approach is to use performances obtained from early training steps [111, 162, 215] nevertheless, the task of performance estimation has been found to be difficult [49].

To illustrate the complexity, we present in Figure 5.9 a summary of all our exper-

iments on image classification with the first version of templates. The plot shows the percentage in which the best final template was also the best (or at least one of the best) at some previous epoch considering accuracy. We need to train all the templates up to 2/3 of the training process and then fully train the best two to find the best template in 95 per cent of cases. To choose a subset at 1/4 of the epochs that 95 per cent of the time includes the best final template, we need to train four templates out of five.

Before trying to predict the final accuracy by using early performances or other more sophisticated methods [48, 96, 121], we decided to analyse the properties of the representations spaces generated by applying the different templates. We firstly make a qualitative exploration using an algorithm for dimension reduction. Later, we test a metric for measuring similarities between representation spaces looking for a correlation between the representation and the final performance distances of the different templates.

5.6.1 Embedding Space of Templates

The area researching techniques for visualising and understanding large, high dimensional data is known as dimensionality reduction [196]. Methods in this field rely on the existence of a smaller intrinsic dimension of the data [55].

For many years the standard method for dimensionality reduction was principal component analysis (PCA) which transform data points into a subspace generated with the first principal components that maximise the variance of the projected data. Later on, t-Distributed Stochastic Neighbor Embedding (t-SNE) was proved to be a more effective method by modelling a distribution that grants a high probability to similar points to be close in the reduced representation [195]. However, one of the disadvantages of t-SNE is that it does not preserve well the global structure of the data. Furthermore, it requires tuning several hyperparameters to produce a meaningful plot [202].

Uniform Manifold Approximation and Projection (UMAP) is a new approach that proposes a number of advantages over t-SNE, such as boosted speed and better conservation of the data's global structure [132].

UMAP follows an approach similar to t-SNE in the sense that they first build an initial neighbourhood graph in the dimensional space of the original data, and later, they try to find a similar graph in a reduced dimensional space.

On the other hand, UMAP and t-SNE differ in the way the initial graph is constructed. UMAP extends a circular region from each point to determine its connectivity. Circles are defined with variable radii depending on the density of the data points within the

zone, using a long radius in low-density regions and a small radius in populated ones. Instead of directly estimating the radius of the circle, UMAP uses the distance from the point to the kth nearest neighbour. k is a hyperparameter that controls the trade-off between global and local structure preservation. To complete the final graph, UMAP weights each edge of connecting points with the distance between them. This procedure allows finding a low-dimensional graph where close points look tight and remote points stay far.

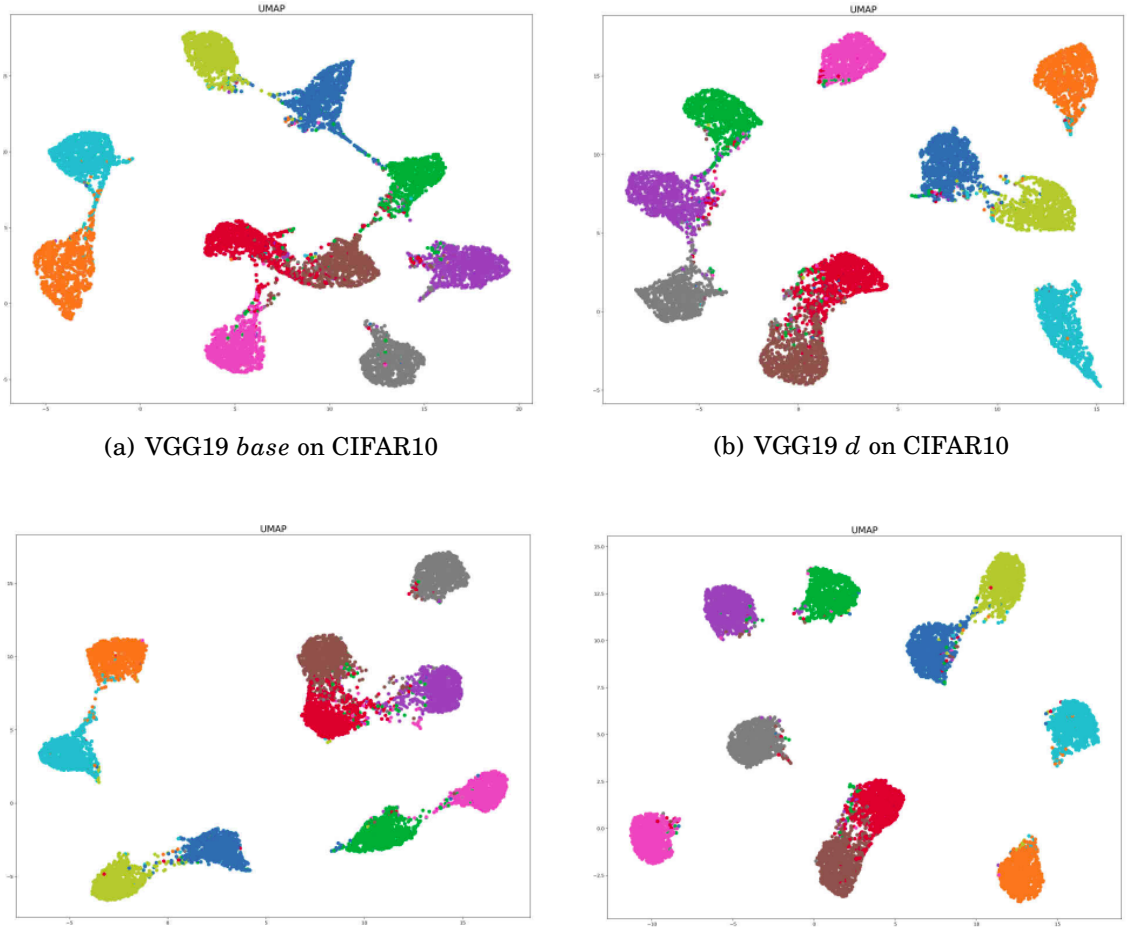
Based on its improvements over other techniques, we decided to use UMAP to compare the representation spaces of base models and templates. We particularly analysed the space projected by the final layer of VGG19 and ResNet50 models on the CIFAR10 dataset. Figure 5.10 show a 2-dimensional representation of the base model compared against the representation obtained with template d. This template reached the highest accuracy on CIFAR10. We observe that for the VGG architecture, the template d distribution space features a clearer separation between classes than the original $base$ space. The separation is more evident for the ResNet architecture, presumably because the accuracies of ResNet models are higher than those of the VGG models. There are less than half mixed boundaries in ResNet with template d than in $base$ model.

We present a comparison of projected representation spaces in Figure 5.11 for the rest of the datasets used for testing templates on the image classification task. We exclusively compare ResNet50 architectures with the original $base$ distribution versus ones with template d distribution.

On the CINIC10 dataset, clusters are tight and difficult to separate. The effect is caused by the high number of samples comprising the dataset. Still, we observe more "fuzzy" class boundaries with the $base$ distribution, meaning that the model struggles more to classify a sample. As the number of classes in the dataset increases, UMAP embeddings are more difficult to analyse. For TinyImagenet, we find more compact clusters in template d than in the base model.

5.6.2 Comparing Representation Spaces of Templates Via CKA Metric

We have discussed in the section 5.6.1 the qualitative properties of the UMAP two-dimensional projections of the base model compared to templates. The analysis suggests that the embedding spaces generated for the templates are better than the base model in the sense that they tend to build clearer boundaries that facilitate the task of image



(a) VGG19 *base* on CIFAR10

(b) VGG19 d on CIFAR10

(c) ResNet50 *base* on CIFAR10

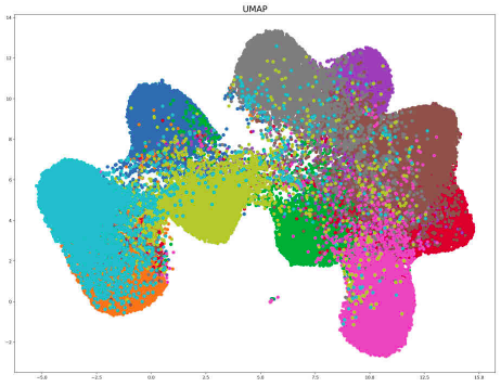
(d) ResNet50 d on CIFAR10

Figure 5.10: UMAP embeddings for the final layer of VGG19 and ResNet50 on CIFAR10. Template d distribution space (b, d) shows a more clear separation between classes than the original *base* space (a, c). Colours in different plots represent the same classes.

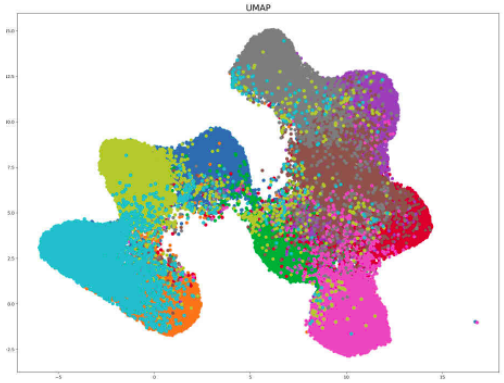
classification. We also found that templates obtain higher performances in other tasks such as audio classification and map localisation.

Additionally, we want to quantitatively analyse those embedding spaces (for the base model and all templates) and their differences to find good markers in the embedding space or, as we study in section 5.6.3, a correlation between differences and accuracies on classification, as an intermediate step to find the best template. The intuition behind this idea is the best templates could show similar properties disregarding the domain or the model. So, we could constrain the space in the training procedure to acquire these desirable properties.

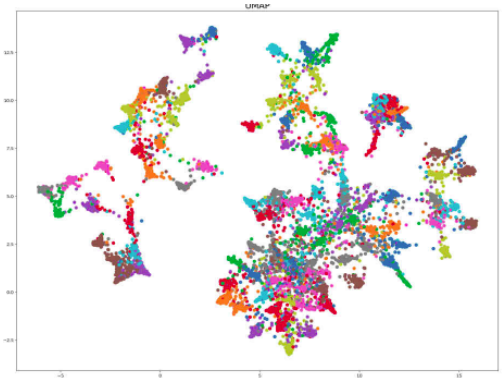
To that end, we rely on the ability of similarity metrics for comparing neural network



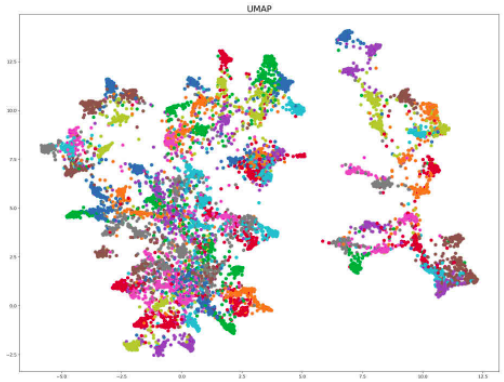
(a) Resnet50 (base) on CINIC10



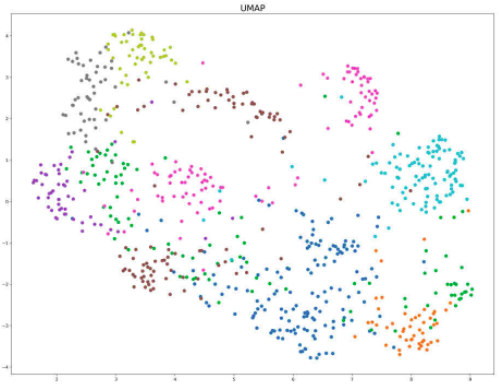
(b) Resnet50 (d) on CINIC10



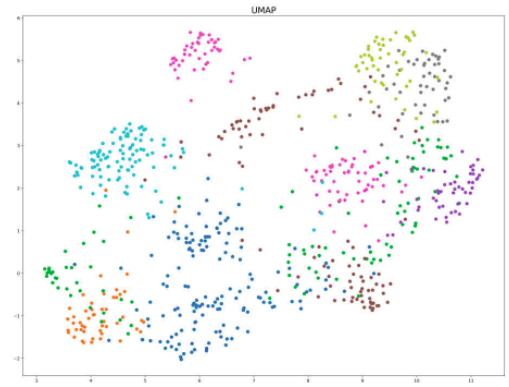
(c) Resnet50 (base) on CIFAR100



(d) Resnet50 (d) on CIFAR100



(e) Resnet50 (base) on 15 classes from Tiny-Imagenet



(f) Resnet50 (d) on 15 classes from Tiny-Imagenet

Figure 5.11: UMAP embeddings for the final layer of ResNet50 on several datasets. Maps for *base* distribution are placed on the left column and maps for d distribution are in the right column.

representations [157]. A following work presents improvements that allow a better comparison of layers with different numbers of neurons using Canonical Correlation Analysis (CCA). The property of this metric to be invariant to affine transforms enables its use without requiring any neuron to neuron alignment [141].

A recent technique, known as Centered Kernel Alignment (CKA), to measure similarity is proposed in [98]. The authors aim that the technique, unlike the previous ones, provides meaningful metrics between representations where there are more dimensions than data points. Furthermore, CKA is more invariant to different initialisations when training the models. Although there has been a lot of discussion around the universal dominance of a particular similarity metric, we chose the latter to analyse templates' embedding spaces because it performs well when comparing representation spaces in vision tasks [42].

Several explorations and analyses have been presented recently using the CKA similarity metric between models. However, authors compare other properties of different architectures such as wider versus deeper models [146], CNNs versus Transformers [158] or self-supervised versus supervised trained models [66]. In our experiments, we analyse another aspect of the networks: the variation of representation across layers for each different distribution of filters in the same architecture.

We computed the CKA metric using the Hilbert-Schmidt Independence Criterion (HSIC) according to [98, 146] describing the problem as below.

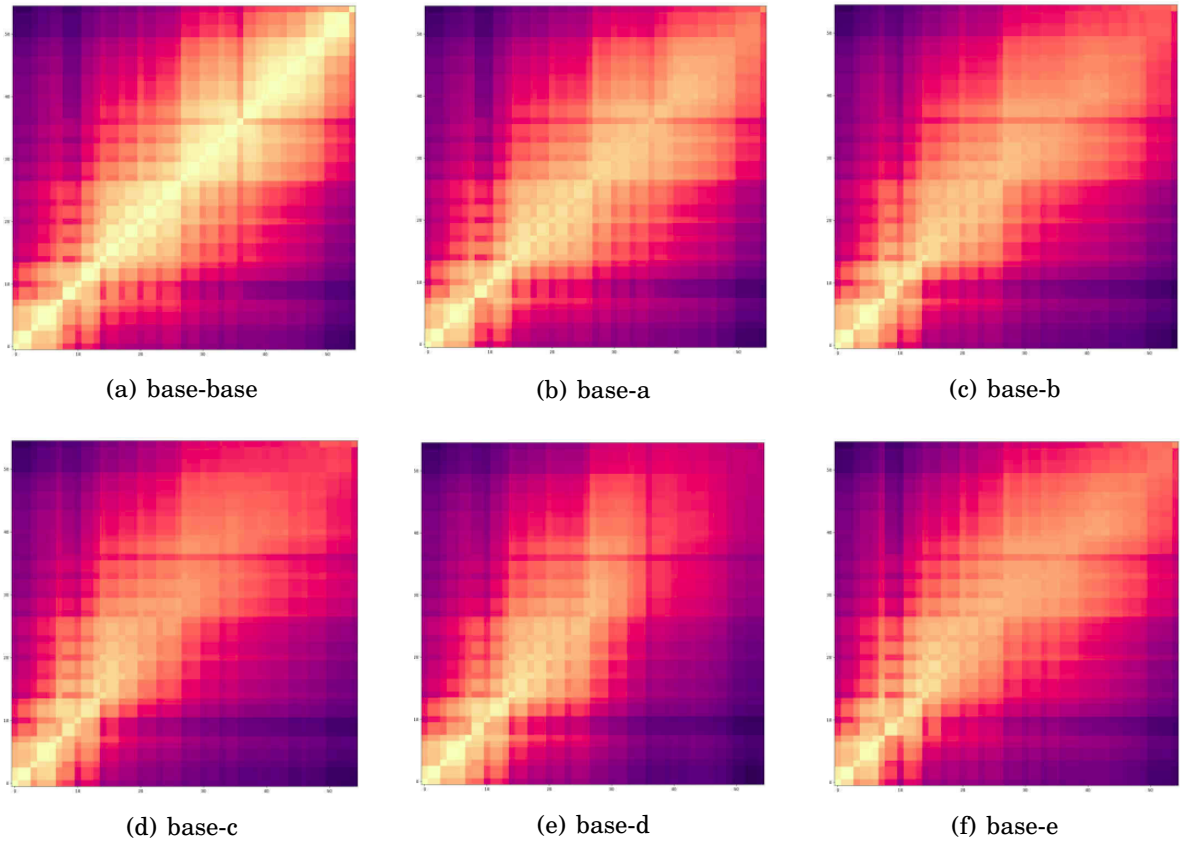
For a pair of layers with p_1 neurons and p_2 neurons, let $X \in R^{mxp_1}$ and $Y \in R^{mxp_2}$ contain their representations of a set of m examples. The mxm elements of Gramian matrices $K = XX^T$ and $L = YY^T$ denotes the similarities between a pair of examples according to the representations in X or Y. Let $H = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ be the centering matrix. The empirical estimator of HSIC is:

$$HSIC(K,L) = \frac{1}{(1-n)^2)} tr(KHLH)$$

HSIC is invariant to orthogonal transformations and, therefore, to permutation of neurons. To make it invariant to scaling, CKA normalises HSIC to produce a value between 0 and 1 given by

$$CKA(K,L) = \frac{HSIC(K,L)}{\sqrt{HSIC(K,K)HSIC(L,L)}}$$

This CKA metric estimates the similarity of a pair of layers of the same or different models. To produce a CKA plot for a whole model, we iterate the computation of CKA with each pair of layers from that model to be compared.



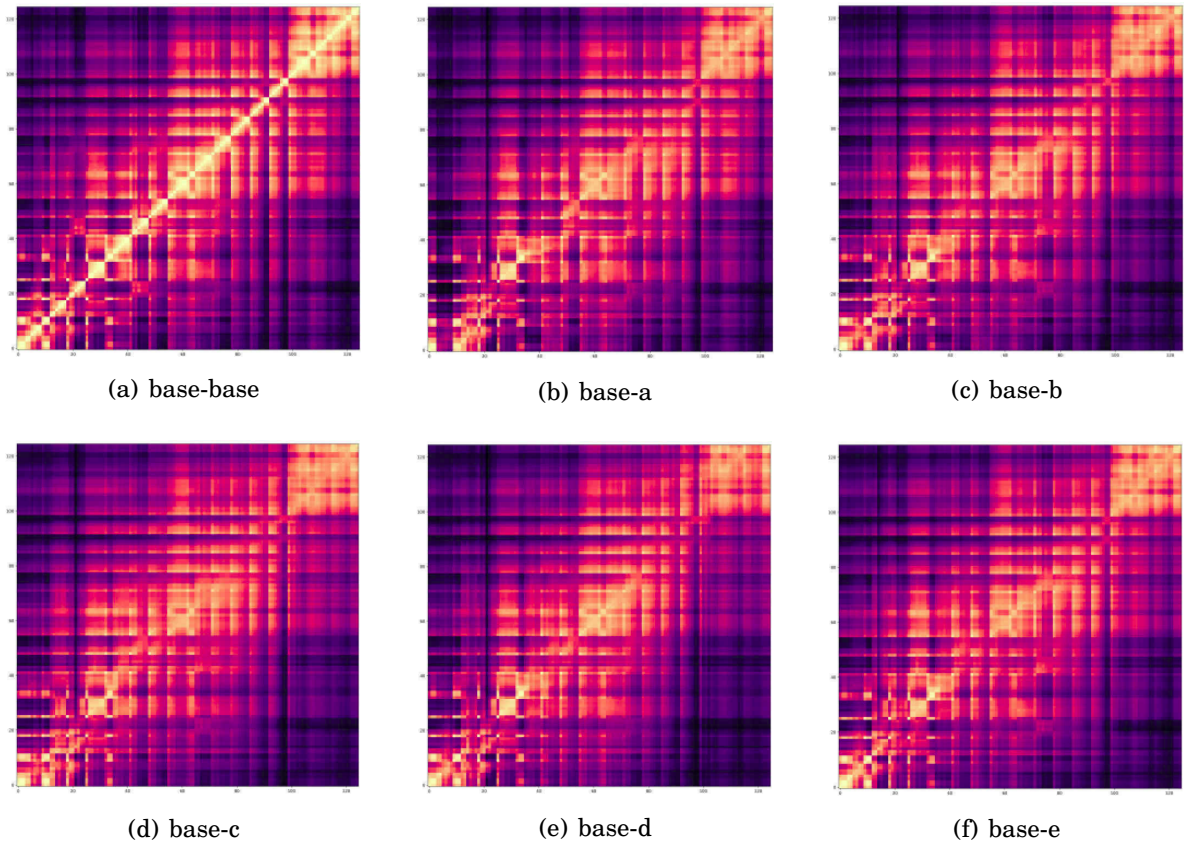
(a) base-base (b) base-a (c) base-b

(d) base-c (e) base-d (f) base-e

Figure 5.12: CKA metric of VGG19 base model vs templates with tiny-imagenet dataset. Abscissas represent each layer in the base model and ordinates show layers in the model obtained from a template.

We presented the CKA similarity of the base model compared with those produced by templates from the VGG19 architecture in Figure 5.12. For reference, we show the CKA metric of the base model against itself in the top-left plot. The bright diagonals reflect total similarity, given that we are measuring the same layer and model. Representations in the first layers are found to be different from the ones from the final layer (in perpendicular corners with respect to the bright diagonal).

There are different degrees of similarity when comparing templates and the base model depending on the dataset in which CKA is computed. For tinyImagenet, all models show a high similarity from the initial to middle layers independently that they have a variable number of neurons. It is in the last layer where the representation spaces of templates start to differ from the base model representation space. Interestingly, the least similar representation to the one produced by the base model is that from template d, which is the model with the highest accuracy on the tinyImagenet dataset.



(a) base-base (b) base-a (c) base-b

(d) base-c (e) base-d (f) base-e

Figure 5.13: CKA metric of ResNet50 base model vs templates with tiny-imagenet dataset. Abscissas represent each layer in the base model and ordinates show layers in the model obtained from a template.

Similar plots are shown in Figure 5.13 for the ResNet50 architecture on tinyImagenet. The top-left plot shows the CKA similarity of the same base model across layers. Visual analysis is challenging given the big number of layers of the model and the dissimilarity of even (post-residual) and odd (block interior) layers [98]. However, we found a distinctive feature in the base-template a CKA metric denoted by the dark bands on the edges of the plot: the first layers of the base model representation are very different from the rest of the layers from the template a which was the best performing model in the dataset.

5.6.3 An Attempt to Correlate CKA Measurements with Accuracy and Parameters

Motivated by the results of the section 5.6.2, we hypothesise about the existence of a correlation between the differences in performance of models and their similarity via

the CKA metric. Going further, we wonder if there exists an ideal CKA similarity plot (comparing the model against itself) that a good model should show.

Suppose we can find this ideal pattern of similarity between layers of the same network. In that case, we may be able to induce the pattern at training time to produce higher-accuracy models. An example of such a hypothesised CKA pattern could present a smooth variance in the similarities of the representations as we go deeper into the layers. So, we explored the correlation of accuracies and CKA similarities between all templates. Because templates produce large variations in the number of parameters of models, we also investigated the existence of a correlation between parameters and CKA similarities.

CKA is a similarity metric for a pair of layers. The final result of comparing pairs of models is, consequently, a matrix of CKA values of $l_1 x l_2$ elements where l_1 and l_2 are the number of layers in each model to be compared. To reduce the $CKA_{l_1 x l_2}$ matrix to a single value to be computed in the correlation, we decided to use two methods: taking the mean of the values (CKA_{mean}) and taking the standard deviation (CKA_{std}).

We also tested the correlation on the models' CKA plot resemblance to the best performing model CKA pattern and to, what we think, could be an ideal CKA pattern. We measure the likeness of two CKA plots using the mean squared error (MSE). For clarity, we called MSE_{best} the computed difference of the CKA matrices between the compared template and the best performing template. We called MSE_{ideal} the difference with our ideal CKA matrix.

We show in tables 5.7 the correlation for VGG19 and ResNet50 models in four of the most used datasets in this work (CIFAR10, CIFAR100, CINIC10 and tinyImagenet). It appears like there is a strong correlation for some of the attributes, such as accuracy and CKA_{mean}, or accuracy and MSE_{best} testing VGG19 on CIFAR100. However, the correlation values are not consistent on the other datasets. Moreover, the correlation values are not consistent for ResNet50 nor among the datasets tested with ResNet50.

We also tried to find if the difference in parameters of models affects the differences between representations measured by CKA. Similar findings to the accuracy can be seen for parameters in Table 5.8. Parameters and CKA_{mean} looks promising, reaching high correlation values for ResNet50 in three datasets. But very low values for VGG19 in the same datasets. In general, correlation values in all attributes of VGG19 are low.

Looking at the correlation for MSE_{best} with high and low values, we could argue that the differences in parameters, which it is believed affect the representation in some way, are not captured by the CKA metric. We instead think that there is no strong

Table 5.7: Spearman correlation between template CKA properties and accuracy for ResNet50 and VGG19. CKA mean and std are obtained from summarising CKA metric of all x all layers in the same model. MSE best difference is computed comparing each template with the best accuracy pattern. MSE ideal is compared with a proposed ideal gradient pattern.

VGG19	mean, acc	std, acc	mse best, acc	mse ideal, acc
cifar10	-0.68	0.36	-0.82	0.32
cifar100	-0.96	0.43	-0.93	0.75
cinic10	-0.68	0.46	-0.54	0.61
tiny-imagenet	-0.57	0.64	-0.68	-0.64

ResNet50	mean, acc	std, acc	mse best, acc	mse ideal, acc
cifar10	0.46	0.21	-0.68	-0.50
cifar100	-0.11	0.07	-0.43	0.00
cinic10	0.54	-0.11	-0.75	0.11
tiny-imagenet	-0.18	0.50	-0.18	0.71

Table 5.8: Spearman correlation between template CKA properties and parameters for ResNet50 and VGG19. CKA mean and std are obtained from summarising CKA metric of all x all layers in the same model. MSE best difference is computed by comparing each template with the best accuracy pattern. MSE ideal is compared with a proposed ideal gradient pattern.

VGG19	mean, param	std, param	mse best, param	mse ideal, param
cifar10	-0.36	-0.26	0.80	0.69
cifar100	-0.44	0.58	0.98	0.58
cinic10	-0.36	0.36	0.40	0.07
tiny-imagenet	0.36	0.11	-0.26	0.26

ResNet50	mean, param	std, param	mse best, param	mse ideal, param
cifar10	-0.86	0.21	0.86	0.96
cifar100	-0.86	0.89	0.75	0.86
cinic10	-0.71	0.86	0.64	0.43
tiny-imagenet	-0.18	0.50	-0.18	0.71

correlation between parameters and the effectiveness of a representation for a particular architecture. Using the templates, we redistributed filters in several different ways producing models with a diverse number of parameters. We found that the highest performance was not always obtained for the model counting the highest number of parameters.

5.7 Conclusion

This chapter presented a new definition of templates named *Templates 2.0*. The new set improves over the previous templates in that the definition allows matching a specific budget of FLOPs which makes fair comparisons with the original models. A noticeable difference from the previous design is that the filter distribution for each template is parameterised with linear segments providing smooth changes in the number of feature maps. This smoothness has proved to be beneficial in obtaining a higher accuracy related to the original architecture. With templates, the VGG model obtained improvements up to 2.11 points in accuracy for image classification tasks. Templates can also produce models with reductions of 90% in parameters, 79% in memory usage and 22% in inference time. Templates obtained higher accuracies with highly optimised models for all the tested datasets, except tiny-Imagenet. With MobileNet V2, templates are able of reducing up to 77% parameters and 11% memory footprint. With MNASNet, templates produced architectures 74.5% smaller in parameters and 4.7% lower inference time.

We extended the experiments to new domains where templates were evaluated for the final output and the intermediate representations they build. In the audio classification datasets, using templates on ResNet50 reached improvements in accuracy of 0.96% for the GTZAN dataset and 2.14% for the ESC-50 dataset. Templates offer reductions in memory footprint and inference time by around 15%, while the number of parameters shows a considerable decrease of 65%.

We showed a relevant example of the benefits of templates. Applying them to the best architecture in the NASBench-101 dataset produced a model with higher accuracy using only one-third of the original parameters. By using one-fifth of the original parameters, a different template obtained only 0.33 fewer points in accuracy.

In the task of localisation utilising embeddings, we changed the model backbone with modified versions using templates. The new models showed increases in the three evaluated datasets with 65%, 41% and 33% fewer parameters, memory and inference time demands, respectively.

The work in section 5.6 takes a step back. We do not offer a way of finding the best template. Our experiments exploring the representation spaces of templates with the CKA similarity metric suggest that the task is especially complex and indicates further experimentation and analysis. We consider that this complexity is the cause of some of the drawbacks of NAS methods, such as lack of convergence and inaccurate model performance prediction.

CHAPTER 6

CONCLUSIONS

This thesis challenges the universality of the pyramidal design in convolutional neural networks. We introduce the idea of taking an existing model and changing its original distribution of filters with a small set of diverse patterns that we call templates. We performed experiments with several models on different domains, showing that original architectures are generally susceptible to performing more efficiently when using the distributions proposed in this work.

6.1 Findings

This section summarises the findings of each chapter within this thesis.

In chapter 3 we introduced the concept of templates to defy the standard incremental design for distributing filters existing in many CNN architectures.

The experimental results on CIFAR-10, CIFAR-100 and Tiny-Imagenet datasets with four popular convolutional models showed that a simple redistribution of the same number of filters could improve the accuracies over the original pyramidal design. For CIFAR-10, models increased up to 1.83 points in accuracy. At the same time, for CIFAR-100 and Tiny-Imagenet, templates were effective for VGG, Inception and MobileNet, reaching improvements of up to 4.88, 1.07 and 3.63, respectively (Table 3.1).

A second experiment varying the size of models with a width multiplier found templates produce more efficient models in terms of resource demands with up to 85% fewer

parameters and a memory footprint up to 76% smaller (Figures 3.3 and 3.4).

In chapter 4 experiments imply that benefits from templates can extend to domains other than image classification. Furthermore, we described how templates could work on top of CNN compression techniques to obtain further improvements.

Using PoseNet, the first CNN model to carry out real-time camera pose estimation from a single image, we found templates were effective in more than half of the sets evaluated in the 7-scenes dataset (Table 4.2). In addition, the improved PoseNet had a lower MSE while producing 8% less memory footprint (Table 4.6). For the Cambridge Landmarks dataset, the modified PoseNet and MobileNetV1 models had fewer localisation errors than the original models in six out of seven scenes (Table 4.5). The top MobileNetV1 template outperformed the original architecture despite utilising 10% fewer parameters. Experiments revealed a singular condition in the super-resolution task with VGG and ResNet backbones. All templates caused increases in resource requirements, and only one template had a higher PNSR than the original model (Tables 4.8 and 4.7). Interestingly, researchers are not using the incremental filter distribution as default but a uniform distribution with a constant number of filters per layer, such as one of the proposed templates. The uniform distribution had already led to a more efficient use of resources in this task.

Regarding comparing model compression methods, the experimental evidence with VGG models on CIFAR-100 suggests that the pruning methods can obtain superior models with higher accuracy using similar resources when pruning from models with different filter distributions. For example, the Geometric Median technique enhanced 3.87 points of accuracy in VGG when pruning 50% from the *reverse-base* template and still accomplishing a small decrease in parameters. The Gate Decorator approach discovered a model, again with the *reverse-base* template, that performed more than one point higher in accuracy with FLOPs close to the original. Filter Basis discovered a model, using the *negative-quadratic* template, 5.3 points better in accuracy than starting with an incremental distribution (Table 4.11).

We compared MorphNet and templates using several neural networks and datasets. Using templates alone or MorphNet and templates improved accuracy in eight out of nine pairs of model-dataset (Table 4.16). Particularly, templates only were superior in six out of nine cases, increasing MobileNetV1 accuracy up to 6.2 points for CIFAR100 (Table 4.15).

Chapter 5 introduced a new definition of templates that allows matching a predefined number of FLOPs with no significant overheating in the search process. An expanded set of experiments with templates were performed in more demanding domains to evaluate their representation capability. The chapter attempted to find a correlation between the internal representation, as seen by the CKA similitude metric, and the template with the best accuracy.

The filter distributions in the new templates were built with linear segments, giving smooth variations in the number of feature maps between layers. The idea introduced in the PyramidNet paper achieves better accuracy than the quadratic-based distributions. The VGG model reached more than 2.11 points higher in accuracy when using templates. Templates also produced models with up to 90 per cent fewer parameters, 79 per cent less memory, and 22 per cent less inference time (Table 5.1). Except for tiny-Imagenet, all of the datasets studied yielded greater accuracies when templates were used with exhaustively tuned models (Table 5.3). A possible explanation for the unsatisfactory results in Tiny-Imagenet is, as mentioned in chapter 2, that architectures with the existing incremental filter distribution have been tuned to perform well in the Imagenet dataset. Therefore, the pattern also fits the derived tiny-ImageNet dataset. Templates with MobileNet V2 templates reduced parameters by up to 77 per cent. Templates with MNASNet resulted in architectures that were 74.5 per cent smaller in parameters (Table 5.1).

We performed an expanded series of experiments evaluating templates in other areas where intermediate representations of models are important. Templates applied to ResNet improved accuracy by 0.96 per cent for the GTZAN dataset and 2.14 per cent for the ESC-50 dataset, decreasing parameters by 65% (Table 5.4). When templates were applied to the highest accuracy model in the NASBench-101 dataset, the resulting network increased accuracy, requiring one-third of the base model parameters (Table 5.5). This is a notable outcome considering we only tested five templates. In the task of mapping and localisation using representation embeddings, models obtained with templates showed reductions up to 65 per cent fewer parameters, 41 per cent less memory footprint and 33 per cent improved inference time (Table 5.6).

Finally, we showed that the representation produced for the models trained with different templates differs from each other (Figure 5.10). So we use the CKA similitude metric to find a correlation between the template with the best accuracy and its internal variation captured via CKA (Table 5.7). Our outcomes using CKA to explore the representation spaces of templates imply that the task is challenging and calls for more

exploration and analysis. We believe that several shortcomings of NAS techniques, such as lack of convergence and erroneous model performance prediction, are caused for this complexity.

6.2 General Advice to Future Deep Learning Practitioners

Overall, templates enhanced performances and reduced resource demands for the models and domains we used. 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.

It is essential to note that a smaller number of parameters does not imply a lower memory usage or a faster inference time. Differences in feature map resolution for filters in different layers, the necessity to store early feature maps in memory for further processing in deeper layers, and hardware and software limits in the parallelisation procedure are some of the causes.

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.

6.3 Future Work

We envision several opportunities for further research in this work.

Finding the best template. We explored in chapter 5 a way to correlate several measures obtained from the between-layer CKA similarity of a model produced with a template and its final validation accuracy. The experiments showed no correlation for the values we tested. Although the number of templates is low, an automatic method for finding a suitable template could either accelerate the identification of an efficient model or explore other patterns to the ones defined in the templates.

Explore the idea of hard-coded representation compression. The information bottleneck (IB) principle [180, 191] found that most of the training process in deep learning is spent on compressing the input to efficient representation that helps generalisation and not on fitting the training labels. Experimental results in this thesis glimpse a tendency to templates with filters in final layers matching the number of classes of the task to perform better. This pattern extends to CNN models explicitly designed for ImageNet, defining the last layers with a large number of filters. We hypothesise that the effort of the CNN to compress the representation described by IB can be reduced by hard-coding the compression in the dimensionality induced by the number of filters. A positive answer to this question could facilitate the choice of filters for each specific task and speed up training.

We highlight that there is much work to do to create algorithms that find the optimal distribution of filters for a given model and task. However, we expect the community to be aware of a new exploration space opened by the described templates that help refine better models.

This thesis offers insights to model designers, both automated and manual, to construct more efficient models by introducing new distributions of filters in the exploration space for neural network model search. In addition, we hope this work helps gather data to better understand the design process of model-task pairs and inspires the re-think of assumptions on model building that are normally given for granted.

APPENDIX A

A.1 Filters in Tested Models with Templates 2.0

In order to facilitate 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 the case of ResNet we set the value of filters at the level of each layer inside residual modules, as presented in Table A.4. VGG design consists of simple layers so we change filters in each of them (Table A.1). For the rest of architectures, we set the value of filters at the level of modules (Tables A.2 and A.3). Last layers of all models are fully connected ones and the number of neurons is imposed by the dataset. We add the schematics of the templates as a visual aid in figure A.1.

Table A.1: 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

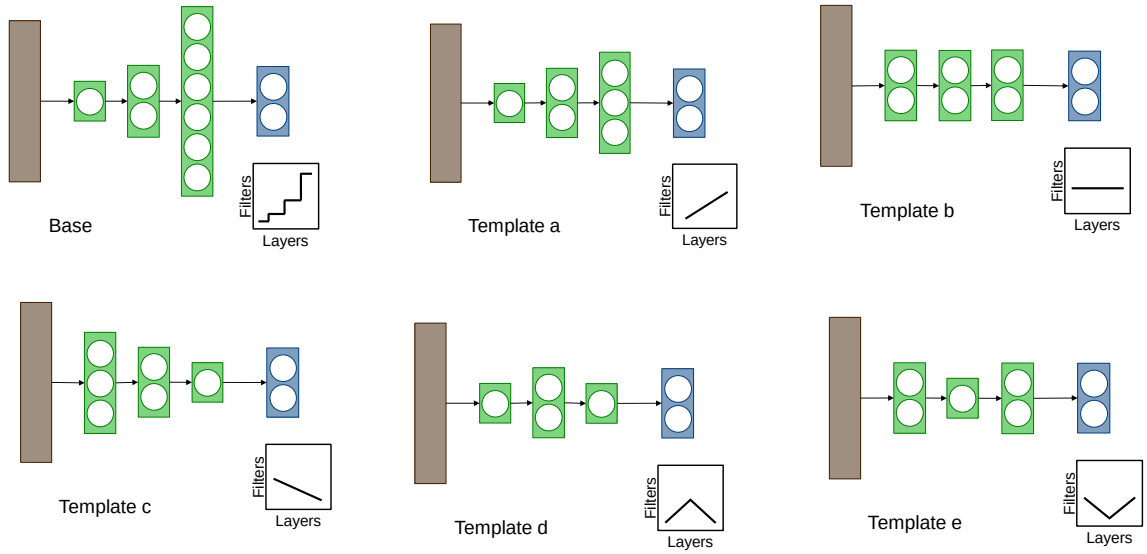


Figure A.1: Schematic distribution of filters per layer in templates. Base distribution is the original pyramidal distribution. Templates follow a smoother transition in the number of filters between layers. Number of filters does not match between models but they are adjusted to fit the original number of FLOPS of the base model.

Table A.2: 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 A.3: 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 A.4: 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]]

APPENDIX B

APPENDIX B

This appendix is a perspective of the challenges lived from my particular point of view. I hope the content helps the future me comprehend the decisions I took, serves future studies about the effects of the COVID-19 pandemic, and helps better cope with them.

When someone decides to study for a doctorate, the person visualises possible complications that may arise during its development. They are seen as obstacles or challenges, but they can never be an impediment to achieving the goal of completing it.

B.1 A Reflection on the COVID-19 Pandemic

Unexpectedly, in December 2019, the appearance of the COVID-19 disease in Wuhan, China, was announced. Its rapid spread worldwide and seriousness represented a public health risk. As a result, governments established restrictions and changes in all areas of social and individual life. Those restrictions generated unusual situations and forced everyday activities to be adjusted to continue with a different and uncertain rhythm.

Each nation made decisions based on the severity of the health emergency and the country's economic conditions. As a result, some items stopped being produced, and a few others increased the production. In addition, outdoor activities were interrupted, including working in offices and attending a face-to-face education, to name a few.

As a doctoral student, and combined with the status of father of a family, the challenges were not easy. Still, at the same time, it was a decisive moment because it brought

various situations to attend to and analyse, both from my own research and from my family life. Fortunately, at every moment, I was supported by the indications of the University and the recommendations of my advisor.

Starting from March 2020, we lived in exceptional circumstances. COVID-19 lockdown disrupted my studies significantly due to university facilities closure, added responsibilities at home such as homeschooling, and technical difficulties in accessing the computational resources needed for my research.

But not only my studies were affected. As a non-native english speaker, I enormously benefited from having daily chats with my lab colleagues. With the university facilities closed, I stopped improving my communication skills. We had some meetings online during the first moments of the pandemic. Still, they never replaced the richness of the face-to-face discussions.

Many of the students suffered from physical and mental stress, caused in some way for being isolated and impeded from participating in social life and recreation. In my case, I was not significantly affected by losing physical closeness with my family as we remained together. On the other hand, there was a perceptible remoteness with friends. Our group used to have frequent activities in the sports centres at the University. Unfortunately, all of the centres were closed during the first months. Despite rules allowing people to exercise, this had to be done in an isolated way.

After several months of missing the educational, cultural and social benefits of living in a foreign country and studying at a top university, some students, including myself, decided to return to our home countries, continuing our research at home.

Not everything was negative during the pandemic. Our periodical seminars, which normally included speakers from places near our location, expanded their scope to researchers around the world. Also, educational systems for remote and blended learning were significantly improved, providing more and better tools.

The preceding circumstances have left a mark on our life. Unfortunately, we did not have control of the unpredictable situations produced by the pandemic. Yet, we must use the experience in the future to recover and rethink the path and the strategies allowing us to move objectively and correct the adversities that the COVID-19 pandemic caused in its way.

BIBLIOGRAPHY

[1] M. ABADI, P. BARHAM, J. CHEN, Z. CHEN, A. DAVIS, J. DEAN, M. DEVIN, S. GHEMAWAT, G. IRVING, M. ISARD, ET AL., *Tensorflow: A system for large-scale machine learning*, in 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 2016, pp. 265–283.

[2] A. F. AGARAP, *Deep learning using rectified linear units (relu)*, arXiv preprint arXiv:1803.08375, (2018).

[3] E. AGUSTSSON AND R. TIMOFTE, *Ntire 2017 challenge on single image super-resolution: Dataset and study*, in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2017, pp. 126–135.

[4] A. ALKHULAIFI, F. ALSAHLI, AND I. AHMAD, *Knowledge distillation in deep learning and its applications*, PeerJ Computer Science, 7 (2021).

[5] Z. ALLEN-ZHU AND Y. LI, *Towards understanding ensemble, knowledge distillation and self-distillation in deep learning*, arXiv preprint arXiv:2012.09816, (2020).

[6] N. ALOYSIUS AND M. GEETHA, *A review on deep convolutional neural networks*, in 2017 International Conference on Communication and Signal Processing (ICCSP), IEEE, 2017, pp. 0588–0592.

[7] L. ALZUBAIDI, J. ZHANG, A. J. HUMAIDI, A. AL-DUJAILI, Y. DUAN, O. AL-SHAMMA, J. SANTAMARÍA, M. A. FADHEL, M. AL-AMIDIE, AND L. FARHAN, *Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions*, Journal of big Data, 8 (2021), pp. 1–74.

[8] R. ANIL, G. PEREYRA, A. PASSOS, R. ORMANDI, G. E. DAHL, AND G. E. HINTON, *Large scale distributed neural network training through online distillation*, arXiv preprint arXiv:1804.03235, (2018).

[9] M. ARORA, U. KANJILAL, AND D. VARSHNEY, *Evaluation of information retrieval: precision and recall*, International Journal of Indian Culture and Business Management, 12 (2016), pp. 224–236.

[10] M. AUGASTA AND T. KATHIRVALAVAKUMAR, *Pruning algorithms of neural networks—a comparative study*, Open Computer Science, 3 (2013), pp. 105–115.

[11] L. J. BA AND R. CARUANA, *Do deep nets really need to be deep?*, arXiv preprint arXiv:1312.6184, (2013).

[12] B. BAKER, O. GUPTA, N. NAIK, AND R. RASKAR, *Designing neural network architectures using reinforcement learning*, arXiv preprint arXiv:1611.02167, (2016).

[13] E. B. BAUM AND D. HAUSSLER, *What size net gives valid generalization?*, Neural computation, 1 (1989), pp. 151–160.

[14] Y. BENGIO, *Deep learning of representations for unsupervised and transfer learning*, in Proceedings of ICML workshop on unsupervised and transfer learning, JMLR Workshop and Conference Proceedings, 2012, pp. 17–36.

[15] M. BERMAN, L. PISHCHULIN, N. XU, M. B. BLASCHKO, AND G. MEDIONI, *Aows: Adaptive and optimal network width search with latency constraints*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11217–11226.

[16] D. BLALOCK, J. J. G. ORTIZ, J. FRANKLE, AND J. GUTTAG, *What is the state of neural network pruning?*, arXiv preprint arXiv:2003.03033, (2020).

[17] A. BLUMER, A. EHRENFEUCHT, D. HAUSSLER, AND M. K. WARMUTH, *Learnability and the vapnik-chervonenkis dimension*, Journal of the ACM (JACM), 36 (1989), pp. 929–965.

[18] A. BORJI AND L. ITTI, *State-of-the-art in visual attention modeling*, IEEE transactions on pattern analysis and machine intelligence, 35 (2012), pp. 185–207.

[19] Y.-L. BOUREAU, J. PONCE, AND Y. LECUN, *A theoretical analysis of feature pooling in visual recognition*, in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 111–118.

[20] J. S. BRIDLE, *Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition*, in Neurocomputing, Springer, 1990, pp. 227–236.

[21] A. BROCK, T. LIM, J. M. RITCHIE, AND N. WESTON, *Smash: one-shot model architecture search through hypernetworks*, arXiv preprint arXiv:1708.05344, (2017).

[22] C. BUCILUǍ, R. CARUANA, AND A. NICULESCU-MIZIL, *Model compression*, in Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 535–541.

[23] Y. CHEBOTAR AND A. WATERS, *Distilling knowledge from ensembles of neural networks for speech recognition.*, in Interspeech, 2016, pp. 3439–3443.

[24] C. CHEN, B. WANG, C. X. LU, N. TRIGONI, AND A. MARKHAM, *A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence*, arXiv preprint arXiv:2006.12567, (2020).

[25] J. CHEN AND X. RAN, *Deep learning with edge computing: A review*, Proceedings of the IEEE, 107 (2019), pp. 1655–1674.

[26] T. CHEN, J. FRANKLE, S. CHANG, S. LIU, Y. ZHANG, Z. WANG, AND M. CARBIN, *The lottery ticket hypothesis for pre-trained bert networks*, Advances in neural information processing systems, 33 (2020), pp. 15834–15846.

[27] W. CHEN, X. GONG, AND Z. WANG, *Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective*, arXiv preprint arXiv:2102.11535, (2021).

[28] J. CHENG, Y.-H. TSAI, S. WANG, AND M.-H. YANG, *Segflow: Joint learning for video object segmentation and optical flow*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 686–695.

[29] F. CHOLLET, *Xception: Deep learning with depthwise separable convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.

[30] J. L. CHU AND A. KRZYŻAK, *Analysis of feature maps selection in supervised learning using convolutional neural networks*, in Canadian Conference on Artificial Intelligence, Springer, 2014, pp. 59–70.

[31] Y. CI, C. LIN, M. SUN, B. CHEN, H. ZHANG, AND W. OUYANG, *Evolving search space for neural architecture search*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 6659–6669.

[32] D. C. CIRESAN, U. MEIER, J. MASCI, L. MARIA GAMBARDELLA, AND J. SCHMIDHUBER, *Flexible, high performance convolutional neural networks for image classification*, in IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, Barcelona, Spain, 2011, p. 1237.

[33] R. CLARK, S. WANG, A. MARKHAM, N. TRIGONI, AND H. WEN, *Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6856–6864.

[34] L. CONTRERAS AND W. MAYOL-CUEVAS, *O-poco: Online point cloud compression mapping for visual odometry and slam*, in 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 4509–4514.

[35] ——, *Towards cnn map representation and compression for camera relocalisation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2018, pp. 292–299.

[36] C. CORTES, X. GONZALVO, V. KUZNETSOV, M. MOHRI, AND S. YANG, *Adanet: Adaptive structural learning of artificial neural networks*, in International conference on machine learning, PMLR, 2017, pp. 874–883.

[37] M. COŞKUN, Ö. YILDIRIM, U. AYŞEGÜL, AND Y. DEMIR, *An overview of popular deep learning methods*, European Journal of Technique, 7 (2017), pp. 165–176.

[38] L. N. DARLOW, E. J. CROWLEY, A. ANTONIOU, AND A. J. STORKEY, *Cinic-10 is not imagenet or cifar-10*, arXiv preprint arXiv:1810.03505, (2018).

[39] M. DENIL, B. SHAKIBI, L. DINH, M. RANZATO, AND N. DE FREITAS, *Predicting parameters in deep learning*, arXiv preprint arXiv:1306.0543, (2013).

[40] E. L. DENTON, W. ZAREMBA, J. BRUNA, Y. LECUN, AND R. FERGUS, *Exploiting linear structure within convolutional networks for efficient evaluation*, in Advances in neural information processing systems, 2014, pp. 1269–1277.

[41] T. DEVRIES AND G. W. TAYLOR, *Improved regularization of convolutional neural networks with cutout*, arXiv preprint arXiv:1708.04552, (2017).

[42] F. DING, J.-S. DENAIN, AND J. STEINHARDT, *Grounding representation similarity with statistical testing*, arXiv preprint arXiv:2108.01661, (2021).

[43] C. DONG, C. C. LOY, K. HE, AND X. TANG, *Image super-resolution using deep convolutional networks*, IEEE transactions on pattern analysis and machine intelligence, 38 (2015), pp. 295–307.

[44] X. DONG AND Y. YANG, *Network pruning via transformable architecture search*, arXiv preprint arXiv:1905.09717, (2019).

[45] ——, *Nas-bench-201: Extending the scope of reproducible neural architecture search*, arXiv preprint arXiv:2001.00326, (2020).

[46] A. DOSOVITSKIY, L. BEYER, A. KOLESNIKOV, D. WEISSENBORN, X. ZHAI, T. UNTERTHINER, M. DEHGHANI, M. MINDERER, G. HEIGOLD, S. GELLY, ET AL., *An image is worth 16x16 words: Transformers for image recognition at scale*, arXiv preprint arXiv:2010.11929, (2020).

[47] E. ELSEN, M. DUKHAN, T. GALE, AND K. SIMONYAN, *Fast sparse convnets*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 14629–14638.

[48] T. ELSKEN, J. H. METZEN, AND F. HUTTER, *Efficient multi-objective neural architecture search via lamarckian evolution*, arXiv preprint arXiv:1804.09081, (2018).

[49] ——, *Neural architecture search: A survey*, The Journal of Machine Learning Research, 20 (2019), pp. 1997–2017.

[50] J. FANG, Y. SUN, Q. ZHANG, Y. LI, W. LIU, AND X. WANG, *Densely connected search space for more flexible neural architecture search*, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10628–10637.

[51] J. FRANKLE AND M. CARBIN, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, arXiv preprint arXiv:1803.03635, (2018).

[52] J. FRANKLE, G. K. DZIUGAITE, D. M. ROY, AND M. CARBIN, *Stabilizing the lottery ticket hypothesis*, arXiv preprint arXiv:1903.01611, (2019).

[53] ——, *Pruning neural networks at initialization: Why are we missing the mark?*, arXiv preprint arXiv:2009.08576, (2020).

[54] T. FUKUDA, M. SUZUKI, G. KURATA, S. THOMAS, J. CUI, AND B. RAMABHADRAN, *Efficient knowledge distillation from an ensemble of teachers.*, in Interspeech, 2017, pp. 3697–3701.

[55] K. FUKUNAGA, *Introduction to statistical pattern recognition*, Elsevier, 2013.

[56] K. FUKUSHIMA, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological cybernetics, 36 (1980), pp. 193–202.

[57] T. GALE, E. ELSEN, AND S. HOOKER, *The state of sparsity in deep neural networks*, arXiv preprint arXiv:1902.09574, (2019).

[58] T. GANCHEV, N. FAKOTAKIS, AND G. KOKKINAKIS, *Comparative evaluation of various mfcc implementations on the speaker verification task*, in Proceedings of the SPECOM, vol. 1, 2005, pp. 191–194.

[59] S. GIRISH, S. R. MAIYA, K. GUPTA, H. CHEN, L. S. DAVIS, AND A. SHRIVASTAVA, *The lottery ticket hypothesis for object recognition*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 762–771.

[60] B. GLOCKER, S. IZADI, J. SHOTTON, AND A. CRIMINISI, *Real-time rgb-d camera relocalization*, in Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on, IEEE, 2013, pp. 173–179.

[61] X. GLOROT AND Y. BENGIO, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[62] Y. GONG, Y.-A. CHUNG, AND J. GLASS, *Ast: Audio spectrogram transformer*, arXiv preprint arXiv:2104.01778, (2021).

[63] I. GOODFELLOW, Y. BENGIO, AND A. COURVILLE, *Deep learning*, MIT press, 2016.

[64] A. GORDON, E. EBAN, O. NACHUM, B. CHEN, H. WU, T.-J. YANG, AND E. CHOI, *Morphnet: Fast & simple resource-constrained structure learning of deep net-*

works, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1586–1595.

[65] J. GOU, B. YU, S. J. MAYBANK, AND D. TAO, *Knowledge distillation: A survey*, International Journal of Computer Vision, 129 (2021), pp. 1789–1819.

[66] T. G. GRIGG, D. BUSBRIDGE, J. RAMAPURAM, AND R. WEBB, *Do self-supervised and supervised methods learn similar visual representations?*, arXiv preprint arXiv:2110.00528, (2021).

[67] A. GUZHOV, F. RAUE, J. HEES, AND A. DENGEL, *Esresnet: Environmental sound classification based on visual domain models*, in 2020 25th International Conference on Pattern Recognition (ICPR), IEEE, 2021, pp. 4933–4940.

[68] G. GWARDYS AND D. M. GRZYWCZAK, *Deep image features in music information retrieval*, International Journal of Electronics and Telecommunications, 60 (2014), pp. 321–326.

[69] M. HAKLAY AND P. WEBER, *Openstreetmap: User-generated street maps*, IEEE Pervasive computing, 7 (2008), pp. 12–18.

[70] D. HAN, J. KIM, AND J. KIM, *Deep pyramidal residual networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5927–5935.

[71] S. HAN, X. LIU, H. MAO, J. PU, A. PEDRAM, M. A. HOROWITZ, AND W. J. DALLY, *Eie: Efficient inference engine on compressed deep neural network*, ACM SIGARCH Computer Architecture News, 44 (2016), pp. 243–254.

[72] K. HE, X. ZHANG, S. REN, AND J. SUN, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[73] ——, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[74] ——, *Identity mappings in deep residual networks*, in European conference on computer vision, Springer, 2016, pp. 630–645.

[75] Y. HE, P. LIU, Z. WANG, Z. HU, AND Y. YANG, *Filter pruning via geometric median for deep convolutional neural networks acceleration*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.

[76] Y. HE, X. ZHANG, AND J. SUN, *Channel pruning for accelerating very deep neural networks*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[77] J. F. HENRIQUES AND A. VEDALDI, *Mapnet: An allocentric spatial memory for mapping environments*, in proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8476–8484.

[78] G. HINTON, O. VINYALS, J. DEAN, ET AL., *Distilling the knowledge in a neural network*, arXiv preprint arXiv:1503.02531, 2 (2015).

[79] G. E. HINTON, N. SRIVASTAVA, A. KRIZHEVSKY, I. SUTSKEVER, AND R. R. SALAKHUTDINOV, *Improving neural networks by preventing co-adaptation of feature detectors*, arXiv preprint arXiv:1207.0580, (2012).

[80] S. HOCHREITER, *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6 (1998), pp. 107–116.

[81] M.-F. HONG, H.-Y. CHEN, M.-H. CHEN, Y.-S. XU, H.-K. KUO, Y.-M. TSAI, H.-J. CHEN, AND K. JOU, *Network space search for pareto-efficient spaces*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 3053–3062.

[82] A. G. HOWARD, M. ZHU, B. CHEN, D. KALENICHENKO, W. WANG, T. WEYAND, M. ANDREETTO, AND H. ADAM, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv preprint arXiv:1704.04861, (2017).

[83] J. HU, L. SHEN, AND G. SUN, *Squeeze-and-excitation networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[84] G. HUANG, Z. LIU, L. VAN DER MAATEN, AND K. Q. WEINBERGER, *Densely connected convolutional networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[85] G. HUANG, Y. SUN, Z. LIU, D. SEDRA, AND K. Q. WEINBERGER, *Deep networks with stochastic depth*, in European conference on computer vision, Springer, 2016, pp. 646–661.

[86] F. N. IANDOLA, S. HAN, M. W. MOSKEWICZ, K. ASHRAF, W. J. DALLY, AND K. KEUTZER, *Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size*, arXiv preprint arXiv:1602.07360, (2016).

[87] S. IOFFE AND C. SZEGEDY, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International conference on machine learning, PMLR, 2015, pp. 448–456.

[88] L. ITTI AND C. KOCH, *Computational modelling of visual attention*, Nature reviews neuroscience, 2 (2001), pp. 194–203.

[89] J. JIN, A. DUNDAR, AND E. CULURCIELLO, *Flattened convolutional neural networks for feedforward acceleration*, arXiv preprint arXiv:1412.5474, (2014).

[90] J. JOHNSON, A. ALAHI, AND L. FEI-FEI, *Perceptual losses for real-time style transfer and super-resolution*, in European conference on computer vision, Springer, 2016, pp. 694–711.

[91] S. KALE AND R. SHRIRAM, *Suspicious activity detection using transfer learning based resnet tracking from surveillance videos*, in International Conference on Soft Computing and Pattern Recognition, Springer, 2020, pp. 208–220.

[92] A. KENDALL AND R. CIPOLLA, *Modelling uncertainty in deep learning for camera relocalization*, in 2016 IEEE international conference on Robotics and Automation (ICRA), IEEE, 2016, pp. 4762–4769.

[93] A. KENDALL, M. GRIMES, AND R. CIPOLLA, *Posenet: A convolutional network for real-time 6-dof camera relocalization*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 2938–2946.

[94] A. KHAN, A. SOHAIL, U. ZAHOORA, AND A. S. QURESHI, *A survey of the recent architectures of deep convolutional neural networks*, Artificial Intelligence Review, 53 (2020), pp. 5455–5516.

[95] J. KIM, J. K. LEE, AND K. M. LEE, *Accurate image super-resolution using very deep convolutional networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 1646–1654.

[96] A. KLEIN, S. FALKNER, J. T. SPRINGENBERG, AND F. HUTTER, *Learning curve prediction with bayesian neural networks*, in ICLR, 2017.

[97] Q. KONG, Y. CAO, T. IQBAL, Y. WANG, W. WANG, AND M. D. PLUMBLEY, *Panns: Large-scale pretrained audio neural networks for audio pattern recognition*, IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28 (2020), pp. 2880–2894.

[98] S. KORNBLITH, M. NOROUZI, H. LEE, AND G. HINTON, *Similarity of neural network representations revisited*, in International Conference on Machine Learning, PMLR, 2019, pp. 3519–3529.

[99] A. KRIZHEVSKY, G. HINTON, ET AL., *Learning multiple layers of features from tiny images*, tech. rep., Citeseer, 2009.

[100] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[101] A. F. KURI-MORALES, *The best neural network architecture*, in Mexican International Conference on Artificial Intelligence, Springer, 2014, pp. 72–84.

[102] G. LARSSON, M. MAIRE, AND G. SHAKHNAROVICH, *Fractalnet: Ultra-deep neural networks without residuals*, arXiv preprint arXiv:1605.07648, (2016).

[103] Y. LE AND X. YANG, *Tiny imagenet visual recognition challenge*, CS 231N, 7 (2015), p. 7.

[104] G. LECLERC, M. VARTAK, R. C. FERNANDEZ, T. KRASKA, AND S. MADDEN, *Smallify: Learning network size while training*, arXiv preprint arXiv:1806.03723, (2018).

[105] Y. LECUN, B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD, AND L. D. JACKEL, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[106] Y. LECUN, L. BOTTOU, Y. BENGIO, P. HAFFNER, ET AL., *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[107] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal brain damage*, in Advances in neural information processing systems, 1990, pp. 598–605.

[108] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al., *Photo-realistic single image super-resolution using a generative adversarial network*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4681–4690.

[109] E. Lee and C.-Y. 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, 2020, pp. 1478–1487.

[110] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient convnets*, arXiv preprint arXiv:1608.08710, (2016).

[111] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, *Hyperband: A novel bandit-based approach to hyperparameter optimization*, The Journal of Machine Learning Research, 18 (2017), pp. 6765–6816.

[112] T. Li, J. Li, Z. Liu, and C. Zhang, *Few sample knowledge distillation for efficient network compression*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14639–14647.

[113] Y. Li, *Deep reinforcement learning: An overview*, arXiv preprint arXiv:1701.07274, (2017).

[114] Y. Li, S. Gu, L. V. Gool, and R. Timofte, *Learning filter basis for convolutional neural network compression*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 5623–5632.

[115] Y. Li, Z. Hao, and H. Lei, *Survey of convolutional neural network*, Journal of Computer Applications, 36 (2016), pp. 2508–2515.

[116] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, *Pruning and quantization for deep neural network acceleration: A survey*, Neurocomputing, 461 (2021), pp. 370–403.

[117] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, *Tune: A research platform for distributed model selection and training*, arXiv preprint arXiv:1807.05118, (2018).

[118] B. LIM, S. SON, H. KIM, S. NAH, AND K. MU LEE, *Enhanced deep residual networks for single image super-resolution*, in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2017, pp. 136–144.

[119] M. LIN, Q. CHEN, AND S. YAN, *Network in network*, arXiv preprint arXiv:1312.4400, (2013).

[120] 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), 2018, pp. 19–34.

[121] H. LIU, K. SIMONYAN, AND Y. YANG, *Darts: Differentiable architecture search*, arXiv preprint arXiv:1806.09055, (2018).

[122] Y. LIU, K. CHEN, C. LIU, Z. QIN, Z. LUO, AND J. WANG, *Structured knowledge distillation for semantic segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2604–2613.

[123] Y. LIU, Y. SUN, B. XUE, M. ZHANG, G. G. YEN, AND K. C. TAN, *A survey on evolutionary neural architecture search*, IEEE transactions on neural networks and learning systems, (2021).

[124] Z. LIU, M. SUN, T. ZHOU, G. HUANG, AND T. DARRELL, *Rethinking the value of network pruning*, arXiv preprint arXiv:1810.05270, (2018).

[125] B. LOGAN, *Mel frequency cepstral coefficients for music modeling*, in In International Symposium on Music Information Retrieval, Citeseer, 2000.

[126] C. LOUIZOS, M. WELLING, AND D. P. KINGMA, *Learning sparse neural networks through l_0 regularization*, arXiv preprint arXiv:1712.01312, (2017).

[127] S. LOUSSAIEF AND A. ABDELKRIM, *Convolutional neural network hyperparameters optimization based on genetic algorithms*, International Journal of Advanced Computer Science and Applications, 9 (2018), pp. 252–266.

[128] J.-H. LUO, J. WU, AND W. LIN, *Thinet: A filter level pruning method for deep neural network compression*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 5058–5066.

[129] E. MALACH, G. YEHUDAI, S. SHALEV-SCHWARTZ, AND O. SHAMIR, *Proving the lottery ticket hypothesis: Pruning is all you need*, in International Conference on Machine Learning, PMLR, 2020, pp. 6682–6691.

[130] K. MARKOV AND T. MATSUI, *Robust speech recognition using generalized distillation framework.*, in Interspeech, 2016, pp. 2364–2368.

[131] W. S. MCCULLOCH AND W. PITTS, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics, 5 (1943), pp. 115–133.

[132] L. MCINNES, J. HEALY, AND J. MELVILLE, *Umap: Uniform manifold approximation and projection for dimension reduction*, arXiv preprint arXiv:1802.03426, (2018).

[133] I. MELEKHOV, J. YLIOINAS, J. KANNALA, AND E. RAHTU, *Image-based localization using hourglass networks*, in Proceedings of the IEEE international conference on computer vision workshops, 2017, pp. 879–886.

[134] H. MENG, Z. LIN, F. YANG, Y. XU, AND L. CUI, *Knowledge distillation in medical data mining: A survey*, in 5th International Conference on Crowd Science and Engineering, 2021, pp. 175–182.

[135] M. MINSKY AND S. A. PAPERT, *Perceptrons: An introduction to computational geometry*, MIT press, 2017.

[136] P. MIROWSKI, A. BANKI-HORVATH, K. ANDERSON, D. TEPLYASHIN, K. M. HERMANN, M. MALINOWSKI, M. K. GRIMES, K. SIMONYAN, K. KAVUKCUOGLU, A. ZISSERMAN, ET AL., *The streetlearn environment and dataset*, arXiv preprint arXiv:1903.01292, (2019).

[137] G. MITTAL, C. LIU, N. KARIANAKIS, V. FRAGOSO, M. CHEN, AND Y. FU, *Hyperstar: Task-aware hyperparameters for deep networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 8736–8745.

[138] S. MITTAL, *A survey of fpga-based accelerators for convolutional neural networks*, Neural computing and applications, 32 (2020), pp. 1109–1139.

[139] D. MOLCHANOV, A. ASHUKHA, AND D. VETROV, *Variational dropout sparsifies deep neural networks*, in International Conference on Machine Learning, PMLR, 2017, pp. 2498–2507.

[140] P. MOLCHANOV, S. TYREE, T. KARRAS, T. AILA, AND J. KAUTZ, *Pruning convolutional neural networks for resource efficient inference*, arXiv preprint arXiv:1611.06440, (2016).

[141] A. MORCOS, M. RAGHU, AND S. BENGIO, *Insights on representational similarity in neural networks with canonical correlation*, Advances in Neural Information Processing Systems, 31 (2018).

[142] M. C. MOZER AND P. SMOLENSKY, *Skeletonization: A technique for trimming the fat from a network via relevance assessment*, in Advances in neural information processing systems, 1989, pp. 107–115.

[143] V. NAIR AND G. E. HINTON, *Rectified linear units improve restricted boltzmann machines*, in Icml, 2010.

[144] A. NEWELL, K. YANG, AND J. DENG, *Stacked hourglass networks for human pose estimation*, in European conference on computer vision, Springer, 2016, pp. 483–499.

[145] D. NGUYEN, S. GUPTA, T. NGUYEN, S. RANA, P. NGUYEN, T. TRAN, K. LE, S. RYAN, AND S. VENKATESH, *Knowledge distillation with distribution mismatch*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2021, pp. 250–265.

[146] T. NGUYEN, M. RAGHU, AND S. KORNBLITH, *Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth*, arXiv preprint arXiv:2010.15327, (2020).

[147] K. PALANISAMY, D. SINGHANIA, AND A. YAO, *Rethinking cnn models for audio classification*, arXiv preprint arXiv:2007.11154, (2020).

[148] B. PAN, H. CAI, D.-A. HUANG, K.-H. LEE, A. GAIDON, E. ADELI, AND J. C. NIEBLES, *Spatio-temporal graph for video captioning with knowledge distillation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 10870–10879.

[149] A. PAVLENKO, *Mapnik*, 2016.

[150] B. PENG, X. JIN, J. LIU, D. LI, Y. WU, Y. LIU, S. ZHOU, AND Z. ZHANG, *Correlation congruence for knowledge distillation*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 5007–5016.

[151] L. PEROTIN, A. DÉFOSSEZ, E. VINCENT, R. SERIZEL, AND A. GUÉRIN, *Regression versus classification for neural network based audio source localization*, in 2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), IEEE, 2019, pp. 343–347.

[152] H. PHAM, M. GUAN, B. ZOPH, Q. LE, AND J. DEAN, *Efficient neural architecture search via parameters sharing*, in International conference on machine learning, PMLR, 2018, pp. 4095–4104.

[153] M. PHUONG AND C. LAMPERT, *Towards understanding knowledge distillation*, in International Conference on Machine Learning, PMLR, 2019, pp. 5142–5151.

[154] K. J. PICZAK, *Esc: Dataset for environmental sound classification*, in Proceedings of the 23rd ACM international conference on Multimedia, 2015, pp. 1015–1018.

[155] N. PINTO, D. DOUKHAN, J. J. DICARLO, AND D. D. COX, *A high-throughput screening approach to discovering good forms of biologically inspired visual representation*, PLoS computational biology, 5 (2009), p. e1000579.

[156] H. PURWINS, B. LI, T. VIRTANEN, J. SCHLÜTER, S.-Y. CHANG, AND T. SAINATH, *Deep learning for audio signal processing*, IEEE Journal of Selected Topics in Signal Processing, 13 (2019), pp. 206–219.

[157] M. RAGHU, J. GILMER, J. YOSINSKI, AND J. SOHL-DICKSTEIN, *Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability*, Advances in neural information processing systems, 30 (2017).

[158] M. RAGHU, T. UNTERTHINER, S. KORNBLITH, C. ZHANG, AND A. DOSOVITSKIY, *Do vision transformers see like convolutional neural networks?*, Advances in Neural Information Processing Systems, 34 (2021).

[159] V. RAMANUJAN, M. WORTSMAN, A. KEMBHAVI, A. FARHADI, AND M. RASTEGARI, *What's hidden in a randomly weighted neural network?*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11893–11902.

[160] M. RANZATO, F. J. HUANG, Y.-L. BOUREAU, AND Y. LECUN, *Unsupervised learning of invariant feature hierarchies with applications to object recognition*, in 2007 IEEE conference on computer vision and pattern recognition, IEEE, 2007, pp. 1–8.

[161] W. RAWAT AND Z. WANG, *Deep convolutional neural networks for image classification: A comprehensive review*, Neural computation, 29 (2017), pp. 2352–2449.

[162] E. REAL, A. AGGARWAL, Y. HUANG, AND Q. V. LE, *Aging evolution for image classifier architecture search*, in AAAI conference on artificial intelligence, vol. 3, 2019.

[163] R. REED, *Pruning algorithms-a survey*, IEEE transactions on Neural Networks, 4 (1993), pp. 740–747.

[164] P. REN, Y. XIAO, X. CHANG, P.-Y. HUANG, Z. LI, X. CHEN, AND X. WANG, *A comprehensive survey of neural architecture search: Challenges and solutions*, arXiv preprint arXiv:2006.02903, (2020).

[165] A. RENDA, J. FRANKLE, AND M. CARBIN, *Comparing rewinding and fine-tuning in neural network pruning*, arXiv preprint arXiv:2003.02389, (2020).

[166] E. REZENDE, G. RUPPERT, T. CARVALHO, F. RAMOS, AND P. DE GEUS, *Malicious software classification using transfer learning of resnet-50 deep neural network*, in 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2017, pp. 1011–1014.

[167] F. ROSENBLATT, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review, 65 (1958), p. 386.

[168] F. RUFFY AND K. CHAHAL, *The state of knowledge distillation for classification*, arXiv preprint arXiv:1912.10850, (2019).

[169] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS, *Learning representations by back-propagating errors*, nature, 323 (1986), pp. 533–536.

[170] O. RUSSAKOVSKY, J. DENG, H. SU, J. KRAUSE, S. SATHEESH, S. MA, Z. HUANG, A. KARPATHY, A. KHOSLA, M. BERNSTEIN, ET AL., *Imagenet large scale visual recognition challenge*, International Journal of Computer Vision, 115 (2015), pp. 211–252.

[171] M. SAHU AND R. DASH, *A survey on deep learning: Convolution neural network (cnn)*, in Intelligent and Cloud Computing, Springer, 2021, pp. 317–325.

[172] H. B. SAILOR, D. M. AGRAWAL, AND H. A. PATIL, *Unsupervised filterbank learning using convolutional restricted boltzmann machine for environmental sound classification.*, in InterSpeech, vol. 8, 2017, p. 9.

[173] N. SAMANO, M. ZHOU, AND A. CALWAY, *You are here: Geolocation by embedding maps and images*, in European Conference on Computer Vision, Springer, 2020, pp. 502–518.

[174] M. SANDLER, J. BACCASH, A. ZHMOGINOV, AND A. 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, 2019, pp. 0–0.

[175] M. SANDLER, A. HOWARD, M. ZHU, A. ZHMOGINOV, AND L.-C. CHEN, *Mobilenetv2: Inverted residuals and linear bottlenecks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

[176] K. A. SANKARARAMAN, S. DE, Z. XU, W. R. HUANG, AND T. GOLDSTEIN, *The impact of neural network overparameterization on gradient confusion and stochastic gradient descent*, in International Conference on Machine Learning, PMLR, 2020, pp. 8469–8479.

[177] J. SCHMIDHUBER, *Deep learning in neural networks: An overview*, Neural networks, 61 (2015), pp. 85–117.

[178] J. L. SCHONBERGER AND J.-M. FRAHM, *Structure-from-motion revisited*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 4104–4113.

[179] A. SHRESTHA AND A. MAHMOOD, *Review of deep learning algorithms and architectures*, IEEE Access, 7 (2019), pp. 53040–53065.

[180] R. SHWARTZ-ZIV AND N. TISHBY, *Opening the black box of deep neural networks via information*, arXiv preprint arXiv:1703.00810, (2017).

[181] J. SIEMS, L. ZIMMER, A. ZELA, J. LUKASIK, M. KEUPER, AND F. HUTTER, *Nas-bench-301 and the case for surrogate benchmarks for neural architecture search*, arXiv preprint arXiv:2008.09777, (2020).

[182] L. SIFRE AND S. MALLAT, *Rigid-motion scattering for texture classification*, arXiv preprint arXiv:1403.1687, (2014).

[183] K. SIMONYAN AND A. ZISSERMAN, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[184] M. STOLLENGA, J. MASCI, F. GOMEZ, AND J. SCHMIDHUBER, *Deep networks with internal selective attention through feedback connections*, arXiv preprint arXiv:1407.3068, (2014).

[185] D. STRIGL, K. KOFLER, AND S. PODLIPNIG, *Performance and scalability of gpu-based convolutional neural networks*, in 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE, 2010, pp. 317–324.

[186] B. L. STURM, *A survey of evaluation in music genre recognition*, in International Workshop on Adaptive Multimedia Retrieval, Springer, 2012, pp. 29–66.

[187] C. SZEGEDY, W. LIU, Y. JIA, P. SERMANET, S. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE, AND A. RABINOVICH, *Going deeper with convolutions*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[188] C. SZEGEDY, V. VANHOUCKE, S. IOFFE, J. SHLENS, AND Z. WOJNA, *Rethinking the inception architecture for computer vision*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[189] M. TAN, B. CHEN, R. PANG, V. VASUDEVAN, M. SANDLER, A. HOWARD, AND Q. V. LE, *Mnasnet: Platform-aware neural architecture search for mobile*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[190] S. THRUN ET AL., *Robotic mapping: A survey*, Exploring artificial intelligence in the new millennium, 1 (2002), p. 1.

[191] N. TISHBY AND N. ZASLAVSKY, *Deep learning and the information bottleneck principle*, in 2015 ieee information theory workshop (itw), IEEE, 2015, pp. 1–5.

[192] A. TROCKMAN AND J. Z. KOLTER, *Patches are all you need?*, arXiv preprint arXiv:2201.09792, (2022).

[193] G. TZANETAKIS AND P. COOK, *Musical genre classification of audio signals*, IEEE Transactions on speech and audio processing, 10 (2002), pp. 293–302.

[194] G. URBAN, K. J. GERAS, S. E. KAHOU, O. ASLAN, S. WANG, R. CARUANA, A. MOHAMED, M. PHILIPOSE, AND M. RICHARDSON, *Do deep convolutional nets really need to be deep and convolutional?*, arXiv preprint arXiv:1603.05691, (2016).

[195] L. VAN DER MAATEN AND G. HINTON, *Visualizing data using t-sne.*, Journal of machine learning research, 9 (2008).

[196] L. VAN DER MAATEN, E. POSTMA, J. VAN DEN HERIK, ET AL., *Dimensionality reduction: a comparative*, J Mach Learn Res, 10 (2009), p. 13.

[197] A. VASWANI, N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, Ł. KAISER, AND I. POLOSUKHIN, *Attention is all you need*, Advances in neural information processing systems, 30 (2017).

[198] A. VEIT, M. J. WILBER, AND S. BELONGIE, *Residual networks behave like ensembles of relatively shallow networks*, Advances in neural information processing systems, 29 (2016), pp. 550–558.

[199] F. WANG, M. JIANG, C. QIAN, S. YANG, C. LI, H. ZHANG, X. WANG, AND X. TANG, *Residual attention network for image classification*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3156–3164.

[200] J. WANG, H. BAI, J. WU, X. SHI, J. HUANG, I. KING, M. LYU, AND J. CHENG, *Revisiting parameter sharing for automatic neural channel number search*, Advances in Neural Information Processing Systems, 33 (2020).

[201] Z. WANG, J. CHEN, AND S. C. HOI, *Deep learning for image super-resolution: A survey*, IEEE transactions on pattern analysis and machine intelligence, 43 (2020), pp. 3365–3387.

[202] M. WATTENBERG, F. VIÉGAS, AND I. JOHNSON, *How to use t-sne effectively*, Distill, 1 (2016), p. e2.

[203] G. WU AND S. GONG, *Peer collaborative learning for online knowledge distillation*, in AAAI, 2021.

[204] L. XIE, X. CHEN, K. BI, L. WEI, Y. XU, L. WANG, Z. CHEN, A. XIAO, J. CHANG, X. ZHANG, ET AL., *Weight-sharing neural architecture search: A battle to shrink the optimization gap*, ACM Computing Surveys (CSUR), 54 (2021), pp. 1–37.

[205] S. XIE, R. GIRSHICK, P. DOLLÁR, Z. TU, AND K. HE, *Aggregated residual transformations for deep neural networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

[206] S. XU, A. HUANG, L. CHEN, AND B. ZHANG, *Convolutional neural network pruning: A survey*, in 2020 39th Chinese Control Conference (CCC), IEEE, 2020, pp. 7458–7463.

[207] C. YING, A. KLEIN, E. CHRISTIANSEN, E. REAL, K. MURPHY, AND F. HUTTER, *Nas-bench-101: Towards reproducible neural architecture search*, in International Conference on Machine Learning, PMLR, 2019, pp. 7105–7114.

[208] Z. YOU, K. YAN, J. YE, M. MA, AND P. WANG, *Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks*, in Advances in Neural Information Processing Systems, 2019, pp. 2130–2141.

[209] J. YU AND T. HUANG, *Autoslim: Towards one-shot architecture search for channel numbers*, arXiv preprint arXiv:1903.11728, (2019).

[210] K. YU, R. RANFTL, AND M. SALZMANN, *How to train your super-net: An analysis of training heuristics in weight-sharing nas*, arXiv preprint arXiv:2003.04276, (2020).

[211] K. YU, C. SCIUTO, M. JAGGI, C. MUSAT, AND M. SALZMANN, *Evaluating the search phase of neural architecture search*, arXiv preprint arXiv:1902.08142, (2019).

[212] R. YU, A. LI, C.-F. CHEN, J.-H. LAI, V. I. MORARIU, X. HAN, M. GAO, C.-Y. LIN, AND L. S. DAVIS, *Nisp: Pruning networks using neuron importance score propagation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9194–9203.

[213] S. ZAGORUYKO AND N. KOMODAKIS, *Wide residual networks*, arXiv preprint arXiv:1605.07146, (2016).

[214] M. D. ZEILER AND R. FERGUS, *Visualizing and understanding convolutional networks*, in European conference on computer vision, Springer, 2014, pp. 818–833.

[215] A. ZELA, A. KLEIN, S. FALKNER, AND F. HUTTER, *Towards automated deep learning: Efficient joint neural architecture and hyperparameter search*, arXiv preprint arXiv:1807.06906, (2018).

[216] T. ZHANG, G.-J. QI, B. XIAO, AND J. WANG, *Interleaved group convolutions*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 4373–4382.

[217] X. ZHANG, X. ZHOU, M. LIN, AND J. SUN, *Shufflenet: An extremely efficient convolutional neural network for mobile devices*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 6848–6856.

[218] F. ZHENG, G. ZHANG, AND Z. SONG, *Comparison of different implementations of mfcc*, Journal of Computer science and Technology, 16 (2001), pp. 582–589.

[219] B. ZHOU, A. LAPEDRIZA, A. KHOSLA, A. OLIVA, AND A. TORRALBA, *Places: A 10 million image database for scene recognition*, IEEE transactions on pattern analysis and machine intelligence, 40 (2017), pp. 1452–1464.

[220] H. ZHOU, J. LAN, R. LIU, AND J. YOSINSKI, *Deconstructing lottery tickets: Zeros, signs, and the supermask*, Advances in neural information processing systems, 32 (2019).

[221] B. ZOPH AND Q. V. LE, *Neural architecture search with reinforcement learning*, arXiv preprint arXiv:1611.01578, (2017).

[222] B. ZOPH, V. VASUDEVAN, J. SHLENS, AND Q. V. LE, *Learning transferable architectures for scalable image recognition*, arXiv preprint arXiv:1707.07012, (2017).

[223] ——, *Learning transferable architectures for scalable image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.