PARTIAL LEAST SQUARES:

A DEEP SPACE ODYSSEY

ARTUR JORDÃO LIMA CORREIA

PARTIAL LEAST SQUARES: A DEEP SPACE ODYSSEY

Tese apresentada ao Programa de Pós--Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais - Departamento de Ciência da Computação. como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: William Robson Schwartz

Belo Horizonte

Novembro de 2020

ARTUR JORDÃO LIMA CORREIA

PARTIAL LEAST SQUARES: A DEEP SPACE ODYSSEY

Thesis presented to the Graduate Program in Ciência da Computação of the Universidade Federal de Minas Gerais - Departamento de Ciência da Computação. in partial fulfillment of the requirements for the degree of Doctor in Ciência da Computação.

Advisor: William Robson Schwartz

Belo Horizonte

November 2020

Correia, Artur Jordão Lima
Partial Least Squares: A Deep Space Odyssey / Artur Jordão Lima Correia. — Belo Horizonte, 2020
xxi, 134 f. : il. ; 29cm

Tese (doutorado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação.
Orientador: William Robson Schwartz

1. Computação - Teses. 2. Visão por computador -Teses. 3. Teoria da estimativa - Teses. 4. Reconhecimento de Padrões. I. Title.

CDU

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Partial Least Squares: A Deep Space Odyssey

ARTUR JORDAO LIMA CORREIA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

William Robson Schwartz
Prof. William Robson Schwartz - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Moacir Antonelli Ponti
Departamento de Ciência da Computação - USP

Hélio Pedrini
Prof. Hélio Pedrini
Instituto de Computação - UNICAMP

[illegible]
Prof. Luiz Eduardo Soares de Oliveira
Departamento de Informática - UFPR

João Paulo Papa
Prof. João Paulo Papa
Departamento de Computação - Unesp

Belo Horizonte, 20 de Novembro de 2020.

Acknowledgments

I am eternally thankful to my family: Ivanilde, Manuela, Americo, Pedro, José and Maria Aparecida for all support they gave me, allowing me to focus on research and studies. I am very grateful to Mirela Pelizaro Valeri and her family for all support they gave me throughout my research.

I would like to thank deeply professor William Robson Schwartz for the outstanding orientation on my graduate study.

I thank my colleagues in Federal University of Minas Gerais. I am very grateful for my colleagues Ricardo Barbosa Kloss, Maiko Min Ian Lie, Fernando Akio de Araujo Yamada and Victor Hugo Cunha de Melo for valuable contributions to this thesis. I am also very grateful for all members of the Smart Sense Lab.

I would like to thank the Brazilian National Research Council – CNPq (Grants 438629/2018-3, 309953/2019-7 and 140082/2017-4), the Minas Gerais Research Foundation – FAPEMIG (Grants APQ-00567-14 and PPM-00540-17) and the Coordination for the Improvement of Higher Education Personnel – CAPES (DeepEyes Project). This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Resumo

Modelos modernos de reconhecimento de padrões visuais são predominantemente baseados em redes convolucionais uma vez que elas têm levado a uma série de avanços em diferentes tarefas. A razão para estes resultados é o desenvolvimento de arquiteturas maiores e a combinação de informações de diferentes camadas da rede convolucional. Tais modelos, entretanto, são computacionalmente custosos dificultando aplicabilidade em sistemas de baixo custo computacional e recursos limitados. Para lidar com esses problemas, propomos três estratégias. A primeira remove estruturas (neurônios e camadas) das redes convolucionais, reduzindo seu custo computacional. A segunda insere estruturas para desenvolver redes convolucionais automaticamente, permitindo construir arquiteturas de alta performance. A terceira combina múltiplas camadas das redes convolucionais, aprimorando a representação dos dados com custo adicional irrelevante. Estas estratégias são baseadas no *Partial Least Squares*, uma técnica de redução de dimensionalidade. Mostramos que o *Partial Least Squares* é uma ferramenta eficiente e eficaz para remover, inserir e combinar estruturas de redes convolucionais. Apesar dos resultados positivos, o *Partial Least Squares* é inviável a grandes conjuntos de dados como ele requer que todos os dados estejam na memória, o que é frequentemente impraticável devido a limitações de hardware. Para contornar tal limitação, propomos uma quarta abordagem, um método de *Partial Least Squares* incremental discriminativo e de baixa complexidade que aprende uma representação compacta dos dados usando uma única amostra por vez, permitindo aplicabilidade em grandes conjuntos de dados. Avaliamos a efetividade das abordagens em várias arquiteturas convolucionais e tarefas supervisionadas de visão computacional, que incluem classificação de imagens, verificação de faces e reconhecimento de atividades. Nossas abordagens reduzem a sobrecarga de recursos computacionais das redes convolucionais e do *Partial Least Squares*, promovendo modelos eficientes em termos de energia e hardware para cenários acadêmicos e industriais. Em comparação com métodos de última geração para o mesmo propósito, obtemos um dos melhores compromissos entre capacidade preditiva e custo computacional.

Abstract

Modern visual pattern recognition models are predominantly based on convolutional networks since they have led to a series of breakthroughs in different tasks. The reason for these achievements is the development of larger architectures as well as the combination of features from multiple layers of the convolutional network. Such models, however, are computationally expensive, hindering applicability on low-power and resource-constrained systems. To handle these problems, we propose three strategies. The first removes unimportant structures (neurons or layers) of convolutional networks, reducing their computational cost. The second inserts structures to design convolutional networks automatically, enabling us to build high-performance architectures. The third combines multiple layers of convolutional networks, enhancing data representation at negligible additional cost. These strategies are based on Partial Least Squares, a discriminative dimensionality reduction technique. We show that Partial Least Squares is an efficient and effective tool for removing, inserting, and combining structures of convolutional networks. Despite the positive results, Partial Least Squares is infeasible on large datasets since it requires all the data to be in memory in advance, which is often impractical due to hardware limitations. To handle this limitation, we propose a fourth approach, a discriminative and low-complexity incremental Partial Least Squares that learns a compact representation of the data using a single sample at a time, thus enabling applicability on large datasets. We assess the effectiveness of our approaches on several convolutional architectures and supervised computer vision tasks, which include image classification, face verification and activity recognition. Our approaches reduce the resource overhead of both convolutional networks and Partial Least Squares, promoting energy- and hardware-friendly models for the academy and industry scenarios. Compared to state-of-the-art methods for the same purpose, we obtain one of the best trade-offs between predictive ability and computational cost.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Pattern recognition plays an important role in cognitive tasks such as natural language processing and image understanding. Modern pattern recognition methods have led to a series of breakthroughs, often surpassing human performance [Deng et al., 2009; Parkhi et al., 2015; Badia et al., 2020]. The reason for these remarkable achievements is the improvement in data representation (i.e., features), which allows discovering new abstractions and patterns from data.

In the context of visual pattern recognition, deep convolutional networks have been the focus of intense research due to their state-of-the-art effectiveness in learning discriminative representations [Krizhevsky et al., 2012]. In particular, most efforts have been devoted to the development of architectures for convolutional networks, since large architectures are a major determinant factor for improving their predictive ability [He et al., 2016; Zagoruyko and Komodakis, 2016; Huang et al., 2019; Kornblith et al., 2019; Tan and Le, 2019; Sankararaman et al., 2020; Rosenfeld et al., 2020; Han et al., 2020], as shown in Figure 1.1. In terms of performance, on the other hand, excessively large architectures are computationally expensive, hindering applicability on low-power and resource-constrained devices. Moreover, such architectures are *data-hungry*, meaning that large datasets are needed to provide a better generalization performance [Kolesnikov et al., 2020], hence, the encouragement for large datasets has been growing [Sun et al., 2017; Kuznetsova et al., 2020].

A parallel line of research to obtain discriminative representations is to discover low-dimensional features through dimensionality reduction techniques. Such techniques are capable of yielding discriminative and compact representations from the original (high-dimensional) data [Li et al., 2019c]. Recent works use dimensionality reduction collaboratively with convolutional networks, where features from the latter are used to feed dimensionality reduction techniques [Vareto and Schwartz, 2020; Suau et al., 2020;

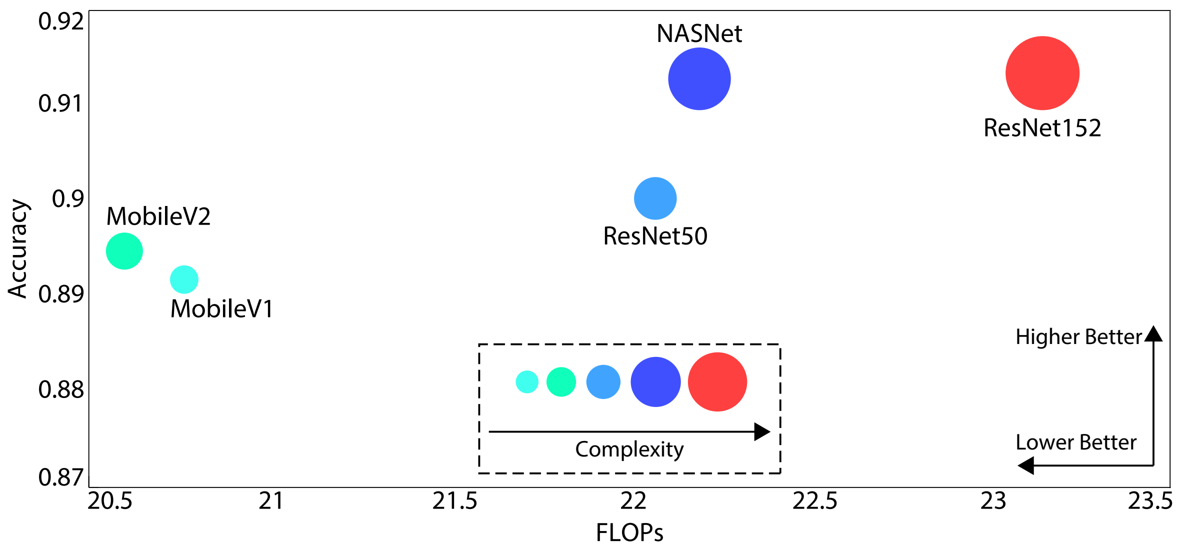


Figure 1.1. Comparison of convolutional networks in terms of predictive ability, computational cost, and complexity. Predictive ability is measured by accuracy. Computational cost is measured by Floating Point Operations (FLOPs). Complexity is measured taking into account the number of neurons (width) and layers (depth), and it is represented by the circle size (larger means more complex). The arrows indicate which direction (in both x and y axes) is better. It is evident that there is a strong relationship between predictive ability and network complexity (circle size), in which more complex networks are more accurate. In turn, network complexity incurs computational cost.

Diniz and Schwartz, 2020]. While these strategies produce encouraging results because the network representation might be enhanced, such a combination is unsuitable for large datasets since traditional dimensionality reduction techniques require all the data to be in memory in advance, which is often impractical due to hardware limitations.

Regardless of the mechanism employed to recognize or improve pattern recognition, there is a trade-off between accuracy and complexity, in which more accurate methods often incur higher complexity and computational cost, as illustrated in Figure 1.1. Thereby, discovering accurate and efficient strategies for pattern recognition, which include enhancing the existing ones, have been the focus of intense research.

1.1 Motivation

Modern visual pattern recognition models are predominantly based on convolutional networks since they are capable of learning effective representations from data [He et al., 2016; Zagoruyko and Komodakis, 2016]. According to previous works [Tan and Le, 2019; Sankararaman et al., 2020; Rosenfeld et al., 2020; Han et al., 2020], large (deeper and wider) convolutional networks lead to better results. Figure 1.1 supports this claim, where larger networks (large circles) have superior predictive ability. In terms of performance, however, such networks suffer from massive computation and memory overhead, incurring slow inference and hindering applicability on low-power and resource-constrained devices. The simplest way to circumvent this dilemma is to evaluate different trade-offs between accuracy and network complexity (i.e., number of neurons and layers), for example, by comparing the performance of ResNet50 (50 layers) with its deeper counterpart ResNet152 (152 layers), see Figure 1.1. This process, however, requires significant human engineering due to its trial-and-error essence. Instead, it is possible to transform or automatically design efficient convolutional networks by employing pruning or neural architecture search (NAS), respectively. The former removes unimportant (or the least important) structures (neurons or layers) from the network, reducing its complexity while preserving as much predictive ability as possible. The latter learns to design accurate and efficient architectures automatically.

Both strategies, however, are not without their limitations. Existing criteria for identifying and removing structures from convolutional networks are ineffective since the accuracy of the original (unpruned) network is often degraded [He et al., 2020; Guo et al., 2020a; Chin et al., 2020; Lin et al., 2020], as shown in Figure 1.2 (left). Besides, many pruning approaches demand a high computational cost, mainly when applied to very deep networks [Huang et al., 2018; Luo et al., 2019; Luo and Wu, 2020]. Regarding the neural architecture search, current strategies analyze a large set of possible candidate architectures and, hence, require vast computational resources and take many days to process even with modern Graphics Processing Units (GPUs) [Baker et al., 2017; Real et al., 2017; Zoph et al., 2018]. Motivated by these issues, we propose simple, effective, and efficient mechanisms for eliminating structures of deep networks as well as discovering high-performance architectures automatically (i.e., without involving human engineering). More precisely, our pruning strategies achieve the best trade-offs between accuracy and computational cost compared to state-of-the-art methods, as illustrated in Figure 1.2 (left). In the context of NAS, our method discovers competitive and low-cost convolutional networks by exploring one order of magnitude fewer models compared to other approaches, thus designing architectures in a few hours on a single GPU, as shown in Figure 1.2 (right).

Besides computational cost concerns, many efforts have been devoted to improve data representation of convolutional networks. In this line of research, previous works have demonstrated encouraging results combining features from different levels (layers) of the network. Such works have followed either multi-scale or HyperNet strategies. While the former redesigns network topology to encode features from shallow and deep

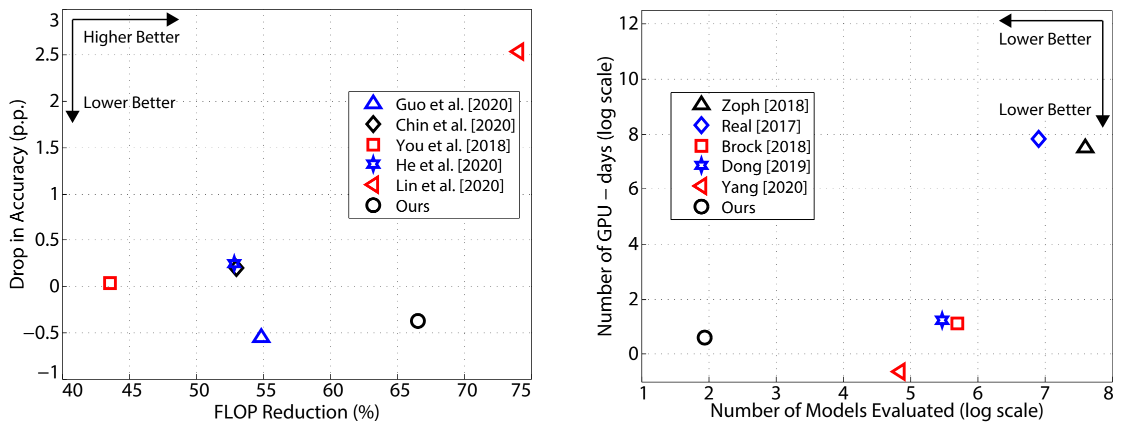


Figure 1.2. Left. Comparison of existing pruning methods on CIFAR-10. Compared to state-of-the-art pruning strategies, our pruning method always provides a better solution (i.e., it is a non-dominated solution) considering one of the performance metrics: accuracy drop (y-axis) or FLOP reduction (x-axis). In this figure, negative values in the y-axis denote improvement regarding the original, unpruned, network. **Right.** Comparison of existing neural architecture search (NAS) methods on CIFAR-10. Our NAS method discovers architectures by exploring one order of magnitude fewer models compared to other approaches. In addition, our method is the most resource-efficient as it designs architectures in a few hours on a single GPU. In both figures, the arrows indicate which direction is better.

layers [Huang et al., 2019; Yang et al., 2020a], the latter preserves network topology, encouraging application on off-the-shelf networks [Hariharan et al., 2015; Kong et al., 2016; Sindagi and Patel, 2019; dos Santos and Ponti, 2019]. Despite improving predictive ability, both multi-scale and HyperNets strategies increase the computational burden significantly since they insert time-consuming operations at multiple levels of the network. To address this problem, we propose an efficient yet accurate approach to extract different levels of representation across multiple layers of deep networks, thus enhancing data representation at negligible additional cost.

A parallel line of research to improve data representation is to learn compact, but discriminative, representations through dimensionality reduction [Li et al., 2019c]. In this context, Partial Least Squares (PLS) has presented remarkable results, mainly when compared to other methods such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) [Schwartz et al., 2009; Sharma and Jacobs, 2011; Hasegawa and Hotta, 2016; Kloss et al., 2017]. The promising results of PLS are associated with its characteristics that include being discriminative and robust to *sample size problem* (when the number of samples is smaller than the number of features). Another attractive aspect of PLS is that it can operate as a feature selection method. However, PLS is unsuitable for large datasets (e.g., ImageNet [Deng et al., 2009]) since all the data need to be available in advance and this could be impractical due to memory constraints. This limitation is not particular to PLS, many dimensionality reduction methods also suffer from this problem [Weng et al., 2003; Zeng and Li, 2014; Alakkari and Dingliana, 2019; Xu and Li, 2019].

To handle the aforementioned problem, many works have proposed incremental versions of traditional dimensionality reduction methods [Arora et al., 2016; Stott et al., 2017; Weng et al., 2003; Zeng and Li, 2014; Alakkari and Dingliana, 2019], where the idea is to learn compact representations using a single sample (or a subset) at a time. Unfortunately, most incremental Partial Least Squares fail to keep all properties of PLS and present a high time complexity. To preserve the fundamental characteristics of PLS, we propose a discriminative and low-complexity incremental Partial Least Squares. Among the advantages of this approach are the preservation of discriminative information, its computational efficiency, and the ability to operate as a feature selection technique.

1.2 Hypotheses

This thesis introduces simple, efficient and effective strategies for improving the trade-off between accuracy, complexity and computational cost in convolutional networks. Specifically, we propose strategies for (i) removing neurons and layers from convolutional networks to decrease the computational cost; (ii) inserting layers to automatically design accurate and low-cost architectures and (iii) combining different levels of representation distributed across the network to improve data representation. These strategies are based on the importance of structures (neurons or layers) that compose the convolutional network. We assign the importance of a specific structure based on the relationship of its output (i.e., feature maps) with the class label on a low-dimensional (compact) space. We find this space by maximizing the covariance between the structure and the class label using Partial Least Squares. Our central hypothesis is that Partial Least Squares learns the importance inherent to predictive ability of the network. Furthermore, we hypothesize that, by using simple algebraic decomposition, it is possible to preserve discriminability on higher-order components of the incremental version of PLS.

Thesis Statement. The statement of this research is as follows:

> *The predictive importance of structures (neurons or layers) composing a convolutional network can be effectively estimated with Partial Least Squares, which in turn can be computed incrementally without degrading its discriminative information. With the estimation of this importance, it is possible to obtain high-performance convolutional networks by removing, inserting or combining structures.*

We demonstrate these claims as well as the effectiveness of our approaches on several convolutional networks and supervised tasks for computer vision. Our results are on par with the state of the art and, in most cases, they achieve the best trade-off between accuracy and computational cost.

1.3 Objectives

From a theoretical perspective, our goal is to demonstrate the potential of Partial Least Squares as a tool for determining the importance of structures composing a convolutional network. Besides, we intend to show that it is possible to preserve underlying properties of Partial Least Squares in its incremental version through simple algebraic decomposition.

From a practical perspective, our goal is to promote mechanisms capable of providing efficient convolutional networks. More specifically, we pretend to provide strategies for (i) accelerating off-the-shelf convolutional networks, (ii) discovering high-performance convolutional architectures automatically and (iii) efficiently improving data representation of convolutional networks. Additionally, we target to provide a memory-friendly version of Partial Least Squares. The main goal behind these strategies is to facilitate the applicability of both convolutional networks and Partial Least Squares on low-power and resource-constrained systems.

1.4 Contributions

The contributions of this dissertation are simple, effective and efficient strategies for improving computational cost and predictive ability of convolutional networks. More precisely, our main contributions are the following. (i) An effective approach to remove (prune) structures (neurons and layers) from convolutional networks. The proposed method identifies potential structures to be removed with minimal or no loss in prediction ability. Compared to existing pruning approaches, our method attains the best trade-off between accuracy and computational cost. (ii) An efficient approach to automatically design high-performance convolutional networks. The proposed method discovers architectures by considering a small search space. In contrast to previous neural architecture search approaches, our method evaluates one order of magnitude fewer models and designs architectures in a few hours on a single GPU. (iii) A low-cost approach to explore multiple levels of representation from convolutional networks. The proposed method captures low-level and refined information distributed over several layers of the network, providing strong and complementary clue that improve the data representation. Different from previous HyperNet strategies, our approach extracts multiple levels of representation at negligible additional cost. (iv) An incremental Partial Least Squares to learn a discriminative and low-dimensional representation of the data using a single sample at a time. The proposed method learns such representation by using a single sample at a time while keeping the properties of traditional Partial Least Squares. Compared to state-of-the-art incremental Partial Least Squares methods, our approach achieves superior performance in both accuracy and time complexity.

The results obtained during our research have been published in important conferences and journals on computer vision and pattern recognition:

Journal Papers

1. Jordao, A., Yamada, F., and Schwartz, W. R. Deep Network Compression based on Partial Least Squares. Neurocomputing, 2020.
2. Jordao, A., Lie, M., and Schwartz, W. R. Discriminative Layer Pruning for Convolutional Neural Networks. Journal of Selected Topics in Signal Processing, 2020.

Conference Papers

1. Jordao, A., Kloss, R. B., and Schwartz, W. R. Latent hypernet: Exploring the layers of Convolutional Neural Networks. International Joint Conference on Neural Networks, 2018.
2. Jordao, A., Kloss, R., Yamada, F., and Schwartz, W. R. Pruning Deep Neural Networks using Partial Least Squares. British Machine Vision Conference Workshops: Embedded AI for Real-Time Machine Vision, 2019.
3. Jordao, A., Yamada, F., Lie, M., and Schwartz, W. R. Stage-Wise Neural Architecture Search. International Conference on Pattern Recognition, 2020.

4. Jordao, A., Lie, M., de Melo, V. H. C., and Schwartz, W. R. Covariance-free partial least squares: An Incremental Dimensionality Reduction Method. Winter Conference on Applications of Computer Vision, 2021.

To promote reproducibility, we release the source code at: https://arturjordao.github.io/PLSDeepSpaceOdyssey/.

1.5 Work Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we define theoretical concepts of deep learning and Partial Least Squares. In Chapter 3, we review the main works related to the contributions of this dissertation. In Chapter 4, we introduce our strategies for removing, inserting and combining structures of convolutional networks and our incremental Partial Least Squares as well. In Chapter 5, we show and discuss the experimental results. In Chapter 6, we present the conclusions of this research and directions for future work. In Appendices A, B and C, we provide the computational time of the convolutional architectures considered in our research, implementation details of our pruning strategies and additional results of our neural architecture search approach, in this order.

Chapter 2

Theoretical Concepts

In this chapter, we introduce the basics of deep learning, which include single and multilayer networks and convolutional networks. Then, we define the concepts of capacity and transfer learning that we use throughout the dissertation. Finally, we describe Partial Least Squares, a dimensionality reduction method that plays an important role in the proposed pruning and HyperNet approaches.

Unless stated otherwise, let $X \subset \mathbb{R}^{n \times m}$ be the matrix of independent variables denoting n training samples in m-dimensional space. Let $Y \subset \mathbb{R}^{n \times k}$ be the matrix of dependent variables representing the class label in a k-dimensional space, where k denotes the number of categories. Finally, let $x_n \subset \mathbb{R}^{1 \times m}$ and $y_n \subset \mathbb{R}^{1 \times k}$ be a single sample of X and Y, respectively. More concretely, y is a one-hot vector with the kth entry equal to 1 and the rest 0.

2.1 Neural Network

In this section, we start by describing single and multilayer networks, which provide useful insights into the properties of deeper and more complex networks. Next, we introduce the convolutional networks, which are designed to recognize patterns in images. Finally, we define the concepts of transfer learning and fine-tuning.

2.1.1 Single-Layer and Multilayer Networks

A neural network is a function $\mathcal{F}$ parametrized by a set of parameters (weights) θ randomly initialized. Given an input x, $\mathcal{F}$ predicts a value $\hat{y}$ based on its parameters θ. Thus, a neural network can be described as $\mathcal{F}(x, \theta) = \hat{y}$. Specifically, $\mathcal{F}$ consists of a series of functions f_i referred to as layers. Such functions are applied sequentially,

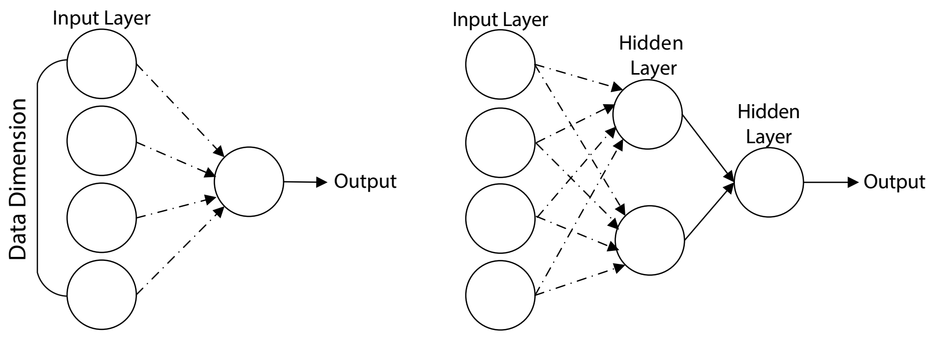


Figure 2.1. Different neural networks architectures. **Left.** Single-layer network. **Right.** Multilayer network.

enabling us to rewrite $\mathcal{F}(x, \theta) = \hat{y}$ as $f_L(f_2(...f_1(x, \theta_1), \theta_2), \theta_L) = \hat{y}$, where L indicates the number of layers composing $\mathcal{F}$, which in turn define the depth of $\mathcal{F}$. Each layer f_i has its own set of parameters, θ_i, and consists of a group of neurons, which are small units that linearly combine an input to generate an output. Intuitively, when $L = 1$, $\mathcal{F}$ is referred to as a single-layer network. On the other hand, when $L > 1$, $\mathcal{F}$ is referred to as a multilayer network. Figure 2.1 illustrates single and multilayer networks.

Training Phase. While the number of layers and neurons define the architecture of $\mathcal{F}$ and are manually predefined, the parameters θ (θ_1, θ_2, θ_L) are randomly initialized. During the learning phase, these parameters are optimized to minimize a cost function (a.k.a loss function). For this purpose, the learning (i.e., training) phase requires the employment of two components: a method for computing multivariable derivatives (gradient) and an optimizer. The first calculates the effective network error based on the expected output (y) and the one predicted ($\hat{y}$) by $\mathcal{F}$ [Rumelhart et al., 1986]. The second walks towards the minimum of the cost function with respect to the gradient [LeCun et al., 1989]. By employing these components, the training stage adjusts θ such that

$$\theta = arg\ min\ \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(x_i, \theta), \tag{2.1}$$

where $\mathcal{L}$ is a loss function, e.g., mean squared error or categorical cross-entropy. We can optimize Equation 2.1 iteratively using Stochastic Gradient Descent (SGD) as follows

$$\theta_{t+1} = \theta_t - \eta\frac{1}{n}\sum_{i=1}^{n} \nabla\mathcal{L}(x_i, \theta_t), \tag{2.2}$$

where θ_t and θ_{t+1} are the current and the updated parameters, respectively, ∇ is the gradient of the loss function $\mathcal{L}$ and η is the learning rate. The latter indicates the intensity of the adjustment in θ, determining the speed at which the network walks towards the minimum of the loss function. A typical procedure is to use different learning rates according to the training epoch, where η is large at the beginning of training, and it is gradually decreased at the final epochs [Loshchilov and Hutter, 2017; He et al., 2019a; Li et al., 2020a]. These steps are repeated k times (epochs) or until the network attains a stopping criterion.

Note that Equation 2.2 optimizes θ with respect to all training samples $x_1, x_2, ...x_n$. Instead, a common practice in current deep models is to optimize θ by considering a subset (batch, B) of samples as follows:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla \mathcal{L}(x_i, \theta_t). \tag{2.3}$$

The size of the batch affects the training dynamic in different ways [Smith et al., 2018; Wu and He, 2018; Yan et al., 2020; Singh and Shrivastava, 2019]. For example, large batch sizes require less updates in θ but demand more memory, while small batch sizes are resource-efficient but might lead to inaccurate batch statistics.

Loss Landscape. After the training phase, the final parameters θ_t (θ for short) are used to measure the predictive ability of $\mathcal{F}$. Besides predictive ability, we can use θ to visualize the loss function curvature (i.e., loss landscape) [Li et al., 2018]. Throughout this chapter, we use the loss landscape to demonstrate, qualitatively, the dynamics of θ when different components are used on $\mathcal{F}$. A brief description of this technique is as follows. Define α and β scale factors generated from a normal distribution. The loss landscape can be visualized by computing the loss for each point of a 3D-grid in terms of $Z_{i,j} = \mathcal{L}(\theta + i \cdot \alpha + j \cdot \beta)$, where $Z_{i,j}$ indicates the loss value to the point i, j of the grid. In summary, the loss landscape is yielded by perturbing $(i \cdot \alpha + j \cdot \beta)$ the final parameters (θ) of the network.

According to Li et al. [2018], the sharper the loss landscape the more sensitive the network is to perturbations in its parameters, hence, it is harder to train and might exhibit poor generalization.

Deeper and Wider Architectures. According to the universal approximation theorem, with enough number of neurons and $L > 1$, a neural network is able to approximate any function [Hornik et al., 1989; Cybenko, 1992; Barron, 1993]. It has been confirmed that deep (many layers) and wide (many neurons) architectures lead to better predictive ability [He et al., 2016; Zagoruyko and Komodakis, 2016; Tan and Le, 2019; Han et al., 2020]. It turns out that larger architectures are able to learn more discriminative representations, thus improving predictive ability. For instance,

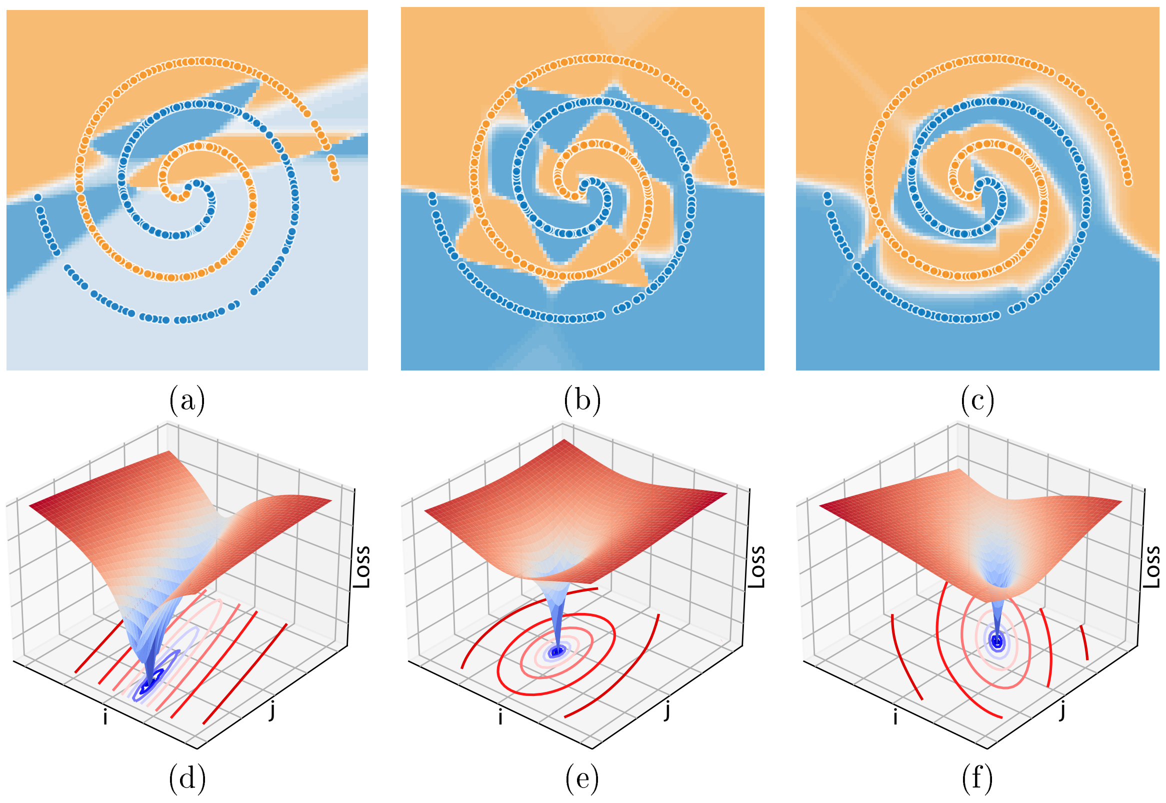


Figure 2.2. Top. Decision boundary of different architectures. (a) Single-layer architecture with two neurons. (b) Single-layer architecture with eight neurons. (c) Multilayer architecture with two layers with four neurons in each layer. **Bottom.** Loss landscape of the architectures (a), (b) and (c). It is possible to observe that large architectures lead to a softer decision surface and flatter loss landscape.

Figures 2.2 (a)-(c) illustrate the decision surface yielded by three architectures with one, two and three layers, respectively. From these figures, it is possible to observe that the decision surface becomes softer as we increase the number of layers. This occurs due to higher discriminability achieved by the deeper networks. Similar trends also occur on the loss landscape, where large networks (Figures 2.2 (e) and (f)) present softer landscapes.

2.1.2 Convolutional Network

As we explain in Section 2.1.1, multilayer architectures are able to approximate any function, thus they could be applied to any task. However, in the context of visual pattern recognition, they are inadequate since they do not consider the spatial structure of the image. This is a consequence of its architecture that associates each input data dimension (in image context one pixel) to one neuron, as shown in Figure 2.3. Such

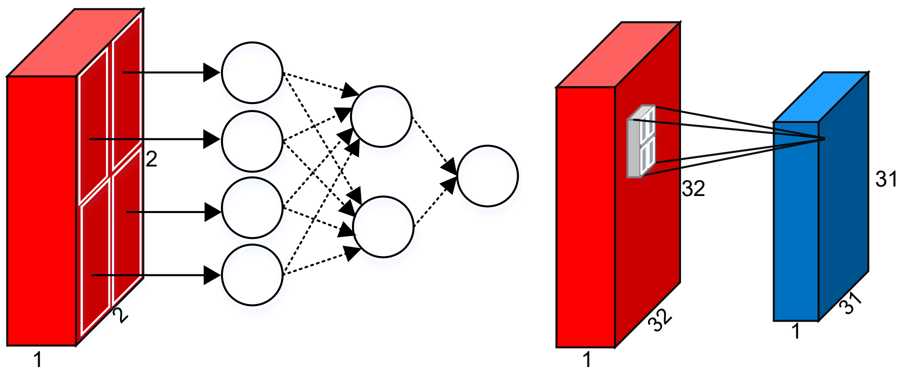


Figure 2.3. Example of a one-channel image (red cube) as input to different network architectures. For simplicity, we remove the bias term. **Left.** Image used directly as input to MLP. In this scheme, each pixel is associated with one neuron. **Right.** Neurons organized as elements of a 2×2 convolutional filter (gray cube). In this modeling, the neurons slide over the image, following the standard convolution process, and yield the feature map (blue cube).

association hinders multilayer networks incapable of learning local dependencies, i.e., patterns in different patches of the image. Another deficiency in using images directly as input to multilayer networks is the large number of parameters to be learned. For example, taking as input an RGB image of 32×32 pixels and an architecture with one hidden layer of 100 neurons, we would have $307,300$ $(32 \times 32 \times 3 \times 100)$ parameters to be estimated.

To handle the problems above, one approach is to compute handcrafted features from the image (e.g., Histogram of Oriented Gradients [Dalal and Triggs, 2005]) and use them as input to the network. Thus, the local structures of the image are encoded in the features. Another strategy is to employ convolutional networks, which interprets neurons as elements of convolutional filters. The intuition behind such models is that since the convolution operation consists of sliding the convolutional filter over the image, the neurons that compose it will be able to exploit local structures of the image, see Figure 2.3 (right). In other words, convolutional networks employ multiple copies of the same neuron in different places through the convolution operation, thus enabling it to learn patterns once and use them in multiple locations. In contrast to approaches based on handcrafted features, which require expert knowledge and expensive human engineering, convolutional networks learn the best representation for the data and task at hand.

Convolutional networks consist of stacks of convolutional and downsampling layers, batch normalization, activation and classification layers, as illustrated in Figure 2.4. Below, we describe these components.

Convolutional Layers. A convolutional layer is a set of k filters that receives an

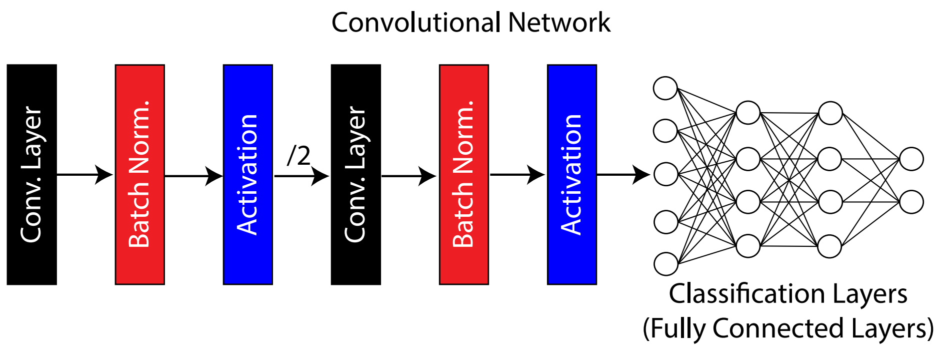


Figure 2.4. Structure of a standard convolutional network. In practice, convolutional architectures consist of stacks of convolutional and downsampling (represented by /2) layers, batch normalization, activations and classification layers.

input and outputs a k-channel convoluted image (feature maps), which can be used as input to the successive layers or presented to a classifier. Due to the nature of the convolution operation, this layer yields feature maps with spatial dimensions smaller than the input provided, see Figure 2.3 (right). Formally, the convolution operation reduces the input in terms of

$$\frac{W-w}{s_x}+1, \frac{H-h}{s_y}+1, \tag{2.4}$$

where W and H are the spatial dimensions of the input, w and h are the filter dimensions and, s_x and s_y are the strides applied during the convolution. In practice, most works employ zero-padding and stride of one, which ensures that the input and output have the same spatial dimension [He et al., 2016; Zagoruyko and Komodakis, 2016; Huang et al., 2017; Howard et al., 2017; Sandler et al., 2018]. The zero-padding process consists of adding zero values on the input's edges, which means increasing the dimensions W and H.

It is worth mentioning that spatial reduction in feature maps plays an important role in learning new representations [Greff et al., 2017b], but modern architectures leave this reduction to downsampling layers only.

Regarding the dimensions of the filter, most human-designed architectures employ filters 3×3 [Simonyan and Zisserman, 2015; He et al., 2016; Zagoruyko and Komodakis, 2016; Huang et al., 2017; Howard et al., 2017; Sandler et al., 2018], as larger filters incur a higher computational cost [He et al., 2019a].

Downsampling Layers. According to Greff et al. [2017a], an important role in learning new representations is to reduce the spatial dimensions of feature maps. There exists two distinct way of achieving this reduction: pooling operations and convolutional layers with stride 2×2. The former reduces spatial dimensions by applying math operations on small regions (i.e., 2×2) of the feature maps. Such operations reduce these regions into a single value. The size of the region is a parameter and typical operations include the maximum and the average value. Some convolutional architectures employ a special type of pooling [Howard et al., 2017; Sandler et al., 2018; Zoph et al., 2018], the global pooling, which reduces the entire feature map into a single value. The latter reduces spatial dimension by employing a convolutional layer with strides 2×2. In contrast to pooling operations, this downsampling strategy has parameters to be learned and is often adopted in modern convolutional networks.

Batch Normalization Layers. One of the key components to the success of deep networks is Batch Normalization [Ioffe and Szegedy, 2015]. The widely know motivation of the Batch Normalization (BN) technique is to normalize the shifts in input distribution caused by updates to the successive layers, a phenomenon refers to as *internal covariance shift*. Recently, Santurkar et al. [2018] showed that such a phenomenon does not exist and the success of BN can be assigned to the fact that it makes the loss landscape more smooth, which improves and accelerates convergence.

In practice, Batch Normalization is an affine transformation that normalizes feature maps using statistics from a batch of samples. Formally, it works as follows. Let $X \subset \mathbb{R}^{B \times m}$ be the input of a BN layer, where B denotes the batch of samples in the m-dimensional space. A BN layer normalizes X in terms of

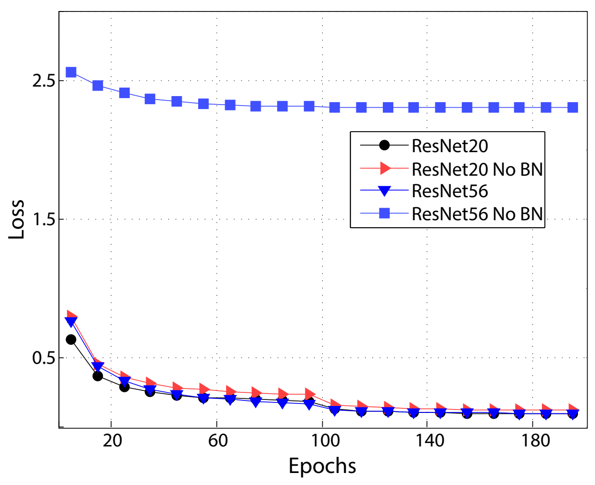
$$\bar{X} = \left(\frac{X - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) \gamma + \beta, \tag{2.5}$$

where μ_B and σ_B are the mean and variance computed from the batch of samples. The parameters γ and β are learnable variables and have their initial values set to one and zero, respectively, by default. These parameters ensure that the non-linearity in X (if it exists) be preserved after BN normalization [Ioffe and Szegedy, 2015].

At the testing stage, to remove the dependency of batch statistics (μ_B and σ_B) and keep the inference deterministic, the mean and variance are replaced by their exponential moving statistics, which in turn are estimated during the training phase.

Despite its simplicity, Batch Normalization plays a key role in training deep convolutional networks. For example, Figure 2.5 (top) shows that networks with BN converge faster and exhibit a flatter loss landscape (bottom). More importantly, as shown in Figure 2.5 (top), by increasing the network depth from 20 to 56, the employment of BN is mandatory since without it the training meet collapse.

Activation Layers. A fundamental component of the deep networks is the activation function. This component, $a(.)$, applies element-wise transformations to a given input



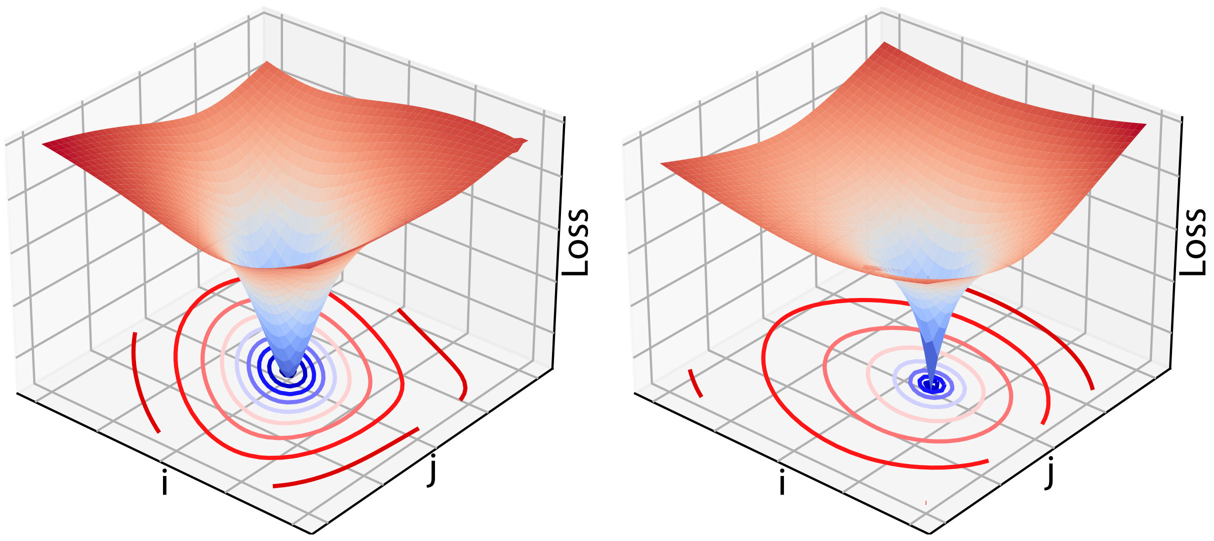


Figure 2.5. Top. Training dynamics of networks with and without Batch Normalization (No BN). Interestingly, a deep network with BN (ResNet56) converged faster than its shallower counterpart without BN (ResNet20 No BN). **Bottom left.** Loss landscape of ResNet20 with Batch Normalization. **Bottom right.** Loss landscape of ResNet20 without Batch Normalization. It is possible to observe that the loss landscape of ResNet with batch normalization is flatter than its No BN version, leading to faster training convergence (as suggested in top).

(often a feature map). Such transformations are targeted to introduce non-linearities to the input, which plays an important role in training dynamics (i.e., convergence) and predictive performance [Xu et al., 2015; Eger et al., 2018; Ramachandran et al., 2018]. Figure 2.6 illustrates the non-linearly introduced by different activation functions.

We can group activation function in two categories: saturated and non-saturated. The former includes activations where the output range lies in a finite interval, e.g., $[-1\ 1]$ (hyperbolic tangent) or $[0\ 1]$ (sigmoid). The latter includes activations where the output range lies in an infinite interval, e.g., $[0\ +\infty)$ (ReLU) or $(-\infty\ +\infty)$ (Leaky

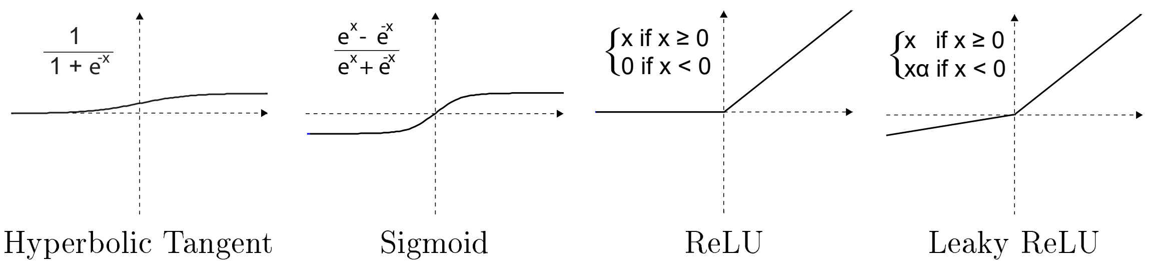

$\frac{1}{1+e^{-x}}$
$\frac{e^x - e^{-x}}{e^x + e^{-x}}$
$\begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$
$\begin{cases} x \text{ if } x \geq 0 \\ x\alpha \text{ if } x < 0 \end{cases}$
Hyperbolic Tangent
Sigmoid
ReLU
Leaky ReLU

Figure 2.6. Activation functions. Degrees of non-linearity (y-axis) when applying different activation functions on an input (x-axis). Different activation functions introduce different degrees of non-linearity, which influence the convergence rate and predictive performance.

ReLU).

From a theoretical perspective, activation functions are equivalent since they enable approximate any function [Hornik, 1991]. In practice, on the other hand, it has been argued that the rectified family of activations (e.g., ReLU and Leaky ReLU) is more suitable for training deep networks [Glorot et al., 2011; Xu et al., 2015; Ramachandran et al., 2018; Eger et al., 2018], leading to better and faster optimization as illustrated in Figure 2.7.

Classification Layers. The learning phase in convolutional networks consists of updating all the parameters composing the architecture. For this purpose, at the end of the architecture, an MLP (hereafter referred to as fully connected layers) is introduced so that we can calculate a loss function and update all the parameters. A well-defined relation between convolutional and fully-connected layers is that the first extracts the features while the second performs the classification [Ke et al., 2017; Caron et al., 2018].

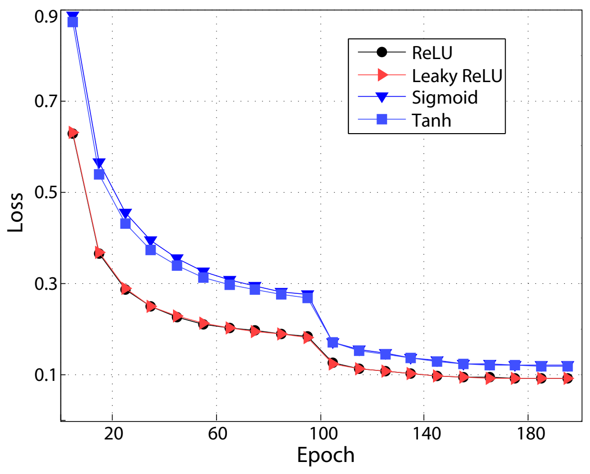


Figure 2.7. Training dynamics of ResNet20 using different activation functions.

2.1.3 Capacity

Prior works have revealed that large (deep and wide) convolutional networks lead to better predictive performance and generalization [He et al., 2016; Zagoruyko and Komodakis, 2016; Huang et al., 2017; Zagoruyko and Komodakis, 2016; Tan and Le, 2019; Han et al., 2020]. The reason for these results is that large architectures have a higher capacity.

The term capacity refers to the ability of the network to learn different data representations (i.e., its expressive power) and exhibits a strong relationship with network size [Tan and Le, 2019; Han et al., 2020]. The network capacity increases by adding more filters (network width) [Zagoruyko and Komodakis, 2016], layers (network depth) [He et al., 2016], or aggregating a set of transformations (cardinality) [Xie et al., 2017]. For example, in Section 2.1.1 (Figure 2.2), we increase the network capacity by adding more hidden layers, hence, it provided a better decision boundary compared to its shallow counterpart.

Although providing better data representation, high-capacity networks introduce some concerns, e.g., its low computational performance. Additionally, high-capacity networks are data-hungry, which means that a larger number of training samples are required to avoid underfitting (i.e., no zero loss in training) and a bad generalization. For example, on CIFAR-10, the VGG architecture (16 layers deep) attains superior accuracy than ResNet (56 layers deep) when no data-augmentation is used, as VGG has less capacity than ResNet. Interestingly, this suggests that there is no relation between the number of parameters and capacity since VGG has 17$\times$ more parameters than ResNet.

Deep-Double Descent. It is well-known from bias and variance regime that above a particular complexity (i.e., capacity), models tend to decrease their predictive performance [Bishop, 2007]. Recently, Nakkiran et al. [2020] observed that deep models do not follow this conventional regime. Their work reveals a phenomenon, called *deep double descent*, in which increasing network capacity above a threshold (depth or width), the network predictive ability first decreases (as expected by the bias-variance regime) and then increases again. Surprisingly, deep double descent is not only restricted to capacity, but also to the number of training epochs and samples.

2.1.4 Transfer Learning and Fine-tuning

Despite state-of-the-art effectiveness in learning discriminative representations, deep convolutional networks require vast computational resources for training from scratch (when the parameters are randomly initialized) and take many days to process even with modern GPUs. For example, on ImageNet, VGG and ResNet take 13 and 20 days, respectively, to train for 90 epochs. Unfortunately, when applied to other datasets (domains), the parameters of the network need to be trained again due to unseen patterns of the new domain. An alternative to training from scratch is to employ a technique referred to as *transfer learning*.

Transfer learning consists of adapting a network trained on one domain (source)[1] to another one (target) by the means of reconditioning the network parameters using the target domain, a process named *fine-tuning*. By using transfer learning, the training stage is faster since the network has already learned basic (e.g., edges and texture) and complex patterns (e.g., parts and shapes of objects) and it only needs to be adjusted to the target domain. According to previous works [Hendrycks et al., 2019; Shafahi et al., 2020; Kolesnikov et al., 2020], for some scenarios, transfer learning is able to achieve better results than training from scratch.

While transfer learning and fine-tuning play an important role in deep learning, there exist some scenarios that influence the effectiveness of these techniques. Based on previous works [Razavian et al., 2014; Yosinski et al., 2014], we can highlight four major scenarios:

1. *Target domain is small and similar to the source domain.* In this case, fine-tuning might lead to overfitting or poor generalization because the target domain has few samples to satisfy the network capacity [Kornblith et al., 2019]. Instead, as both domains are similar (i.e., some categories are visually akin), it is more adequate to use feature maps from the network to train a linear classifier [Razavian et al., 2014].

2. *Target domain is large and similar to the source domain.* In this setting, due to large amount of data in the target domain, fine-tuning will be robust to overfitting and provide good generalization. In addition, since the domains are similar, it is possible to frozen (do not change weights) some early layers since the patterns learned by them, likely, will be the same [Yosinski et al., 2014].

3. *Target domain is small and different from the source domain.* In this case, fine-tuning might fail because the target domain lies on a low data regime. An alternative is to learn a classifier using feature maps from the network, but, since the domains are different, it is more suitable to use feature maps from early layers. This is because deep layers contain more dataset-specific features, hence, early layers might work better [Kloss et al., 2018].

[1]In this dissertation, the term domain indicates a dataset or task.

4. *Target domain is large and different from the source domain.* In scenario, because of the large number of samples in the target domain, fine-tuning process will work well. Even though the domains are different, the fine-tuning can improve the results [Ke et al., 2017]. However, whether the domains are very different, for example, the origin domain is image classification and target domain is activity recognition from wearable sensors data, it is more recommended to learn the network from scratch to avoid divergence problems [Azizpour et al., 2016; Kornblith et al., 2019].

Based on these cases, it is important to know the characteristics of both source and target domains before performing transfer learning and fine-tuning. This way, it is possible to avoid unexpected results.

2.2 Partial Least Squares

Partial Least Squares (a.k.a Projection on Latent Structures) is a dimensionality reduction method that yields a set of discriminative latent variables taking into account the relationship between independent (X) and dependent (Y) variables [Geladi and Kowalski, 1986; Abdi, 2010].

The idea behind Partial Least Squares (PLS) is to find a projection matrix $W(w_1, w_2, ..., w_c)$ that projects the high dimensional space onto a low c-dimensional space (latent space), where $c \ll m$. In essence, W can be interpreted as a weight matrix that assigns importance for each feature of X. To find W, PLS aims at maximizing the covariance (Cov for short) between the independent and dependent variables. Formally, PLS constructs W such that

$$w_i = maximize(\text{Cov}(Xw, Y)), \text{ s.t}\|w\| = 1, \tag{2.6}$$

where w_i denotes the ith component of the c-dimensional space. The exact solution to Equation 2.6 is given by

$$w_i = \frac{X^TY}{\|X^TY\|}, \tag{2.7}$$

with X and Y normalized (transformed into Z-scores).

From Equation 2.7, it is possible to compute all c components (c is a parameter) using either Nonlinear Iterative Partial Least Squares (NIPALS) [Abdi, 2010] or Singular Value Decomposition (SVD). Most works employ NIPALS since it is capable of finding only the c first components, while SVD always finds all the m components,

Algorithm 1: NIPALS for binary PLS

```
1 for  $i = 1$ to $c$ do
2     $w_i = \frac{X^T Y}{\|X^T Y\|}$
3     $t_i = X w_i$
4     $q_i = \frac{Y^T t_i}{\|Y^T t_i\|}$
5     $p_i = \frac{X^T t_i}{\|X^T t_i\|}$
6     $X = X - t_i p_i^T$
7     $Y = Y - t_i q_i^T$
8 end
```

Algorithm 2: NIPALS for multi-class PLS

```
1 for  $i = 1$ to $c$ do
2     randomly initialize $u \in \mathbb{R}^{m \times 1}$
3     $w_i = \frac{X^T u}{\|X^T u\|}$
4     $t_i = X w_i$
5     $q_i = \frac{Y^T t_i}{\|Y^T t_i\|}$
6     $u = Y q_i$
7     Repeat $3 - 6$ until convergence
8     $p_i = X^T t_i$
9     $X = X - t_i p_i^T$
10    $Y = Y - t_i q_i^T$
11 end
```

being computationally prohibitive for large datasets [Xu and Li, 2019; Maalouf et al., 2019].

Algorithms 1 and 2 introduce the steps of NIPALS to binary (when Y is single-column) and multi-class (when Y is multiple-columns) problems, respectively. In Algorithms 1 and 2, t_i denotes the projected samples (a.k.a factor scores) on the current component w_i, p_i and q_i represent the loadings with respect to X and Y, in this order. Figure 2.8 gives a graphical representation of the matrices found by PLS (binary) in one iteration of NIPALS.

It is worth mentioning that the single difference from Algorithm 2 to 1 is the convergence step, which compresses the multiple-columns of Y to a single value. This convergence step (step 7 in Algorithm 2) is achieved when no changes occur in w_i. In addition, we might define a finite number of steps as a convergence criterion for ensuring that the method stops.

In contrast to common dimensionality reduction techniques such as Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA), PLS presents many advantages. For example, compared to PCA, PLS requires substantially fewer components to achieve its optimal accuracy [Schwartz et al., 2009]. Compared to LDA, PLS is robust to sample size problem[2] since the computation of its components, given by Equation 2.7, does not involve inversion of matrices. The sample size problem takes place when the data matrix X consists of more features than samples. Unfortunately,

[2]Sample size problem, zero determinant and singularity are names for the same problem.

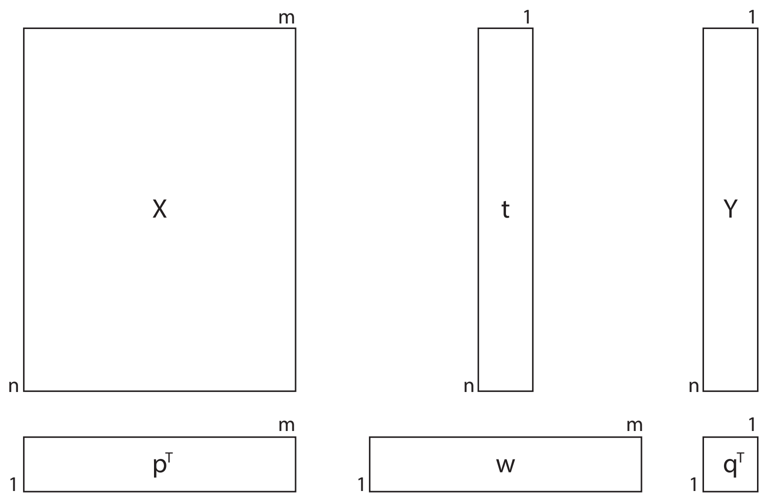


Figure 2.8. Graphical representation of the matrices estimated by PLS in one iteration of the NIPALS algorithm.

this is a common scenario in approaches associated with deep learning, where the features present high dimensionality. In these cases, dimensionality reduction techniques that involve inversion of matrices, e.g., LDA, are highly limited.

2.2.1 Variable Importance in Projection

Besides being more flexible and, often, attaining superior performance than traditional dimensionality reduction techniques [Schwartz et al., 2009; Hasegawa and Hotta, 2016; Kloss et al., 2017], another interesting aspect of PLS is that it can operate as a feature selection method. For this purpose, after computing the projection matrix W, we need to employ Variable Importance in Projection (VIP) that estimates the importance of each feature f_j w.r.t its contribution to yield the low dimensional space. According to Mehmood et al. [2012], VIP is defined as

$$f_j = \sqrt{m \sum_{i=1}^{c} SS_i(w_{ij}/\|w_i\|^2)/\sum_{i=1}^{c} SS_i}, \tag{2.8}$$

where SS_i is the sum of squares explained by the ith component, which can be expressed as $q_i^2 t_i' t_i$ (defined in Algorithms 1 and 2) [Mehmood et al., 2012]. Importantly, the feature importance is given by its linear relationship with the class label. In the context of deep learning, previous works argued that the linear relationship between deep features and their labels provides surprising results [Donahue et al., 2014; Razavian et al., 2014; Azizpour et al., 2016; Brendel and Bethge, 2019; Kornblith et al., 2019]. This suggests that linear models (i.e., PLS) are good candidates to be employed in this context.

While there exist many other techniques for feature selection with PLS such as genetic algorithm PLS [Hasegawa et al., 1997] and Monte-Carlo based PLS [Cai et al., 2008], VIP is simpler, require fewer computations and has no parameters to be set [Mehmood et al., 2012]. Hence, VIP is the most used technique for feature selection with PLS [Schwartz et al., 2009; dos Santos Junior et al., 2016; Diniz and Schwartz, 2020].

Chapter 3

Related Work

In this chapter, we review the main works related to the contributions of this dissertation. In Section 3.1, we describe the convolutional networks considered throughout our research. In Section 3.2, we introduce pruning approaches that remove different structures from convolutional networks. In Section 3.3, we review state-of-the-art neural architecture search approaches. In Section 3.4, we present representative works that focus on exploring multiple layers in convolutional networks. Finally, in Sections 3.5 and 3.6, we describe incremental dimensionality reduction methods and modern feature selection techniques, respectively.

3.1 Convolutional Networks

While there exist many convolutional networks, our review considers the ones that are often employed in the topics of our work, which include plain, residual and lightweight networks. It is important to mention that due to implementation details, throughout our research, we consider only the architectures described in this section.

Plain Networks. The simplest convolutional architecture is the one that connects a layer i to its subsequent layer $i + 1$. Due to this structure, these networks are named plain networks [He et al., 2016]. One of the most popular plain networks is the *Visual Geometry Group* (VGG) architecture [Simonyan and Zisserman, 2015]. The VGG architecture consists of 3×3 convolutional layers, where each one is followed by 2×2 max-pooling. An interesting characteristic of VGG is that when the feature maps of a layer are halved (due to pooling-operations), the number of filters of the subsequent layer is doubled.

While VGG-like settings such as 3×3 filters are often adopted in architectures for image classification [He et al., 2016; Huang et al., 2017; He et al., 2019a], in the context of activity recognition based on wearable sensors (hereafter referred to as activity recognition), many other configurations have been explored. Different from images, the input samples in activity recognition are temporal windows generated from raw signals (a detailed description of this procedure will be given in Section 5.1) and networks such as VGG are not designed to explore this data structure. Motivated by this, many works have proposed plain architectures to classify activities from wearable data [Chen and Xue, 2015; Ha et al., 2015; Rueda et al., 2018; Xu et al., 2018]. For example, Chen and Xue [2015] proposed an architecture with three convolutional layers, where each layer is followed by 2×1 max-pooling operations. Besides designing architectures, other works have suggested learning filters separately for each modality (e.g., accelerometer and gyroscope) [Ha et al., 2015; Ha and Choi, 2016] as well as improving the input sample representation before forwarding them to the network [Lu and Tong, 2019].

Residual Networks. It is well-known that deeper networks incur high representation capacities and, hence, lead to state-of-the-art effectiveness in learning discriminative representations [He et al., 2016; Tan and Le, 2019]. Unfortunately, deep networks are harder to optimize [Ioffe and Szegedy, 2015; He et al., 2016; Ghorbani et al., 2019]. To address this problem, He et al. [2016] proposed to design a residual learning architecture, named *Residual Network* (ResNet). Their residual architecture consists of connecting a layer i with a subsequent layer $i + j$, $j > 1$. This connection (a.k.a skip-connection) among layers is done by adding (element-wise) their feature maps. Even though simple, He et al. [2016] demonstrated that these connections enable opti-

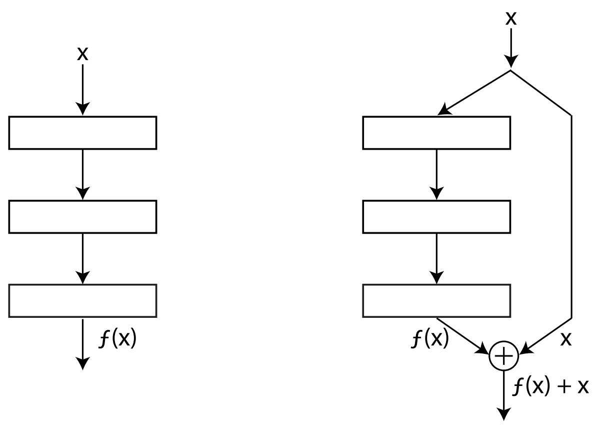


Figure 3.1. Left. Plain network. The output of a layer i is directly connected to its subsequent layer $i+1$. **Right.** Residual network. The output of the layers i and $i+j$, $j > 1$, are added to compose the final output. The symbol $\oplus$ indicates element-wise adding operation. Formally, given an input x, plain networks output $f(x)$ while their residual counterpart output $f(x) + x$.

mize ultra-deep networks (e.g., $100 - 1000$ layers) and achieve notable improvements in accuracy. Figure 3.1 (left) illustrates the core idea behind residual architectures.

Due to the success and simplicity of residual networks, most architectures are predominantly based on residual learning [Zagoruyko and Komodakis, 2016; Xie et al., 2017; Sandler et al., 2018; Zoph et al., 2018; Tan and Le, 2019; Vahdat et al., 2020].

Lightweight Networks. To reduce the number of parameters and floating point operations of convolutional networks, Howard et al. [2017] proposed a lightweight version of convolutional layers. This lightweight version, referred to as depthwise separable convolutions, replaces a (dense) convolutional operation into two lightweight operations: depthwise and 1×1 convolution. The former applies a filter for each input channel separately, which is different from standard convolutions — a filter slides over all input channels. The latter combines the output of the first operation through 1×1 convolution.

Modern architectures, including the ones yielded by neural architecture search, employ depth-wise separable convolutions [Zoph et al., 2018; Sandler et al., 2018; Howard et al., 2019; Vahdat et al., 2020]. Recent works, however, have argued that lightweight networks lack efficient implementation in current deep learning frameworks [Wang et al., 2018a; Vahdat et al., 2020; Gupta and Tan, 2020]. We believe this phenomenon is because depth-wise separable convolutions might increase latency, as a single layer (standard convolution) is replaced into two layers. Thereby, it is interesting to investigate alternatives to reduce both parameters and floating-point operations.

3.2 Pruning Structures in Convolutional Networks

Due to the over-parameterized regime of convolutional networks, many of its structures (neurons and layers) become redundant or unimportant [Han et al., 2015; Zhang et al., 2019; Chatterji et al., 2020]. Thus, it is possible to remove such structures with minimal or no loss in the prediction ability (i.e., accuracy).

Pruning approaches are leveraged by an analogy to human brain plasticity, which can recover from damages after appropriate treatment. In this sense, the idea behind pruning is to identify structures of the network that provide the lowest *damage* (i.e., drop in accuracy), thus enabling a simple and effective *recovery*. For this purpose, existing pruning approaches apply different criteria for determining the importance of neurons and layers and focus on eliminating specific structures. Figure 3.2 summarizes the main types of existing pruning approaches.

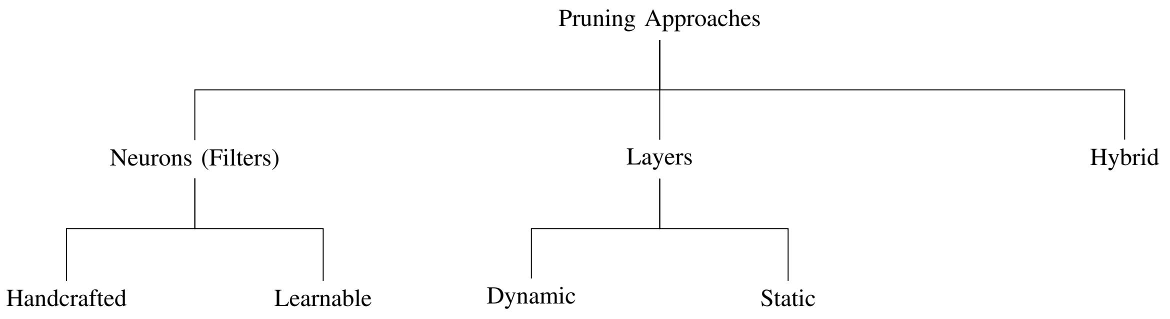


Figure 3.2. Existing pruning strategies grouped by the type of structure removed: neurons (filters), layers or both (hybrid). Strategies that remove neurons are divided according to the essence of the criteria for assigning neuron importance: learnable or handcrafted. Strategies that remove layers are divided according to the way of eliminating layers: dynamic or static.

3.2.1 Pruning Neurons

Pruning neurons (filters) from convolutional networks consist of locating the ones that could be removed with small or no loss in network accuracy. In particular, pruning approaches focus on eliminating filters in convolutional layers since they dominate most computation [Han et al., 2015; Li et al., 2017]. Towards this end, Han et al. [2015] proposed a three-step iterative pipeline: locate and remove potential filters based on their importance, and adjust the weights of the resulting (pruned) network. Despite simple, modern pruning approaches employ slight variations of this pipeline, often modifying only the criteria for assigning filter importance.

Handcrafted Criteria. Previous works suggested that simple statistics computed on the filters such as ℓ_1-norm [Li et al., 2017; Liu et al., 2019b; Frankle and Carbin, 2019; Renda et al., 2020], ℓ_2-norm [He et al., 2018a], and geometric mean [He et al., 2019b] are capable of identifying unimportant filters. Specifically, these statistics are estimated considering the weights of the filters, which means that filters are represented by their weights. Instead, other works have observed that estimating filter importance based on its output (feature maps) is more appropriate since it takes into account the influence of data [Luo et al., 2019; Yu et al., 2018; Lin et al., 2020; Tan and Motani, 2020]. For example, Lin et al. [2020] and, Tan and Motani [2020] demonstrated that low-rank and average absolute value of the feature maps, respectively, indicate low-importance filters; thus, such filters can be removed without degrading network accuracy.

Importantly, some strategies operate in a layer-by-layer fashion [Li et al., 2017; Huang et al., 2018; Luo et al., 2019]. In this scheme, the network is pruned considering one layer at a time and, after pruning a layer, some epochs of fine-tuning are performed. Thereby, the total number of fine-tuning stages grows linearly to the number of layers. For example, the approach by Luo et al. [2019] performs 16 stages of fine-tuning for each layer pruned; therefore, to prune a 56-layer network is necessary (at least[1]) 56×16 stages of fine-tuning, which is computationally expensive.

Different from the aforementioned works, Luo and Wu [2020] proposed to estimate filter importance based on the distribution divergence of the network after its removal. More concretely, the importance of a filter is assigned by Kullback-leibler divergence (KL) [Kullback and Leibler, 1951] between the softmax of the original (unpruned) network and the network without this filter. Despite the positive results, this approach is computationally expensive since after removing a filter is necessary to forward samples through the network. In summary, to prune a network of 400 filters this approach requires 400 forward predictions.

Compared to existing handcrafted criteria, our criterion (PLS) for assigning filter importance is more suitable to indicate unimportant filters, as it achieves the lowest drop in accuracy. Compared to the layer-by-layer strategies, we show that our approach is more efficient since it achieves superior performance with only 10 (or less) stages of fine-tuning.

Learnable Criteria. Instead of designing handcrafted criteria, in this category, filter importance is imposed as an optimization criterion [Liu et al., 2017; Huang and Wang, 2018; Li et al., 2019c; Chin et al., 2020; Guo et al., 2020a]. Such approaches typically associate each filter with a learnable variable (scale factor), which induce unimportant filters to have small scaling factors during the optimization stage (i.e., training phase). Overall, scaling factors are interpreted as the importance of a filter. Thus, filters associated with small scaling factors are the least important ones and can be removed. Due to the optimization phase, most works in this category require that the network be trained from scratch.

Instead of learning scale factors, other works in this category propose to learn agents. These agents take filters as input and output binary decisions indicating whether a filter will be kept or removed [Huang et al., 2018; He et al., 2018b]. Through reinforcement learning, these agents are encouraged to remove filters while satisfying some policy, e.g., computational-budget (the pruned network has the best accuracy given an amount of hardware resources) or quality (the pruned network has the smallest loss in accuracy). In contrast to scaling factors approaches, agent-based strategies can be applied to off-the-shelf networks since the agents are not learned jointly with the network.

Compared to learnable criteria strategies, we show that our pruning method obtains one of the best trade-offs between accuracy and computational cost. In addition,

[1]In practice, layer-by-layer approaches perform additional fine-tuning epochs after pruning all layers.

we remove more floating point operations than cost-aware approaches [Huang et al., 2018; He et al., 2018b], even without considering the computational cost in the pruning process.

3.2.2 Pruning Layers

A recent trend in compression and acceleration of deep networks by pruning is to remove entire layers instead of small components such as filters. In this family of pruning, most strategies are predominantly grounded on the unraveled view of residual networks [Veit et al., 2016]. The unraveled view states that each stage in the network learns a single level of representation, and modules (set of layers) within a stage only refine representations on the same level [Greff et al., 2017b]. This view ensures that the removal of a single module does not degrade predictive ability. Most importantly, there is only evidence for unraveled view in residual-based networks — the output of preceding layers is propagated to successive layers (see Figure 3.1 left) [He et al., 2016]. Thereby, all strategies focusing on removing layer/modules are limited to residual networks and their variations.

It is worth mentioning that before the work by Veit et al. [2016], Huang et al. [2016] had proposed to eliminate entire modules, but, from a regularization perspective to training very deep networks. Roughly speaking, Huang et al. [2016] removed modules on the training stage while Veit et al. [2016] removed modules on the testing stage.

Dynamic. The main characteristic shared by this category is that modules are pruned, on the fly, based on the input presented to the network. For this purpose, previous works have followed two distinguished directions. The former introduces decision gates (i.e., a softmax function) for each module composing the network. During inference, each decision gate decides to execute or skip its respective module [Shafiee et al., 2019; Veit and Belongie, 2020]. The latter learns agents that output binary decisions indicating, at once, the modules to be pruned [Wu et al., 2018; Wang et al., 2018b].

Both decision gate and agent-based approaches obtain different computational cost since the number of removed modules vary across images, as illustrated in Figure 3.3 (top and middle).

Compared to these approaches, our method to prune layers obtains competitive results; however, it achieves such results independently of the input given to the network, which can be particularly attractive in fixed-resources scenarios.

Static. Different from dynamic strategies, in this category, the same modules are always pruned regardless of the input given to the network, as shown in Figure 3.3 (bottom). Representative works, such as Veit et al. [2016]; Greff et al. [2017b]; Han et al.

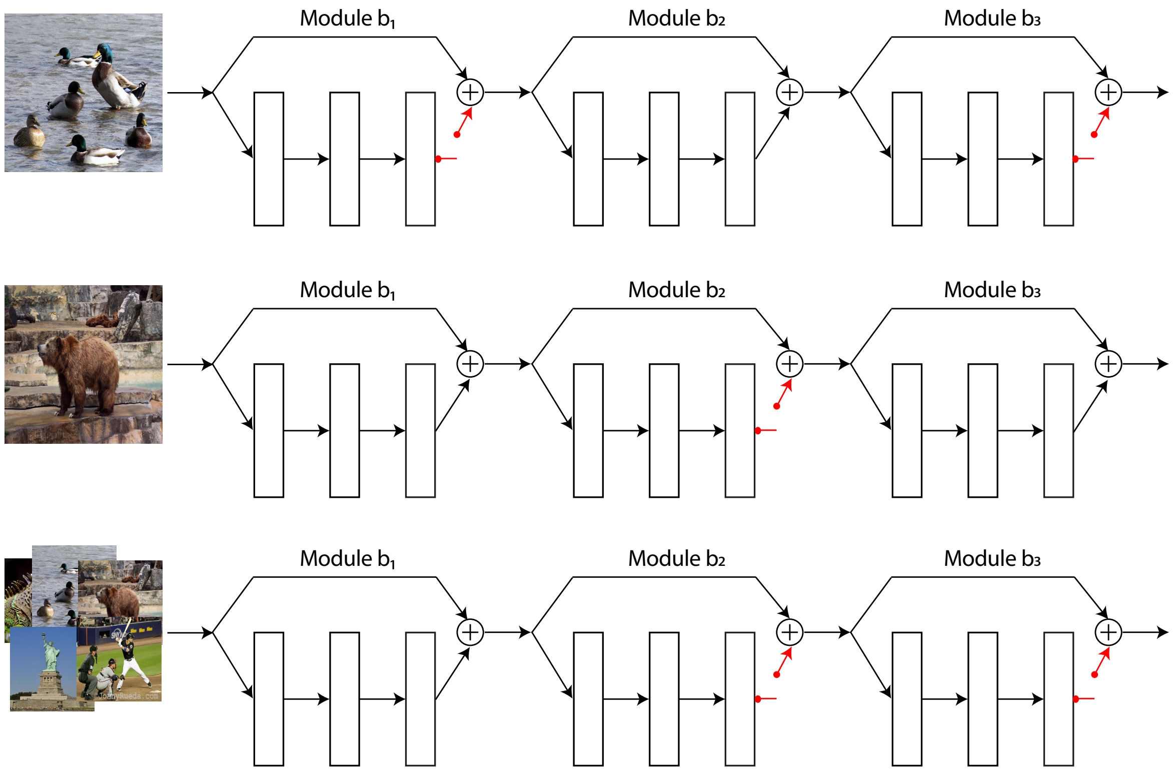


Figure 3.3. Strategies that remove modules (set of layers) from convolutional networks. The red switching indicates that a module was removed. **Top** and **middle**: Dynamic strategies, where modules are pruned according to the input image. For example, given a duck image (top), the modules b_1 and b_3 are removed. Given a bear image (middle), on the other hand, only the module b_2 is removed. **Bottom**: Static strategies, where modules are pruned regardless of the input presented to the network. For example, to all the images of the dataset, the modules b_2 and b_3 are always pruned.

[2017], remove module-by-module (i.e., one module at a time) and evaluate network accuracy. In particular, these works explore pruning from a theoretical perspective, i.e., they are not concerned with the computational cost. For example, Han et al. [2017] observed that increasing network width decreases accuracy loss after removing layers.

More recently, some works proposed to prune layers (in a static way) focusing on reducing computational demand [Huang and Wang, 2018; Fan et al., 2020]. For example, Huang and Wang [2018] added scaling factors to residual modules and, after optimizing them, removed modules associated with near-zero values. Fan et al. [2020] proposed to employ the stochastic depth regularization [Huang et al., 2016] to train an ultra-deep network. At the testing phase, their approach removes layers until satisfying a computational budget. Despite the positive results, such strategies require training a network from scratch, hindering applicability on off-the-shelf networks. In contrast, our approach does not require this training phase. Moreover, compared to Huang and Wang [2018], we provide a pruned network with better computational cost and accuracy drop.

3.2.3 Pruning Hybrid Structures

Strategies focusing on pruning neurons and filters are complementary and could benefit from each other. To the best of our knowledge, only the work by Cai et al. [2020] explores such a scenario. In their approach, the authors proposed to train a dense and computationally expensive network. Then, subnetworks are sampled from the dense network such that satisfying a given hardware constraint. Even though their approach achieves promising results, it is computationally prohibited since a large number of GPUs is required to train the dense network. We show that it is possible to eliminate both filters and layers in a two-step way: remove layers first and, then, remove filters. Unfortunately, due to the architecture employed by Cai et al. [2020], we are not able to compare our two-step strategy with them. More specifically, throughout our evaluation, we employ traditional and off-the-shelf networks (i.e., ResNet-based networks), hence, our results cannot be compared directly with Cai et al. [2020].

We highlight that Huang and Wang [2018] proposed to remove filters and layers by using scaling factors (as we explained before). However, their strategy removes either filters or layers, but not both. Thus, we do not consider it as a hybrid strategy.

3.3 Neural Architecture Search

Current pattern recognition methods are capable of achieving results better than humans [Deng et al., 2009; Parkhi et al., 2015; Badia et al., 2020]. Most methods, however, rely on domain expertise and intense human engineering. Consequently, there have been substantial efforts to automate the process of creating, training and deploying methods, namely Automated Machine Learning (AutoML) [Wistuba et al., 2017; Elsken et al., 2019; Yang and Shami, 2020]. In the context of AutoML for visual pattern recognition, many works have proposed Neural Architecture Search (NAS) strategies, which focus on discovering convolutional architectures automatically.

Given a criterion such as accuracy or fixed resource budget, NAS attempt to optimize the target criterion by training and evaluating a large set of candidate architectures. As a consequence, existing NAS approaches require vast computational resources, parallel processing infrastructure and take many days to process even with modern GPUs [Baker et al., 2017; Real et al., 2017; Zoph et al., 2018]. It is important to mention that, to alleviate this problem, most NAS approaches train the candidate

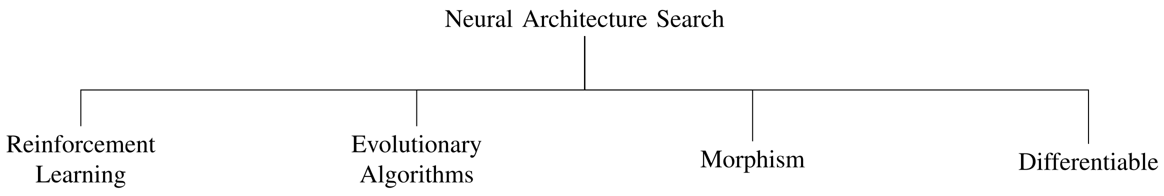


Figure 3.4. Existing neural architecture search strategies grouped by the type of mechanism employed to create candidate architectures.

architectures for few epochs (i.e., 10-20) [Real et al., 2017; Baker et al., 2017; Zoph et al., 2018; Li et al., 2020b], which might yield unreliable models during the search process [Dong and Yang, 2020; Sciuto et al., 2020; Yang et al., 2020b].

In general, NAS approaches employ different strategies such as reinforcement learning and evolutionary algorithms. Figure 3.4 summarizes the main strategies used by state-of-the-art NAS approaches.

3.3.1 Reinforcement Learning

To automate the process of creating convolutional networks, a typical technique is to use reinforcement learning (RL) to generate candidate architectures. Baker et al. [2017] employed this strategy for selecting types of layers and their parameters (i.e., depth, receptive field, stride). In contrast, Zoph et al. [2018] proposed to learn transferable architectures by applying the scheme of human-designed convolutional networks, in which layers share a similar structure. Their method uses a recurrent neural network to predict a cell, which consists of a set of layers (e.g., convolution, identity, pooling) and their connections. The final architecture is obtained by repeating the best cell N times, where N is manually predefined. The idea of searching cells rather than an entire architecture is still widely employed by modern NAS approaches [Chen et al., 2019; Vahdat et al., 2020; Yang et al., 2020b].

Similarly to Zoph et al. [2018], we show that our NAS is capable of building architectures that generalize well across datasets, such that we can learn a model on a small dataset and transfer it to large datasets. More importantly, our method is orthogonal to this approach (hence orthogonal to most NAS) in the sense that we discover N given a predefined cell.

3.3.2 Evolutionary Algorithms

Since using neural networks to learn architectures is time-consuming and requires careful parameter setting [Wu et al., 2018; Dong and Yang, 2019], many works employ evolutionary algorithms to guide the search [Real et al., 2017; Dong and Yang, 2020; Yang et al., 2020b]. In general, an evolutionary framework builds convolutional networks by considering each candidate architecture as an individual of the population and operations such as inserting or removing layers/connections are considered possible mutations [Real et al., 2017]. Improving upon this idea, Yang et al. [2020b] proposed to share parameters between individuals of the population and employ a Pareto-front sorting strategy for selecting the non-dominated candidates — candidates that are no worse than any other on a given performance metric. As expected, their approach is able to discover high-performance architectures in a few hours.

Compared to this family of strategies, our NAS approach is able to design competitive architectures by exploring one order of magnitude fewer *individuals*. More specifically, our NAS discovers more accurate and efficient architectures than Real et al. [2017] while evaluating 10× fewer models. Compared to Yang et al. [2020b], our method builds more parameter-efficient architectures with slightly inferior accuracy.

3.3.3 Morphism

Although RL and evolutionary NAS are capable of building accurate models, their search process is computationally expensive since each candidate architecture needs to be trained from scratch in most cases. To handle this problem, recent works attempt to transfer the knowledge of previous pre-trained networks to the candidate architectures [Elsken et al., 2018; Kandasamy et al., 2018; Jin et al., 2019]. To this end, a popular technique is network morphism, which creates new networks by means of function-preserving transformations [Chen et al., 2016]. In essence, network morphism allows the original and the modified network to have the same prediction ability. Elsken et al. [2018] employed network morphism to initialize architectures, aiming at reducing the cost of training them from scratch. Cai et al. [2018] applied RL to generate transformations on an initial network, for example, DenseNet [Huang et al., 2017]. As suggested in their work, using an existing and pre-trained architecture is an efficient manner of exploring the search space, being possible to reuse its weights as well as its successful initial structure. Our method takes advantage of these observations, but focuses exclusively on depth. Kandasamy et al. [2018] and Jin et al. [2019] employed Bayesian optimization to guide transformations during the search process. While computationally efficient, these approaches yield low-accuracy architectures. To achieve competitive results, many hyper-parameters need to be set manually [Jin et al., 2019], rendering an unfair comparison with other NAS approaches.

Compared to morphism-based NAS, our method enables reusing weights of pre-trained convolutional networks more easily because it does not require a careful selection of the morphism operations.

3.3.4 Differentiable

In this category of NAS, the architecture and its weights are learned jointly during the gradient descent optimization [Brock et al., 2018; Dong and Yang, 2019; Chen et al., 2019; Liu et al., 2019a]. For this purpose, differentiable NAS approaches convert the discrete search space into a continuous one such that the elements (i.e., number of filters, stride and connections) composing an architecture can be viewed as parameters to be learned. During the gradient optimization phase (a.k.a search phase), one candidate architecture is built at the end of each training epoch; thus, the number of epochs defines the number of candidate architectures. To further improve efficiency and reduce memory demand, the candidate architectures are shallow, but, after the search phase, the final architecture has its depth increased to improve representation capacities. In contrast to this pipeline, Chen et al. [2019] proposed to increase depth during the search phase. This is achieved by increasing network (candidate architectures) depth after some search iterations.

In general, compared to other approaches, differentiable NAS considerably improves the time required for discovering architectures. Particularly, compared to reinforcement- and evolutionary-based algorithms, differentiable NAS build architectures requiring one order of magnitude fewer GPU-days [Dong and Yang, 2019; Chen et al., 2019; Vahdat et al., 2020]. On the other hand, these approaches are parameter sensitive, which means that they need careful tuning of their hyper-parameters [Dong and Yang, 2019]. Besides, compared to other NAS, the search space needs to be drastically reduced due to memory constraints [Wan et al., 2020].

We show that our NAS leads to competitive architectures without requiring a careful parameter setting. Specifically, we need to set only two parameters, which exhibit a small influence on the accuracy of the candidate architectures. In terms of computational cost, our NAS is computationally efficient, as our search space consider exploring depth only. Concurrently to our work, Chen et al. [2019] also focus on adjusting depth, however, they increase the depth uniformly (similar to human-designed architectures) while we learn a depth for different levels of the network.

3.4 Exploring Layers in Convolutional Networks

The idea of incorporating multiple levels of features has received great attention in computer vision tasks [Hariharan et al., 2015; Kong et al., 2016; Bell et al., 2016; Huang et al., 2017; Wang et al., 2018a; Sindagi and Patel, 2019; Huang et al., 2019]. Previous works observed that combining features from early and deep layers improves data representation [Bell et al., 2016; Kong et al., 2016; Zhou et al., 2020]. Such combination, however, could be non-trivial since features (feature maps) from early and deep layers present different spatial dimensions and lies on a high dimensional space.

To address the aforementioned problem, some strategies (referred to as HyperNets) insert operations after each layer to be combined, as shown in Figure 3.5. For instance, Bell et al. [2016] employed 1×1 convolution to normalize feature maps from previous layers as well as reducing their dimensionality. Instead, Kong et al. [2016] used max-pooling and deconvolution layers to re-scale all feature maps for the same spatial resolution. Then, these re-scaled feature maps feed convolutional layers, which in turn are connected to fully-connected layers. Interestingly, Kong et al. [2016] observed that adjacent layers are correlated and, when combined, do not enhance data

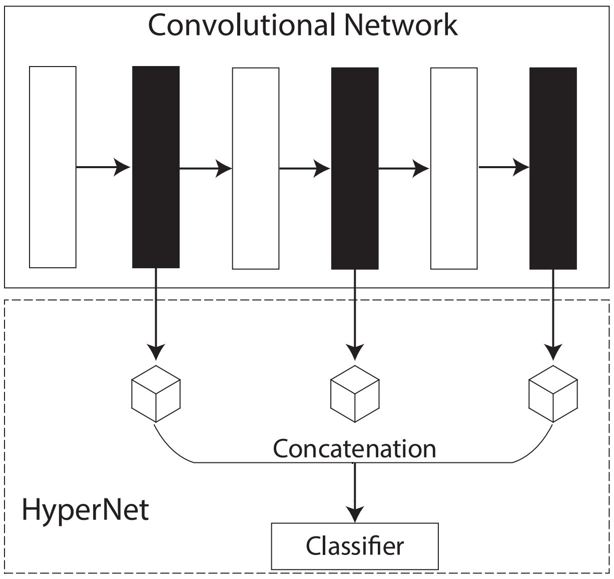


Figure 3.5. Overall process to build a HyperNet. After setting the layers to be combined (represented by black boxes), operations such as convolution, pooling or re-scaling are applied to their outputs yielding a feature map (represented by a cuboid). Then, these feature maps are concatenated and presented to a classifier (e.g., a fully connected layer). In this process, since the original architecture (top) is unchanged the network topology is preserved, thus enabling applicability on off-the-shelf-networks.

representation. Similarly, Sindagi and Patel [2019] demonstrated that combining layers by concatenating their feature maps leads to better results than other operations, e.g., addition.

An interesting aspect of the HyperNets above is that they preserve the topology (see Figure 3.5) of the original network, thus enabling applicability on off-the-shelf networks. However, the complexity and computation of the network increase considerably due to the additional operations such as 1×1 convolutions. Additionally, features from earlier layers are high-dimensional, which further increases the number of floating-point operations. Therefore, such strategies might be prohibitive for applications with limited memory and low computational power. Our HyperNet, on the other hand, is capable of combining multiple layers at negligible additional cost and handling the problem of high dimensionality as well.

Instead of exploring multiple layers in off-the-shelf networks, a parallel line of research focuses on designing convolutional architectures to encode multiple levels of features, namely multi-scale networks. A representative approach in this category is the work by Huang et al. [2017]. In their approach, a convolutional layer takes as input the feature maps of its preceding layers, as illustrated in Figure 3.6. This architecture topology increases significantly the computational cost, as a layer operates on a high-dimensional input. Improving upon this model, similar to HyperNets, Wang et al. [2018a] proposed to reduce the computational overhead by carefully reducing the dimensionality of the layers before concatenating them. Surprisingly, their architecture obtained better performance than lightweight networks such as MobileNet [Howard et al., 2017].

It is important to mention that due to the design of multi-scale networks it is not possible to compare them with HyperNets approaches.

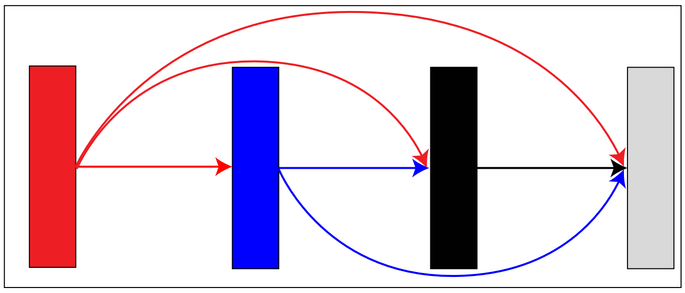


Figure 3.6. Overview of a multiscale convolutional network. This architecture encodes features (indicated by colored arrows) from shallow and deep layers. For this purpose, a layer i takes as input the feature maps from all preceding layers. For example, the last convolutional layer (gray box) receives as input its preceding layers (red, blue and black boxes).

3.5 Incremental Dimensionality Reduction

Traditional dimensionality reduction methods are unsuitable for large datasets since all the data need to be available in advance and this could be impractical due to memory constraints. To handle this problem, many works have proposed incremental dimensionality reduction methods. These approaches estimate the projection matrix using a single data sample (or a subset) at a time while keeping some properties of the traditional dimensionality reduction methods [Weng et al., 2003; Zeng and Li, 2014].

To enable PCA to operate in an incremental scheme, Weng et al. [2003] proposed to compute the principal components without estimating the covariance matrix, which is unknown and impossible to be calculated in incremental methods. For this purpose, their method, named *Candid Covariance-free Incremental Principal Component Analysis* (CCIPCA), updates the projection matrix for each sample x, replacing the unknown covariance matrix by the sample covariance matrix (xx^T). While CCIPCA provides a minimum reconstruction error of the data, it might not yield very discriminative subspaces since label information is ignored (similarly to traditional PCA) [Martínez and Kak, 2001].

To achieve discriminability, incremental methods based on LDA have been proposed [Hiraoka et al., 2000; Lu et al., 2012]. In particular, this class of methods is less explored since they present some problems (e.g., the sample size problem), which makes them infeasible for some tasks. Different from incremental LDA methods, incremental PLS methods are more flexible and present better results [Zeng and Li, 2014]. Motivated by this, Arora et al. [2016] proposed an incremental PLS based on stochastic optimization (SGDPLS), where the idea is to optimize an objective function using a single sample at a time. Similarly to Arora et al. [2016], Stott et al. [2017] proposed applying stochastic gradient maximization on NIPALS, extending it for incremental processing. Even though they present promising results on synthetic data, their approach presented convergence problems when evaluated on real-world datasets. Thus, we consider only the approach by Arora et al. [2016], which was the one that converged for several of the datasets evaluated and presented better results.

While SGDPLS is effective, SGD-based methods applied to dimensionality reduction are computationally expensive and present convergence problems, as demonstrated by Weng et al. [2003] and, Zeng and Li [2014]. In addition, this class of approaches requires careful parameter tuning and their results are often sensitive to the type of dataset [Weng et al., 2003]. To address convergence problems in SGD-based PLS, Zeng and Li [2014] proposed to decompose the relationship between independent and dependent matrices (variables) into a sample relationship (i.e., a single sample with its label).

This process is performed only to compute the first component, while the higher-order components are estimated by projecting the first component onto an approximated covariance matrix using a few PCA components. As we mentioned earlier, since traditional PCA cannot be employed in incremental methods, Zeng and Li [2014] used CCIPCA to reconstruct the principal components of the covariance matrix.

In contrast to the aforementioned incremental PLS methods, our incremental PLS presents superior performance in both accuracy and execution time for estimation of the projection matrix, which is an important requirement for time-sensitive and resource-constrained tasks. Compared to the method of Zeng and Li [2014] (called *incremental PLS* - IPLS), we show that the proposed method separates the data better since our higher-order components keep the properties of traditional PLS.

3.6 Feature Selection

Another line of research widely employed to reduce computational cost is feature selection. Feature selection consists of ranking and selecting a subset of features based on a specific criterion. In order to discover the most relevant features, many techniques have been proposed such as LASSO regression [de Geer, 2008; Rooyen et al., 2015], mutual information [Yang and Moody, 1999; Fleuret, 2004] and eigenvector centrality [Roffo and Melzi, 2016a,b]. Such techniques exhibit different accuracies and computational complexity for ranking the features. Among the state-of-the-art feature selection techniques, the strategy by Roffo et al. [2015, 2017, 2020] is the most successful in terms of accuracy and efficiency. Thus, throughout this section, we focus on describing their feature selection framework.

Roffo et al. [2015] proposed to interpret feature selection as a graph problem. In their method, named *Infinity Feature Selection* (infFS), each feature represents a node in an undirected fully-connected graph and the paths in this graph represent the combinations of features. Following this model, the goal is to find the best path taking into account all the possible paths (in this sense, all the subsets of features) on the graph, by exploring the convergence property of the geometric power series of a matrix. Improving upon this model, Roffo et al. [2017] suggested quantizing the raw features into a small set of tokens before applying the process of Roffo et al. [2015]. By using this pre-processing, their method (referred to as *Infinity Latent Feature Selection* - ilFS) achieved even better results than infFS. Recently, Roffo et al. [2020] presented a more efficient version of infFS, which considers supervised (infFS$_{\mathrm{S}}$) and unsupervised (infFS$_{\mathrm{U}}$) scenarios.

Although the framework by Roffo et al. [2015, 2017, 2020] achieved state-of-the-art results, some works have demonstrated that PLS coupled with Variable Importance in Projection attains promising results in feature selection [Schwartz et al., 2009; de Melo et al., 2013; dos Santos Junior et al., 2016; Diniz and Schwartz, 2020]. We show that the proposed incremental PLS with VIP achieves comparable results when compared to PLS+VIP as well as with state-of-the-art feature selection techniques.

We highlight that while there exist many other feature selection techniques, the works by Roffo et al. [2015, 2017, 2020] outperform (or are in par with) most existing feature selection techniques. Therefore, we limit our comparison only with these works. In addition, the complexity of other feature selection techniques grows quickly as the number of samples increases, thus they are prohibitive for large datasets such as the ones in computer vision [Krizhevsky et al., 2009; Deng et al., 2009; Huang et al., 2012].

Chapter 4

Proposed Approaches

In this chapter, we introduce the proposed approaches to improve computational cost and data representation of convolutional networks. We start by describing our pruning approach that locates potential structures (neurons and layers) to be removed from convolutional networks. Then, we present our neural architecture search approach that designs high-performance networks automatically. Afterward, we describe our HyperNet approach that captures different levels of representation distributed over early and deep layers of the network. Finally, we introduce our incremental Partial Least Squares that learns the low-dimensional latent space by using a single sample at a time. Throughout the chapter, we use the mathematical definitions stated in Chapter 2.

4.1 Pruning Approaches

Problem Definition. Let $\mathcal{F}$ be a convolutional network with L layers, where the number of neurons in each layer $f_i \in \{1, 2, ..., L\}$ is defined by $|f_i|$. Define $\mathcal{F}'$ a network without some structures of $\mathcal{F}$ such that $|f'_i|_{i=1}^{L'} \leq |f_i|_{i=1}^{L}$ (pruning filters) or $L' < L$ (pruning layers). Thus, $\mathcal{F}'$ is an efficient and lower-complexity version of $\mathcal{F}$. Figure 4.1 illustrates $\mathcal{F}'$ yielded from the removal of filters and layers of $\mathcal{F}$.

Our target is to identify and remove structures from $\mathcal{F}$ that preserve as much accuracy as possible, which means yielding $\mathcal{F}'$ such that its accuracy is close (ideally superior) to $\mathcal{F}$.

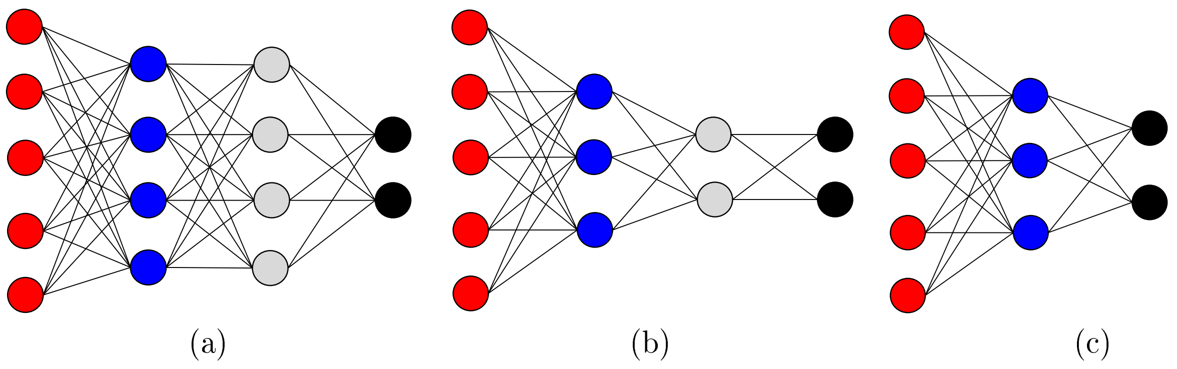


Figure 4.1. Pruning approaches considering different structures: neurons or layers. (a) Original, unpruned, network. (b) Pruning approach that removes neurons, i.e., $|f'_i|_{i=1}^{L'} \leq |f_i|_{i=1}^{L}$. (c) Pruning approach that removes layers, i.e., $L' < L$. Because (b) and (c) have fewer neurons and layers (they are less complex than (a)), such networks are an efficient version of (a).

4.1.1 Pruning Filters

This section defines the proposed method to eliminate filters in convolutional networks. We start by describing the representation of filters as feature vectors. Then, we introduce how to measure filter importance. Finally, we describe how to remove filters with low importance. Figure 4.2 shows an overview of our strategy for removing filters.

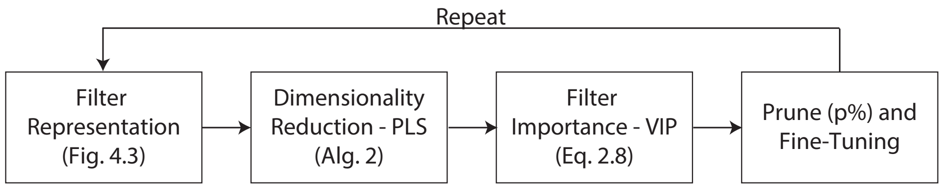

Repeat
Filter Representation (Fig. 4.3)
Dimensionality Reduction - PLS (Alg. 2)
Filter Importance - VIP (Eq. 2.8)
Prune (p%) and Fine-Tuning

Figure 4.2. Overview of our strategy for removing filters from convolutional networks. First of all, the filters composing a convolutional network are represented as feature vectors. Then, we project these feature vectors onto a compact space using PLS. Finally, we assign an importance score for each feature (filters) using VIP and remove $p\%$ of the filters based on this score.

Filter Representation. The first step in our pruning filter method is to represent filters that compose the network as feature vectors. For this purpose, we present the training data to the network and interpret the feature maps of each convolutional filter as a feature vector (or a set of features). These feature maps are high dimensional and might lead to memory constraints. However, it is well-known that pooling operations can encode the most important information about large feature maps [Hu et al., 2018; Li et al., 2019b; Veit and Belongie, 2020]. Therefore, we apply a pooling operation to reduce their dimension. We consider the following pooling operations: global max-

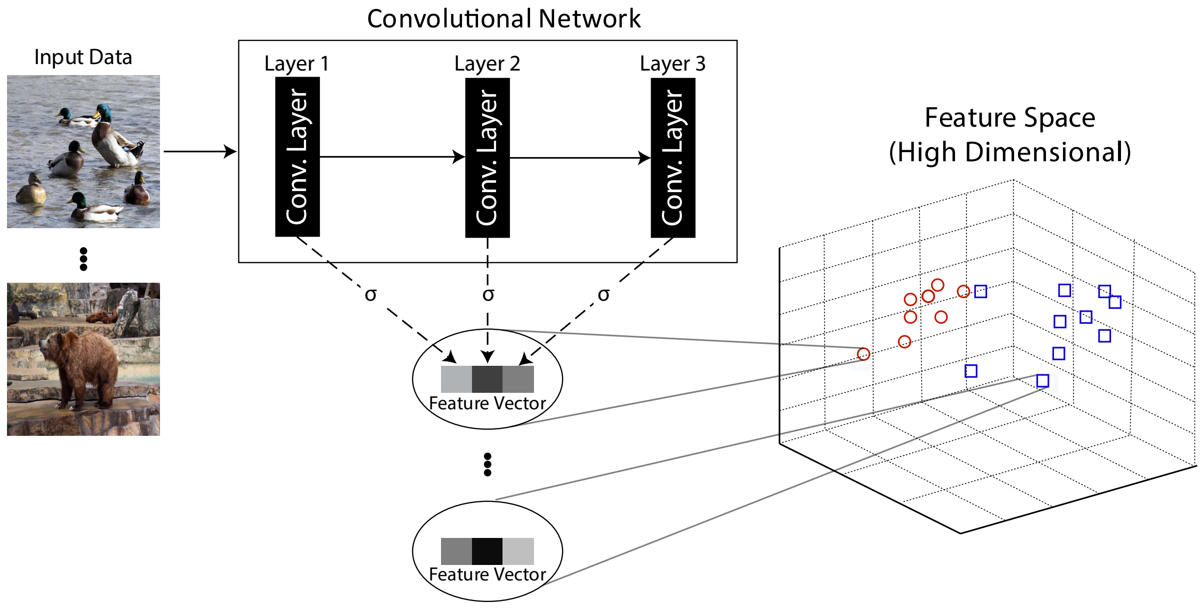


Figure 4.3. Representation of convolutional filters as feature vectors. First, we present samples to the network and extract feature maps from the convolutional layers. Then, we apply a pooling operation (indicated by σ) on these feature maps and interpret the output of the pooling as feature vectors. For simplicity, each layer of the network consists of one filter only (one dimension of the feature space). Red and blue points denote positive and negative samples, respectively.

pooling, global average pooling and max-pooling 2×2. Afterward, the output of the pooling operation is interpreted directly as one feature (when using the global pooling operations) or as a set of features (when using the max-pooling 2×2). Specifically, each filter is represented by its feature map followed by the pooling operation. Finally, the filter representations from different layers are concatenated to compose the final feature vector that represents all filters of the network. Figure 4.3 illustrates this process.

The intuition for using the feature map as a feature is that we are able to measure its relationship with the class label on the latent space (PLS criterion). In this way, a filter associated with a feature with low relationship might be removed.

Filter Importance. After executing the previous steps, we have created a high dimensional feature space, representing all convolutional filters of the network at once. Then, we measure the filter importance score to remove the ones with low importance. To this end, we project the high dimensional space onto a latent space using PLS and employ the VIP (Equation 2.8) technique to estimate the contribution of each feature in generating the latent space. Recall that, following the modeling performed in the first step of our method, each feature corresponds to a filter. In particular, when using the max-pooling operation as filter representation, we have a set of features for each

```
Algorithm 3: Pruning Filters from Convolutional Networks
  Input  : Convolutional Network $\mathcal{F}$
           Pooling operation $\sigma$
           Number of iterations $k$
  Output: Pruned Convolutional Network $\mathcal{F}'$
1  $\mathcal{F}' \leftarrow \mathcal{F}$
2  for $j \leftarrow 1$ to $k$ do
3      $X = \{\}$
4      for $f_i \in \mathcal{F}'$ do
5          $o_i \leftarrow$ feature maps from $f_i$
6          $X \leftarrow X \cup \{\sigma(o_i)\}$
7      end
8      Estimate importance score of each feature (filter) of $X$ using PLS+VIP
9      $\mathcal{F}' \leftarrow \mathcal{F}' \setminus p\%$ lowest-score filters
10     Fine-tune $\mathcal{F}'$
11 end
```

filter; therefore, the final score to a filter on this representation is the average of its scores.

Prune and Fine-tune. Given the importance of all filters that compose the network, we can remove $p\%$ of the filters associated with low scores. The removal stage consists of creating a new network $\mathcal{F}'$, without the discarded filters, and transferring the weights of the kept filters. In other words, $\mathcal{F}'$ *inherits* the weights of the kept structures of $\mathcal{F}$. Finally, we perform some stages of fine-tuning in $\mathcal{F}'$ to compensate for the structures that have been removed. An alternative to fine-tuning is training the pruned network from scratch. The latter, according to recent observations, leads to worse results than fine-tuning [Liu et al., 2019b; Evci et al., 2019; Fan et al., 2020]. Our experiments corroborate this observation, in which training the pruned network from scratch does not bring notable improvements to our method.

The process above composes one iteration of our method. Such a process can be repeated until a specific number of iterations is reached, where the input network to the next iteration is the pruned network of the previous iteration. Algorithm 3 summarizes all the steps of the proposed method to prune filter.

4.1.2 Pruning Layers

This section defines the proposed method to eliminate layers in convolutional networks. We start by describing the representation of layers as feature vectors. Then, we introduce how to measure layer importance. Finally, we describe how to remove layers with

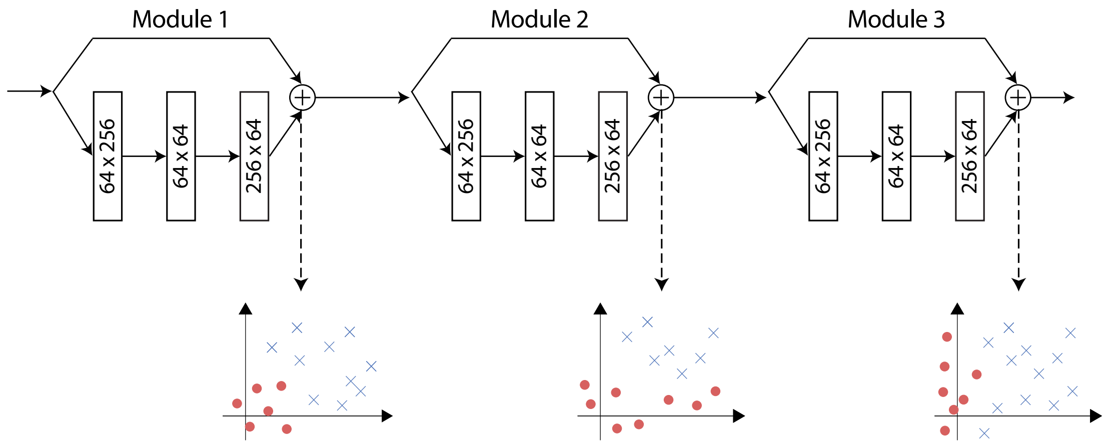


Figure 4.4. Representation of the modules (set of layers) as features. At the end of each module (the add operation '+') we extract the feature maps and interpret them as feature vectors.

low importance.

Layer Representation. Modern convolutional architectures consist of modules — stack of layers with the same configuration (i.e., number of filters and spatial resolution). Following these architectures, we are unable to remove single layers within modules due to implementation details[1] (incompatible dimensions). Fortunately, based on prior works [Veit et al., 2016; Greff et al., 2017b; Han et al., 2017; Fan et al., 2020], we can eliminate entire modules without degrading the representation capabilities of the network.

To eliminate modules from convolutional networks, the first step in our approach is to represent modules as features. Similar to the process for representing filters as features, we could interpret the feature maps of the layers composing b_i as a set of features. However, the last layer of a module contains information about the entire module (i.e., its preceding layers) [Veit et al., 2016; Huang et al., 2017; Greff et al., 2017a]. Thereby, to represent each module b_i, we can extract feature maps considering only its last layer, as illustrated in Figure 4.4.

Layer Importance. Given a feature map X_i from a module (its last layer) b_i, the next step in our method is to measure the importance of the features composing X_i. This way, we are estimating the importance of b_i to which we could remove the least important ones. For this purpose, we project X_i onto a low-dimensional space using PLS and, then, employ VIP to estimate the contribution of each feature in generating this space. Such a process will provide a set of importance scores; thus, we average these values to compose the final importance of b_i.

[1]We refer the reader to Appendix B for additional details.

Prune and Fine-tune. With the importance of all modules that compose the network, we can remove $p\%$ of the modules associated with low scores. The removal stage is similar to the process of removing filters — we create a new network $\mathcal{F}'$, without the discarded modules, and transfer the weights of the kept modules. Finally, we perform some stages of fine-tuning in $\mathcal{F}'$ to compensate for the structures removed.

4.2 Neural Architecture Search

Problem Definition. Let $\mathcal{F}$ be a convolutional network composed of S stages. Each stage $s_i \in S$ consists of b_i modules (set of layers as illustrated in Figure 4.5), which in turn define the depth of stage s_i. Following the structure of modern architectures, the layers within a stage operate on the same input/output resolution (i.e., their feature maps have the same dimension). In previous works, including NAS, b is the same for all stages or defined empirically. For instance, ResNet39 has six residual blocks in each of its stages (i.e., $b_{i\in\{1,\ldots,S\}} = 6$), as shown in Figure 4.6 (top). Our target is to design architectures by learning the number of modules b_i for each stage s_i, as illustrated in Figure 4.6 (bottom).

4.2.1 Stage-wise Architecture Search

This section defines the proposed method to automatically design convolutional networks. We start by describing the cell modules, which are the components employed

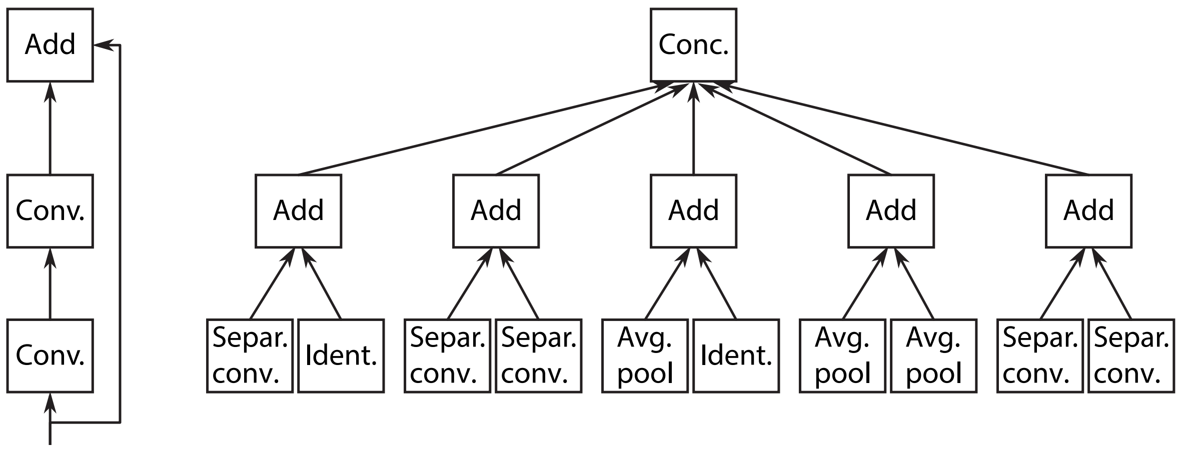


Figure 4.5. Left. Residual modules employed in ResNets [He et al., 2016]. **Right.** Cell modules employed in NASNets [Zoph et al., 2018]. Add indicates element-wise addition operation. Sep. Conv. indicates depthwise separable convolutions. Ident. indicates that the received input is propagated with no transformation. Avg. pool and Conc. indicate average pooling and concatenation operation, respectively.

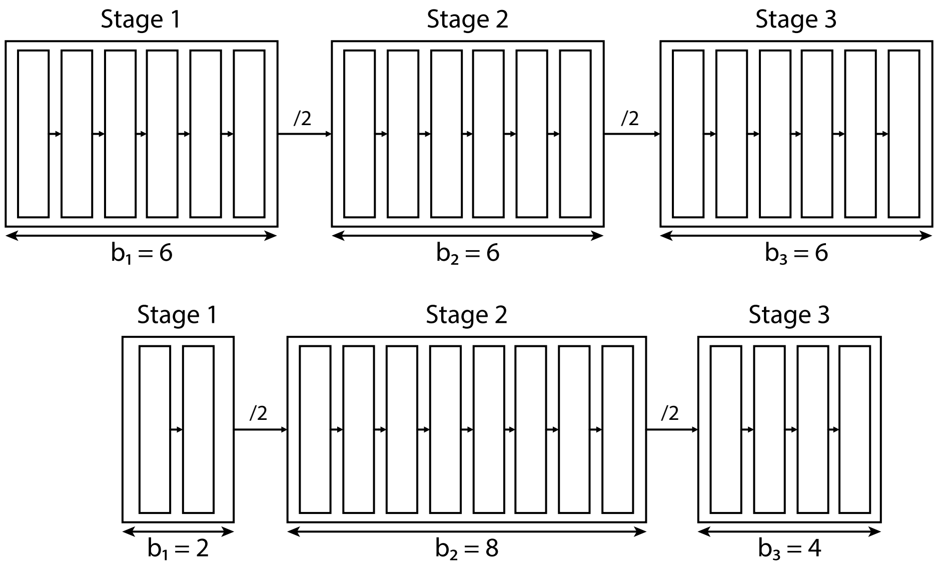


Figure 4.6. Top. Structure of modern architectures, in which depth (number of modules) is the same for all stages. **Bottom.** Structure of our architectures, in which the depth of each stage is adjusted based on the importance of its features. Following these structures, the number of modules in each stage defines its depth. In this example, the mid-stage of our architecture is more important as it is deeper, while the early-stage is less important as it is the shallower.

to build our architectures. Then, we introduce how to measure the importance of these cell modules. Finally, we describe how to insert cell modules, which means generating candidate architectures, and how to transfer the knowledge of pre-trained networks to such architectures, respectively.

Modules. The first step in our neural architecture search approach is to define a module type. We consider two types of modules: residual blocks from ResNet [He et al., 2016] (Figure 4.5 left) or cells from NASNet [Zoph et al., 2018] (Figure 4.5, right). We do not explore the combination of both, meaning that the discovered architecture is either ResNet-based or NASNet-based. We limit our experiments to these two types of modules due to their relevance in modern architectures and because the combination of different modules can generate incompatible dimensions in the feature maps, thus requiring a careful implementation [Wan et al., 2020].

Stage Importance. The next step in our method is to measure the importance score for each stage $s_i \in S$. For this purpose, we apply a process similar to Figure 4.3, which is the following. Given a stage s_i of a convolutional network, we present the training samples to the network and extract the feature maps from the last layer of this stage. As before, the reason for considering the last layer is that it contains information about previous layers, hence, about the entire stage [Veit et al., 2016; Huang et al., 2017; Greff et al., 2017a]. It is important to mention that this claim is valid only when the identity (i.e., skip-connection layer) is propagated to successive layers [Veit et al., 2016].

Let X_i be the features of s_i estimated following the procedure above. The next step is to compute the importance of these features and average their values to compose the final importance score for each stage. Specifically, by estimating the importance of X_i we are estimating the importance of the stage s_i. Such importance is estimated by presenting X_i to PLS followed by VIP (similar to the process employed to remove structures, Section 4.1).

Adjusting Stage Depth. Once we are able to estimate the importance score α_i for each stage s_i, the next step is to build a candidate architecture by adjusting the depth of each stage based on its importance. To this end, we first create a network $\mathcal{F}$ with S stages ($|s| = S$), each one containing the same number of modules, for example, by employing $S = 3$ and $|b_i|_{i=1}^{S} = 6$ (i.e., ResNet39 in Figure 4.6, top). Then, we create a temporary architecture T by increasing the depth of s_i to $b_i + \delta$, where δ is the growth step, i.e., the number of modules that can be inserted in a stage in a single iteration. Afterward, we compute the importance scores $\alpha_{\mathcal{F},i}$ and $\alpha_{T,i}$, for each stage s_i of the initial and temporary architectures, respectively. Finally, we update b_i to $b_i + \delta$ if $\alpha_{T,i} > \alpha_{\mathcal{F},i}$ and create a candidate architecture $\hat{\mathcal{F}}$ using the updated b_i. It is worth mentioning that the importance scores are comparable in terms of magnitude. The idea behind this incremental process is to measure if increasing depth will improve the representation learned by the candidate architecture.

The process above composes one iteration of our method, where at the end of each iteration one candidate architecture is discovered. The input for the next iteration is the candidate architecture designed with the values of b_i updated. Algorithm 4 summarizes all the steps of the proposed method.

In practice, given k iterations, our method creates only $2k+1$ architectures, which is an order of magnitude fewer than state-of-the-art NAS approaches.

Weight Transfer Technique. Similar to previous NAS approaches [Real et al., 2017; Zoph et al., 2018], the process of creating a model consists of training it from scratch for some epochs, which can be computationally prohibitive for large datasets such as ImageNet. However, since our method employs the same structure (i.e., modules) of existing architectures, we propose to transfer the knowledge (weights) from a pre-trained network to our candidate architecture. For example, when employing residual modules, our candidate architecture can use the weights of a pre-trained ResNet. This way, instead of training from scratch, we only need to adjust the weights by fine-tuning for a few epochs to compensate changes in the magnitude of the feature maps [Veit

Algorithm 4: Stage-Wise Neural Architecture Search

```
Input : Number of iterations $k$
        Number of stages $S$
        Initial number of modules per stage $b_0$
        Growth step $\delta$
Output: Set of candidate architectures $\mathbb{C}$

1  Create $\mathcal{F}$ with $S$ stages and $b_0$ modules each
2  for $j \leftarrow 1$ to $k$ do
3      Create $T$ with $S$ stages and $b_i + \delta$ modules each
4      for $i \leftarrow 1$ to $S$ do
5          Compute importance scores $\alpha_{\mathcal{F},i}$ and $\alpha_{T,i}$
6          if $\alpha_{T,i} > \alpha_{\mathcal{F},i}$ then
7              $b_i \leftarrow b_i + \delta$
8          end
9      end
10     Create $\hat{\mathcal{F}}$ with $S$ stages and the updated $b_i$
11     $\mathcal{F} \leftarrow \hat{\mathcal{F}}$
12     $\mathbb{C} \leftarrow \mathbb{C} \cup \{\hat{\mathcal{F}}\}$
13 end
```

et al., 2016; Greff et al., 2017a]. One restriction of this strategy is that the depth of a stage (number of modules) of the candidate architecture cannot exceed the depth of the network that is providing the weights. In practice, we show that this does not occur as our candidate architectures are shallower than existing networks.

In essence, our weight transfer technique is similar to the morphism strategy, however, this solution is simpler since it does not require careful selection of the morphism operations [Cai et al., 2018; Jin et al., 2019].

4.3 HyperNet Approach

Problem Definition. Let X_i be an output (feature map) of a specific layer $f_i \in \{1, 2..., L\}$ from a convolutional network $\mathcal{F}$ of L layers. Define $\mathbb{O}$ a set of feature maps X_i such that $|\mathbb{O}| > 1$. We assume that $\mathbb{O}$ provides better data representation than using a single X_i. Figure 4.7 supports this assumption. Our target is to efficiently and properly yield $\mathbb{O}$, which means combining multiple X_i in an efficient yet accurate way.

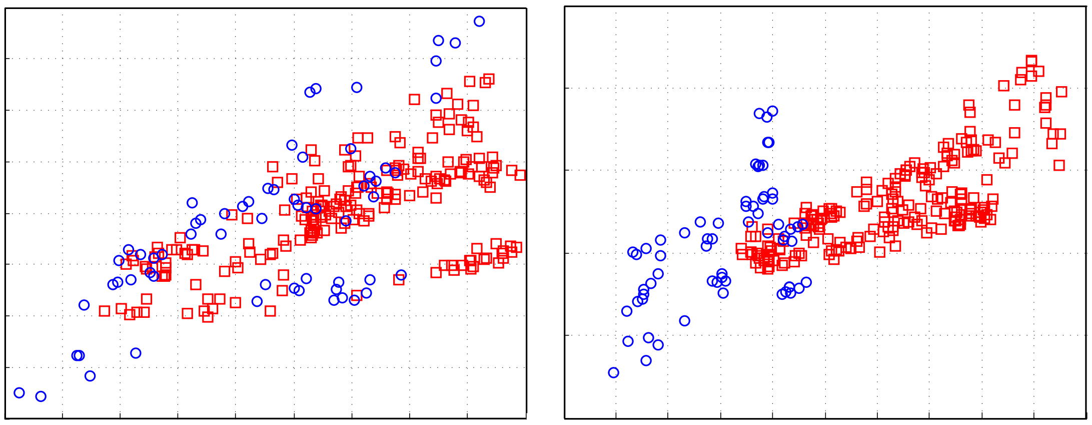


Figure 4.7. Projection of two categories onto the two first components of Partial Least Squares. **Left.** Projection using feature maps from the last convolutional layer (i.e., X_L). **Right.** Projection using feature maps from early and the last layers (i.e., $\mathbb{O}$). The feature space is better separated when features from early and deep layers are combined. This happens due to additional clues provided by the low-level information (early layers).

4.3.1 Latent HyperNet

This section defines the proposed Latent HyperNet approach to combine low-level and refined information distributed over the layers of convolutional networks. We start by describing the process for selecting the layers to be combined. Then, we introduce how to combine these layers efficiently.

Selecting Layers. The first step is our Latent HyperNet (LHN) is to define a set of layers, $l \subset L$, to be combined. This is a typical step in HyperNet approaches and it is necessary because some early layers contain simple patterns (i.e., edges), which do not contribute to the classification but increase computational cost. In addition, as observed by previous works [Kong et al., 2016; Hariharan et al., 2015], adjacent layers are strongly correlated and can harm the data representation. Therefore, setting the layers to be combined is more appropriate than using all of them.

Combining Layers. Once we have set the layers l, we use the feature maps X_i of each layer $f_i \in l$ to learn a PLS model. Such feature maps are high dimensional, which reinforces the employment of PLS as it is proper for these cases.

Following this model, each $f_i \in l$ will have a PLS model (i.e., a projection) associated with it, as shown in Figure 4.8. Alternatively, we might concatenate the feature maps from $f_i \in l$ and then, learn a single PLS model. However, the memory consumption would increase significantly since the result of this concatenation is an even higher-dimensional space, hence, this strategy might be prohibitive for memory-constrained applications. In addition, we will show that the two strategies for learning

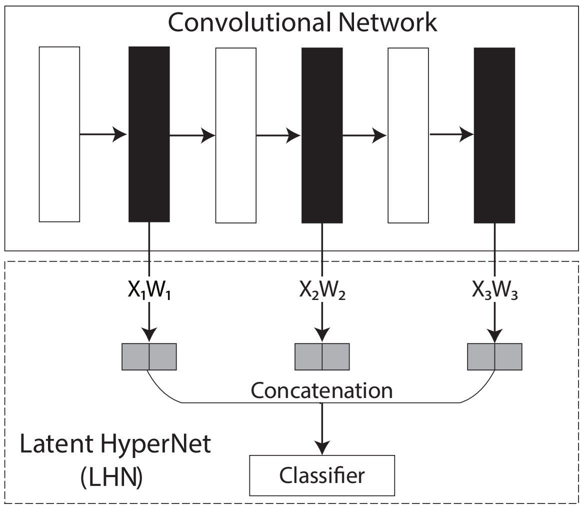


Figure 4.8. Process to build the Latent HyperNet considering a 3-layer convolutional network. After setting the layers to be combined (represented by black boxes), we learn a PLS projection (W_{th}) using their feature maps (X_{th}). Then, we project ($X_{th}W_{th}$), concatenate and present the low-dimensional feature maps to a classifier. In this example, each PLS projects the high-dimensional feature maps onto two dimensions.

PLS present similar performance.

After executing the above steps, we project the feature maps X_i on its respective PLS model yielding compact representations of X_i, which in turn are concatenated in $\mathbb{O}$. In summary, before inserting X_i into $\mathbb{O}$ we reduce its dimensionality using PLS. Algorithm 5 summarizes these steps.

Importantly, our LHN neither modifies the design nor the learned weights of the network, as shown in Figure 4.8, enabling it to be easily adaptable to any network. Additionally, in contrast to HyperNet approach of Kong et al. [2016], our LHN allows the combination of any layer that composes the network, for example, convolutional

Algorithm 5: Latent HyperNet

```
Input  : Convolutional Network $\mathcal{F}$
         Set of layers to be combined $l$
Output: Latent features $\mathbb{O}$
1 for $f_i \in l$ do
2     $X_i \leftarrow$ feature maps from $f_i$
3     if Training phase then
4         Find PLS projection $W_i$
5     end
6     $\mathbb{O} \leftarrow \mathbb{O} \cup \{X_i W_i\}$
7 end
```

and fully connected layers.

4.4 Incremental Partial Least Squares

Problem Definition. Let $W(w_1, w_2, ..., w_c)$ be a projection matrix that projects the high dimensional space onto a low c-dimensional space. Considering that W was obtained by PLS, which means that each component w_i maximizes the covariance between Xw_i and Y, where X and Y represent all the data samples and their respective labels. For the sake of simplicity, we omit the additional steps during the computation of w_i (see Algorithm 1 for more details). Our target is to find W using a single sample $x \in X$, and its respective label y, at a time while maintaining the property of maximizing the covariance across all c-components.

4.4.1 Covariance-free Partial Least Squares

This section defines the proposed method to estimate the projection matrix of PLS incrementally (i.e., using a single sample at a time). We start by describing how to decompose the covariance between dependent and independent variables into an incremental regime, thus enabling the estimation of the first latent-space component incrementally. Then, we introduce how to compute higher-order components incrementally.

Covariance Decomposition. To operate in an incremental scheme and preserve the properties of PLS, our incremental Partial Least Squares approach focuses on ensuring that, as in traditional PLS, the relationship between independent and dependent variables (Equation 2.7) be kept on all the components. To achieve this goal, our method works as follows. First, we center the data to the mean of the training samples X. However, different from traditional methods, in incremental approaches the mean is unknown since we cannot assume that all the data are known a priori [Weng et al., 2003; Zeng and Li, 2014]. To face this problem, we centralize the current data sample using an approximate centralization process [Weng et al., 2003], which consists of estimating an incremental mean using the nth sample. According to Weng et al. [2003], we can compute the incremental mean μ_n w.r.t. the nth data sample as

$$\mu_n = \frac{n-1}{n}\mu_{(n-1)} + \frac{1}{n}x_n. \tag{4.1}$$

Once we have centralized the sample, the next step in our method is to compute the component w_i following Equation 2.7. As we mentioned, X and its respective

Y are unknown or are not in memory in advance, which prevents us from employing Equation 2.7 directly. However, as suggested by Zeng and Li [2014], we employ the following decomposition:

$$X^T Y = \sum_{k=1}^{n-1} x_k^T y_k + x_n^T y_n. \tag{4.2}$$

By replacing $X^T Y$ in Equation 2.7 by Equation 4.2, it is possible to calculate the ith component of PLS considering a single sample at a time. In other words, Equation 4.2 enables to compute w_i incrementally.

Higher-Order Components. To compute the higher-order components (w_i, $i > 1$), we employ a *deflation* process, which consists of subtracting the contribution of the current component on the sample before estimating the next component [Andrew and Tan, 1998; Mackey, 2008]. Following the NIPALS algorithm, the deflation process works as follows

$$t = X w_i, \tag{4.3}$$

$$p = X^T t, \; q = Y^T t, \tag{4.4}$$

$$X = X - t p^T, \; Y = Y - t q^T, \tag{4.5}$$

where t denotes the projected samples onto the current component w_i, and p and q represent the scores of this projection. It should be noted that while t works in an incremental scheme (since we can project one sample at a time), p and q cannot be computed since X and Y are neither known nor are in memory in advance. However, in light of Equation 4.2, we can decompose p and q as

$$p = \sum_{k=1}^{n-1} x_k^T t_k + x_n^T t_n, \; q = \sum_{k=1}^{n-1} y_k^T t_k + y_n^T t_n. \tag{4.6}$$

By embedding Equation 4.6 on the deflation process, we can remove the contribution of the current component and repeat the process to compute a single component w_i (as we argued before). Observe that Equation 4.5 can be computed sample-by-sample working, therefore, in an incremental scheme. At this stage, we obtain all the requirements to find c components incrementally. Since the proposed method does not use the covariance matrix to estimate higher-order components, as proposed by Zeng and Li [2014], we refer to it as *Covariance-free Incremental Partial Least Squares* (CIPLS). Algorithm 6 summarizes the steps of CIPLS.

```
Algorithm 6: CIPLS Algorithm
    Input  : $n$th data sample $x_n$ and its label $y_n$
             Number of components $c$
             Projection matrix $W_{(n-1)} \in \mathbb{R}^{m \times c}$
             Loading matrix $P_{(n-1)} \in \mathbb{R}^{m \times c}$
             Loading matrix $Q_{(n-1)} \in \mathbb{R}^{1 \times c}$
    Output: Updated matrices $W$, $P$ and $Q$

 1  Update $\mu_n$ using Equation 4.1
 2  $\bar{x}_n = x_n - \mu_n$
 3  for $i = 1$ to $c$ do
 4      $w_i = \bar{x}_n^\top y_n + w_{i(n-1)}$, where $w_i \in W$
 5      $t_n = \frac{\bar{x}_n w_i}{\|\bar{x}_n w_i\|}$
 6      $p_i = \bar{x}_n^\top t_n + p_{i(n-1)}$, where $p_i \in P$
 7      $q_i = y_n^\top t_n + q_{i(n-1)}$, where $q_i \in Q$
 8      $\bar{x}_n = \bar{x}_n - t_n p_i^\top$
 9      $y_n = y_n - t_n q_i^\top$
10  end
```

According to Algorithm 6, the proposed method maintains the propriety of capturing the relationship between X and Y for all the components (step 4 in Algorithm 6). In addition, since we compute all components at once for each sample, our method has a time complexity of $O(ncm)$, where n, c and m denote the number of samples, number of components, and dimensionality of the data, respectively.

Chapter 5

Experiments

In this chapter, we present the experiments to validate our hypotheses and assess the effectiveness of the proposed methods. First, we briefly explain the applications, and their respective datasets, used throughout the experiments (Section 5.1). Then, we present the experimental setup (Section 5.2). Finally, we introduce the experiments of our strategies for removing (Section 5.3), inserting (Section 5.4) and combining (Section 5.5) structures from convolutional networks and the experiments of our incremental version of PLS (Section 5.6), in this order.

5.1 Applications and Datasets

Throughout the experiments, we consider the following applications: image classification, activity recognition and face verification. Since most of the works related to our research conduct and report their results on the image classification task, we focus mainly on this application.

For each application, we consider different datasets that vary in sample resolution, number of samples and classes. Table 5.1 summarizes the main features of each dataset.

Activity Recognition. Human activity recognition based on wearable sensor (activity recognition) consists of assigning a category of activity to signals provided by wearable sensors such as accelerometers, gyroscopes and magnetometers.

The first step to perform activity recognition is to generate data samples from raw signals. For this purpose, we follow a typical process that consists of segmenting the signals into windows of the same size, as shown in Figure 5.1. In this process, a window (fixed size) slides over the signal and at each position, the content within of window becomes itself a data sample. The window size is defined in seconds and it determines

Table 5.1. Main features of each dataset. While samples from face verification and image classification are three-channels (RGB) matrices, samples from activity recognition are one-channel matrices.

Application	Dataset	Sample Spatial Resolution (height $\times$ width)	Number of Samples	Number of Classes
Activity Recognition	USCHAD	500×6	$9,824$	12
	WISDM	100×3	$20,846$	7
	UTD-MHAD1	50×6	$3,771$	21
	UTD-MHAD2	50×6	$1,137$	5
Face Verification	LFW	144×192	$6,000$	2
	YTF	160×160	$5,000$	2
Image Classification	CIFAR-10	32×32	$50,000$	10
	ImageNet	224×224 and 32×32	$1,331,167$	$1,000$

the sample height. Following previous works [Song et al., 2017; de Souza et al., 2018], we use windows of five seconds. According to this procedure, the number of sensors and the size of the window define the width and height of the sample, respectively, as illustrates Figure 5.1.

To validate our methods on activity recognition, we consider the following datasets: USCHAD [Zhang and Sawchuk, 2012], WISDM [Lockhart et al., 2011] and UTD-MHAD1-2 [Chen et al., 2015]. These datasets present a large diversity in the activities and cover the most performed daily activities, thus enabling us to examine

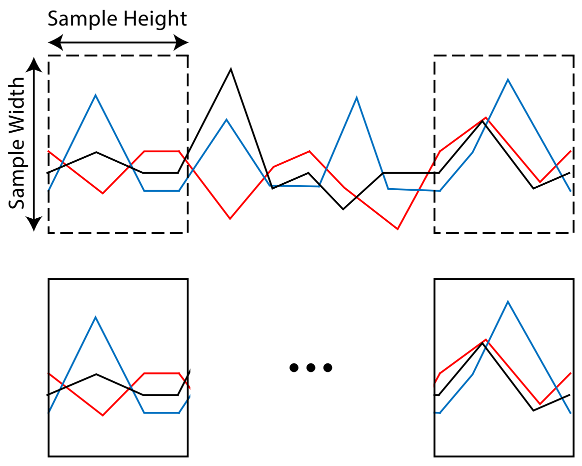


Figure 5.1. Process to generate data samples from raw signals. A window of fixed size (denoted by the dashed box) slides over the signal and, for each slide, the content inside the window yields a data sample (solid box). In this process, the window size defines the sample height while the number of sensors defines the sample width.

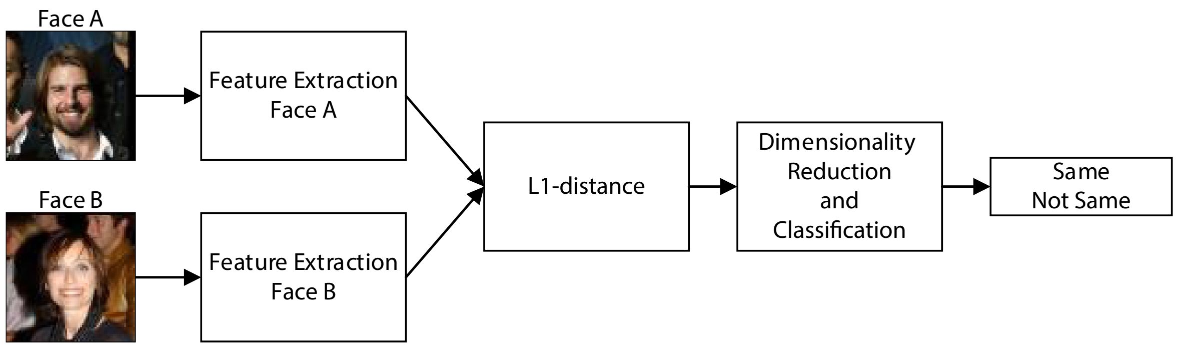

Face A
Feature Extraction Face A
Face B
Feature Extraction Face B
L1-distance
Dimensionality Reduction and Classification
Same
Not Same

Figure 5.2. Face verification pipeline. First, features from faces A and B are extracted and presented to a metric distance (pair-wise operation). Then, the distance metric result feeds a dimensionality reduction method (optional step), which yields a compact representation. Finally, this representation is presented to a classifier that determines if it belongs to the same identity or not.

the effectiveness of the methods on data with high variability.

Face Verification. Given a pair of face images, face verification determines whether this pair belongs to the same person. For this purpose, we use a three-stage pipeline [Vareto et al., 2017a; Kloss et al., 2018] as follows. First, they extract feature vectors of each face using a deep learning model. In this work, we use the feature maps from the last convolutional layer of the VGG16 model, learned on the VGGFaces dataset [Parkhi et al., 2015], as feature vector. Then, we compute the distance between the two feature vectors employing the ℓ_1-distance metric. Finally, we present the result of the distance metric either to a dimensionality reduction method, aiming at yielding a compact representation, or directly to a classifier. Figure 5.2 illustrates these steps.

We conduct our evaluation on two face verification datasets, namely, Labeled Faces in the Wild (LFW) [Huang et al., 2012] and Youtube Faces (YTF) [Wolf et al.,

Figure 5.3. Left. Faces from Labeled Faces in the Wild (LFW). **Right.** Faces from Youtube Faces (YTF).

2011]. These datasets are composed of aligned faces of famous people and present a high diversity in pose, lighting and facial expression. Figure 5.3 illustrates some faces from the LFW and YTF datasets.

Image Classification. This task consists of deciding to which category, given a set of categories, an image belongs. This is done by extracting features from the image and feeding these features to a classifier, which assigns a category to this image.

Following previous works [He et al., 2016; Blalock et al., 2020; Dong and Yang, 2020], we employ two *mandatory* datasets: CIFAR-10 [Krizhevsky et al., 2009] and ImageNet [Deng et al., 2009]. In particular, for the ImageNet dataset, we also use its 32×32 version since it has been demonstrated to be more challenging than the original version (224×224) [Loshchilov and Hutter, 2017], therefore, we can evaluate the methods in a harder scenario. It is worth mentioning that low-resolution datasets have received great attention in computer vision tasks [Hendrycks and Gimpel, 2017; Hendrycks et al., 2019; Chun et al., 2019; Dong and Yang, 2020; Wang et al., 2020; Xie and Yuille, 2020].

Figure 5.4 illustrates images from the CIFAR-10 and ImageNet datasets.

Figure 5.4. Left. Images from ImageNet. **Right.** Images from CIFAR-10.

5.2 Experimental Setup

Throughout the experiments, we adopt the evaluation protocol and the classification metric defined by each dataset, as shown in Table 5.2. On the datasets where the evaluation protocol is cross validation, we report the mean classification accuracy.

Parameter Assessment. To calibrate the parameters of the methods (for example, the number of component c of PLS), we use a validation set with 10% of the training data. In particular, for our LHN applied to activity recognition, we calibrate the parameters using the USCHAD dataset [Zhang and Sawchuk, 2012].

Table 5.2. Standard evaluation protocol and classification metric employed by each dataset. Top-5 accuracy indicates the fraction of images for which the correct category is among the five labels considered most probable by the model.

Application	Dataset	Evaluation Protocol	Metric
Activity Recognition	USCHAD	10-fold cross validation	Accuracy
	WISDM	10-fold cross validation	Accuracy
	UTD-MHAD1	10-fold cross validation	Accuracy
	UTD-MHAD2	10-fold cross validation	Accuracy
Face Verification	LFW	10-fold cross validation	Accuracy
	YTF	10-fold cross validation	Accuracy
Image Classification	CIFAR-10	Hold-out	Accuracy
	ImageNet	Hold-out	Top5-accuracy

Convolutional Networks. For each application we conduct experiments, we use different convolutional networks. On activity recognition, we employ three architectures proposed by ourselves as well as the architecture by Chen and Xue [2015]. On face verification, as suggested by Kloss et al. [2018] and Vareto et al. [2017b], we employ VGG16 learned on the VGGFaces dataset [Parkhi et al., 2015] as features extractor. Finally, on image classification, we use well-known deep convolutional networks, VGG16, ResNet and MobileNets (V1 and V2).

We fine-tune the architectures for 200 and 12 epochs on the CIFAR-10 and ImageNet datasets, respectively, applying horizontal random flip and random crop data augmentation. During this process, we employ SGD with a learning rate starting at 0.01 and decrease it by a factor of 10 after reaching 50% and 75% of the total of epochs. Regarding the regularization schemes, we apply the same configuration as proposed in the original architecture.

Computational Cost. To measure the computational cost of our approaches, we use the number of floating point operations (FLOPs[1]) that is a standard metric to measure the computational cost in convolutional networks [He et al., 2018a, 2019b; Blalock et al., 2020]. Following previous works [Li et al., 2017; Huang et al., 2017], we compute FLOPs in terms of

$$\sum_{i=1}^{L}(W_i \cdot H_i \cdot C_i) \cdot (w_i \cdot h_i \cdot K_i), \tag{5.1}$$

[1]Different from float point operations per seconds (a.k.a FLOPs), in this work, the term FLOPs refers to the number of float point operations only.

where W_i, H_i and C_i denote width, height and the number of channels of the input provided to layer i, respectively. w_i, h_i and K_i denote the filter dimensions (width and height) and the number of filters of the layer i, respectively.

Statistical Tests. To assess the differences in efficacy and efficiency among the compared methods, throughout the experiments we follow the approach by Jain [1990] and perform statistical tests based on a paired t-test using 95% confidence. In particular, on face verification, we use the 90% confidence since this task presents a high variance between the folds of the evaluation protocol [Kloss et al., 2018].

We highlight that the statistical tests were conducted only for activity recognition and face verification. It turns out that training convolutional networks on image classification is computationally expensive, due to the great number of samples (see Table 5.1) and the high computational cost of convolutional networks. For example, the simplest convolutional network used on activity recognition has $24,834$ parameters and $815,616$ FLOPs. On the other hand, VGG16 and ResNet20 applied to image classification[2] have $15,001,418$ and $274,442$ parameters and, $313,463,808$ and $40,813,184$ FLOPs, respectively. Besides, on image classification, we have more than 200 unique models, which means that we would need to retrain/fine-tuning all these models to conduct statistical tests. Therefore, we restrict the statistical evaluation for activity recognition and face verification only.

Machine Setup. All experiments were executed on an Intel Xeon silver 4116 CPU with 200GB RAM and a single NVIDIA GTX 1080.

5.3 Pruning Approaches

In this section, we introduce the proposed strategies for pruning filters (Section 5.3.1) and layers (Section 5.3.3) from convolutional networks.

Following the common practice of many works [Li et al., 2017; Huang et al., 2018; Lin et al., 2020; Tan and Motani, 2020], we examine some aspects and parameters of our method by considering VGG16 only on CIFAR-10. When considering the ImageNet dataset, due to memory constraints, we use 10% of training samples to learn PLS.

Throughout this section, we set the pruning rate of 10% in all experiments and assess the quality of the pruning approaches using two metrics: FLOP reduction and accuracy drop. The former is the percentage of FLOPs removed regarding the original (unpruned) network, where the higher the FLOP reduction the better. The latter is the difference between the accuracy of the original and the pruned network, where the

[2]Values computed using the CIFAR-10 dataset.

lower the drop in accuracy the better and negative values denote improvement of the pruned network upon the original network. Importantly, according to Blalock et al. [2020], when the drop in accuracy is within one percentage point, it can be considered as a negligible loss in accuracy.

5.3.1 Pruning Filters in Convolutional Networks

In this section, we introduce the experiments of the proposed strategy for removing filters from convolutional networks. We first introduce the experiments regarding the parameters of our method. Then, we show the behavior of removing filters iteratively and the importance of representing all filters of the network at once. Next, we compare the proposed criterion for defining filter importance with existing pruning criteria and state-of-the-art pruning approaches, respectively. Finally, we compare our method with state-of-the-art pruning approaches, present qualitative results and final remarks.

5.3.1.1 Influence of the Filter Representation

It is well-known that pooling operations can encode the most important information about large feature maps [Hu et al., 2018; Li et al., 2019b; Veit and Belongie, 2020]. Thus, our first experiment evaluates different pooling operations, referred to as filter representation, to represent feature maps as features. For this purpose, we execute ten pruning iterations using different pooling operations. As illustrated in Figure 5.5, accuracy decreases slower when global max-pooling is employed. On the contrary, by using the max-pooling 2×2 accuracy drops faster, where at the 10th iteration the method drops 26 p.p. compared to the unpruned network. In addition, this representation has the drawback of consuming additional memory compared to the global operations (global max. and average), which reduce the feature map to one dimension.

Besides playing an important role in the pruning performance, the filter representation has an interesting aspect regarding scores assigned by VIP. Note that, to select a filter to be removed means that VIP assigned a low score to it, indicating that it is unimportant to explain the class label. In particular, by modifying the filter representation, we drastically alter the selection of filters to be removed. Figure 5.6 reinforces this idea, where we show the relation between the pruning iteration and the number of removed filters per layer. According to Figure 5.6, the max-pooling representation eliminates a larger number of filters from layers 3 to 7, while the global average pooling has a similar distribution of the VIP scores, since it removes filters from all layers uniformly (except for layers 3, 10 and 11). On the other hand, the global max-pooling representation removes a larger number of filters from layers 3 to 9 and keeps the filters

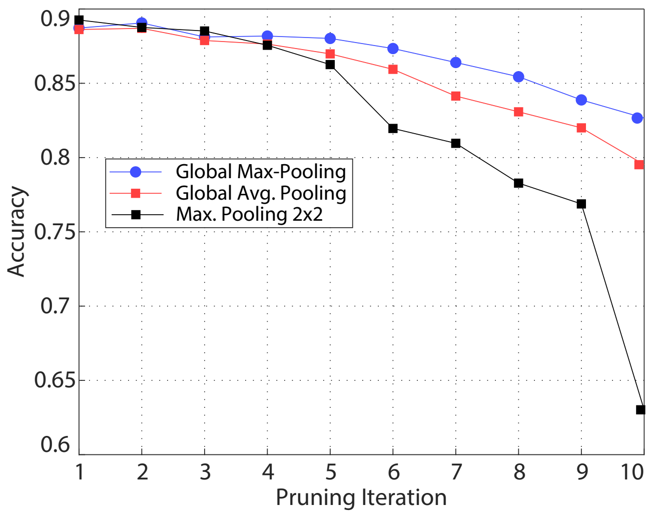


Figure 5.5. Accuracy obtained by pruning VGG16 on the CIFAR-10 dataset (validation set) using different filter representations.

from layers 1 and 2. Finally, VIP assigns high scores for the filters from the layers 10 and 11. Based on the results, we used the global max-pooling as filter representation in the remaining experiments.

According to Li et al. [2017], filters from the first layer should not be pruned since their removal degrades network performance significantly. Our method is able to identify this because either it does not remove or it removes few filters from this layer, as shown in Figure 5.6, which indicates the suitability of PLS to identify the relevant filters. It is important to mention that in the work by Li et al. [2017], the conclusion that the filters from the first layer are important was done by a human analyzing the

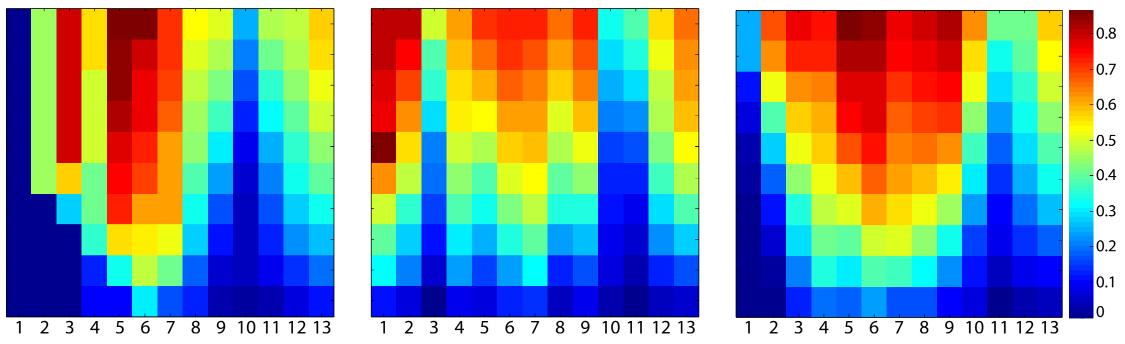


Figure 5.6. Heat map of the relation between the number of filters removed, by layer, and the iteration of the proposed method using different filters representations. **Left.** Max-pooling 2×2. **Middle.** Global Average pooling. **Right.** Global Max pooling. Warmer regions indicate that more filters were removed. In these figures, the x-axis represents the VGG index while the y-axis represents the pruning iteration.

accuracy drop when removing these filters. However, this is performed automatically in our work.

Another interesting aspect concerning VIP distributions is that the linear projection of PLS explains well the relationship between filters and the class label. This is because if there were a strong non-linear relationship, VIP would always assign a low score to some filters since they would not be explained by PLS. Thereby, filters from specific layers would always be removed. As shown in Figure 5.6, however, this behavior did not occur, since filters from different layers were removed.

5.3.1.2 Iterative Pruning vs. Single Pruning

In this experiment, we show that it is more appropriate to execute our method iteratively, as illustrated in Figure 4.2, with a low pruning ratio (i.e., 10%) instead of using a single pruning iteration with a high pruning ratio. In other words, if we intend to remove i.e. 40% of filters, it is better to execute some iterations of our method with a low pruning ratio instead of setting a pruning ratio of 40% and execute only a single iteration. To this end, we first execute five iterations of the proposed method with a pruning ratio of 10%. Then, after each iteration, we compute the percentage of removed filters, p_i. Finally, we use each p_i as the pruning ratio to execute a single iteration of the method.

According to the results shown in Table 5.3, performing our method iteratively with a low pruning ratio is more effective than using it with a large pruning ratio, which led to a higher drop in accuracy. For instance, by executing five iterations of the method with a pruning ratio of 10%, we are able to remove 40% of filters while improving the network accuracy (indicated by negative values in Table 5.3). On the other hand, by applying a single iteration with a pruning ratio of 40%, the accuracy

Table 5.3. Drop in accuracy when executing our method with few iterations and a low pruning ratio (Iterative Pruning), and when executing a single iteration with a high pruning ratio (Single Pruning). Results on CIFAR-10 (test set). The first column is the percentage of removed filters in iterations 1, 3, 5, and 10, respectively. The arrows indicate which direction is better.

Percentage of Removed Filters (%)	Iterative Pruning Accuracy Drop$\downarrow$	Single Pruning Accuracy Drop$\downarrow$
10	-0.89 (it=1)	-0.89
27	-1.08 (it=3)	-0.03
40	-0.69 (it=5)	1.76
65	1.56 (it=10)	20.21

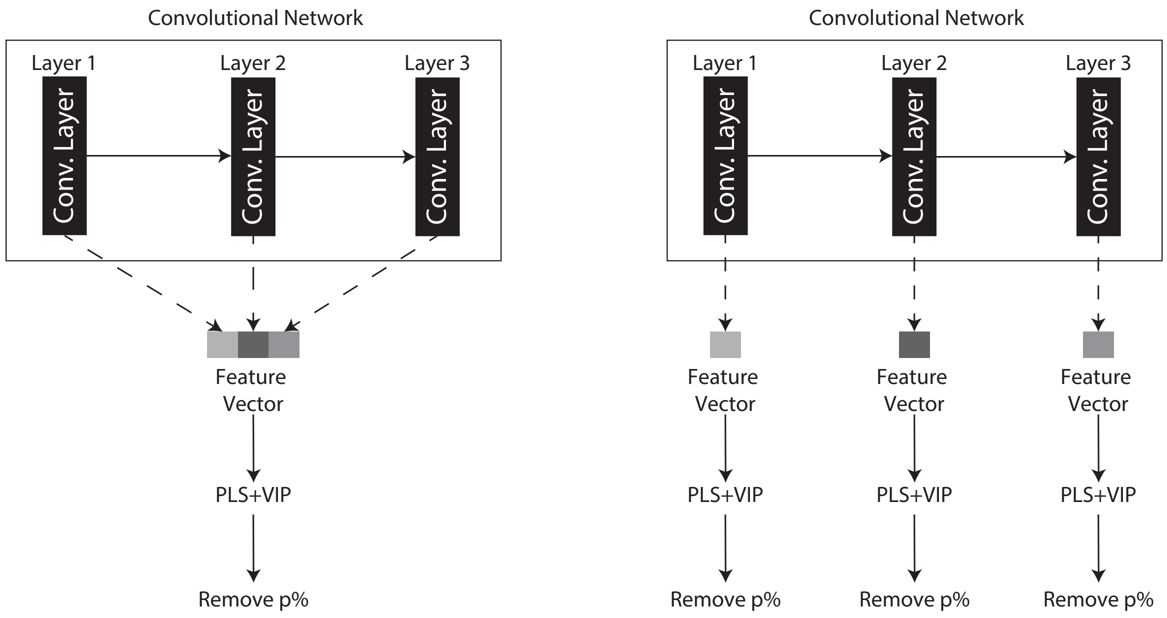


Figure 5.7. Left. Single projection scheme. In this strategy, a single PLS model is learned considering all filters that compose the network at once. **Right.** Multiple projections scheme. In this strategy, one PLS model is learned considering filters layer-by-layer (i.e., one PLS per layer).

decreased 1.76 p.p..

5.3.1.3 Multiple Projections vs. Single Projection

This experiment shows the performance of our method when using the filters layer-by-layer and all filters at once to learn PLS. While the former has a PLS model associated with each layer, the latter has only one PLS model, as shows Figure 5.7.

Table 5.4 (last rows) shows the results of our pruning approach when using single and multiple projections, called PLS(Single)+VIP and PLS(Multi)+VIP, respectively. On the CIFAR-10 dataset, PLS(Multi)+VIP and PLS(Single)+VIP achieved similar performance, where PLS(Multi)+VIP obtained a drop in accuracy 0.08 p.p. better than PLS(Single)+VIP. On both versions of ImageNet, however, PLS(Multi)+VIP attained a drop in accuracy worst than PLS(Single)+VIP.

The results above indicate that learning PLS projection considering all filters, at once, is more suitable for pruning filters. The reason for these results is that filters coming from different layers degrade network performance in different ways [Li et al., 2017]. Therefore, removing $p\%$ of filters of each layer (as is done in PLS(Multi)+VIP) is more critical than removing $p\%$ of all filters[3]. This remark reinforces the usage of

[3]Remove $p\%$ considering all filters at once is different from removing $p\%$ of filters for each layer.

Table 5.4. Drop in accuracy using different criteria for determining filter importance. PLS(Multi)+VIP indicates our method projecting the filters layer-by-layer. PLS(Single)+VIP indicates our method projecting all the filters that compose the network at once. Negative values denote improvement regarding the original (unpruned) network. The best results are in bold. The arrows indicate which direction is better.

Filter Importance Criterion	CIFAR-10 Acc. Drop↓	ImageNet 32×32 Acc. Drop↓	ImageNet 224×224 Acc. Drop↓
ℓ_1-norm	−0.69	6.22	**-0.62**
infFS [Roffo et al., 2015]	−0.69	6.31	−0.50
ilFS [Roffo et al., 2017]	0.65	6.04	−0.36
infFS$_U$ [Roffo et al., 2020]	0.48	6.30	−0.33
KL [Luo and Wu, 2020]	−0.59	6.37	−0.41
HRank [Lin et al., 2020]	−0.84	6.70	−0.47
ABS [Tan and Motani, 2020]	−0.62	6.58	−0.42
PLS(Multi)+VIP	**-0.97**	6.42	−0.50
PLS(Single)+VIP	−0.89	**5.81**	−0.58

PLS(Single)+VIP. Thus, we use PLS(Single)+VIP (hereafter referred to as PLS+VIP) in the remaining experiments.

5.3.1.4 Comparison with other Pruning Criteria

In this experiment, we compare the proposed criterion (PLS+VIP) for assigning filter importance with other criteria and state-of-the-art feature selection techniques. To this end, we use one iteration of pruning and follow the process suggested by Yu et al. [2018], which consists of setting the same pruning ratio (10%) and modifying only the criterion for selecting the filters to be removed.

Table 5.4 shows the results obtained by different pruning criteria on the CIFAR-10 and ImageNet datasets. Compared to state-of-the-art pruning criteria [Luo and Wu, 2020; Lin et al., 2020; Tan and Motani, 2020], PLS+VIP obtained the lowest drop in accuracy. Compared to state-of-the-art feature selection [Roffo et al., 2015, 2017, 2020], PLS+VIP also achieved superior results. In particular, only the ℓ_1-norm criterion on ImageNet 224×224 outperformed our criterion. Observe that, even when considering the multiple projections strategy, PLS(Multi)+VIP, we also outperformed most criteria. The reason for these results is that PLS preserves filters with high relationship with the class label, which are the most important to the classification ability of the network.

Interestingly, in many settings (criteria $\times$ dataset), the drop in accuracy is negative, which means that the pruned network obtained an improvement in accuracy compared to the original (unpruned) network. Such results are expected since pruning has been demonstrated as a powerful tool for regularization, which might obtain superior generalization than over-parameterized networks [Huang et al., 2016; Li et al., 2019a; Fan et al., 2020; Bartoldson et al., 2020].

Finally, it is important to note that, when evaluated on the original (224×224) ImageNet dataset, the drop in accuracy of the methods is small. In contrast, on the 32×32 version, the drop is notable. This behavior supports the employment of this version, where we can evaluate the criteria in a more difficult scenario. We emphasize that the single difference between these versions of ImageNet is the image size.

5.3.1.5 Comparison with Existing Pruning Approaches

This experiment compares the proposed method with state-of-the-art pruning approaches. For this purpose, we report the results using different pruning iterations. We highlight that the results of previous methods were taken from their original papers. Table 5.5 and 5.6 summarize the results.

On the CIFAR-10 dataset, regardless of the architecture, our method obtained the lowest drop in accuracy. In terms of FLOP reduction, only the recent approach by Lin et al. [2020] achieved superior results. On the VGG16 architecture, the proposed approach and the methods by Lin et al. [2020] and, Liu et al. [2017] achieved a FLOP reduction above 90%. Such achievements, however, are not surprising since previous works have argued that VGG is a redundant architecture [Luo et al., 2019], mainly compared to ResNet architectures. Table 5.5 reinforces this remark, where the highest FLOP reduction in ResNet is around 1.28× less than VGG.

Besides removing more FLOPs, our method is also computationally more efficient in terms of the number of fine-tuning necessary. For instance, the methods by Hu et al. [2016] and Huang et al. [2018] require 16 stages of fine-tuning to prune VGG16. On the other hand, our method achieves competitive results with only around five fine-tuning stages. Similar to our method, the approaches by Yu et al. [2018] and He et al. [2018b] demand few fine-tuning stages, however, we achieve a better trade-off between FLOP reduction and accuracy drop, as shown in Table 5.5.

On the ImageNet dataset, by pruning VGG16, with only three iterations of the proposed method, we were able to achieve 1.75× more FLOPs reduction than most pruning methods on similar accuracy drop. In particular, only the approach by Luo et al. [2019] obtained a higher FLOP reduction than our method. On the ResNet50 architecture, our method achieves competitive results. However, compared to the most recent pruning approaches, we achieved a lower FLOP reduction. It turns out that,

Table 5.5. Comparison of existing pruning methods. Negative values denote improvement regarding the original network. The best FLOP reduction and accuracy drop are shown in bold. The arrows indicate which direction is better.

	Method	FLOP Reduction↑	Accuracy Drop↓
VGG16 on CIFAR-10	Hu et al. [2016]	28.29	−0.66
	Li et al. [2017]	34.00	−0.10
	He et al. [2019b]	35.90	0.34
	Guo et al. [2020a]	54.89	−0.57
	Huang et al. [2018]	64.70	1.90
	Lin et al. [2020]	92.00	2.73
	Liu et al. [2017]	**95.70**	3.35
	Ours (it=1)	23.13	**-0.89**
	Ours (it=5)	67.25	−0.63
	Ours (it=10)	90.66	1.50
ResNet56 on CIFAR-10	Yu et al. [2018]	43.61	0.03
	He et al. [2018b]	50.00	0.90
	He et al. [2019b]	52.60	0.10
	He et al. [2018a]	52.60	1.33
	He et al. [2020]	52.90	0.25
	Chin et al. [2020]	53.00	0.20
	Guo et al. [2020a]	54.89	−0.55
	Lin et al. [2020]	**74.10**	2.54
	Ours(it=1)	7.09	−0.60
	Ours(it=5)	35.23	**-0.90**
	Ours(it=11)	66.54	−0.38
ResNet110 on CIFAR-10	He et al. [2018a]	40.80	0.30
	Yu et al. [2018]	43.78	0.18
	He et al. [2019b]	52.30	−0.17
	He et al. [2020]	60.30	−0.11
	Lin et al. [2020]	**68.70**	0.85
	Ours(it=1)	6.85	−0.59
	Ours(it=5)	33.16	**-1.51**
	Ours(it=10)	60.17	−0.93

when pruning this architecture, our approach removed few filters from 3×3 convolutions layers within the bottleneck building block, which are the ones with the higher number

Table 5.6. Comparison of existing pruning methods. Negative values denote improvement regarding the original network. The arrows indicate which direction is better.

	Method	FLOP Reduction↑	Accuracy Drop↓
VGG16 on ImageNet(224×224)	Hu et al. [2016]	19.69	0.84
	He et al. [2017]	20.00	1.7
	Wang et al. [2018b]	20.00	2.00
	He et al. [2018b]	20.00	1.40
	Luo et al. [2019]	**69.81**	1.88
	Ours(it=1)	11.02	**−0.58**
	Ours(it=3)	35.73	1.75
	Ours(it=5)	58.51	3.69
ResNet50 on ImageNet(224×224)	Hu et al. [2016]	19.69	0.84
	He et al. [2017]	20.00	1.7
	Wang et al. [2018b]	20.00	2.00
	He et al. [2018b]	20.00	1.40
	Li et al. [2019a]	50.00	0.36
	He et al. [2019b]	53.50	0.55
	Guo et al. [2020a]	55.71	−0.28
	He et al. [2020]	60.80	0.83
	Luo and Wu [2020]	72.86	1.41
	Lin et al. [2020]	**76.03**	3.29
	Ours(it=1)	6.13	**-1.92**
	Ours(it=5)	27.45	−0.31
	Ours(it=10)	44.50	1.01

of FLOPs, thus we achieve lower FLOP reduction.

In summary, our strategy for removing filters obtained one of the best trade-offs between FLOP reduction and accuracy drop. To better visualize such a trade-off, we plot the top 5 best pruning methods (according to FLOP reduction) and the proposed approach in Figure 5.8. From Figure 5.8 (left), it is possible to note that our method always provides a better solution (i.e., it is a non-dominated solution) considering accuracy drop or FLOP reduction. On the other hand, In Figure 5.8 (right), some strategies outperformed our method in both accuracy drop and FLOP reduction.

Based on the aforementioned discussion, we have shown that the proposed method achieves a superior reduction in FLOPs. This is an effect of the layers where it removes the filters. According to Figure 5.9 (left), the layers 2, 4, 6, 7, 9 and 10 of VGG16 have

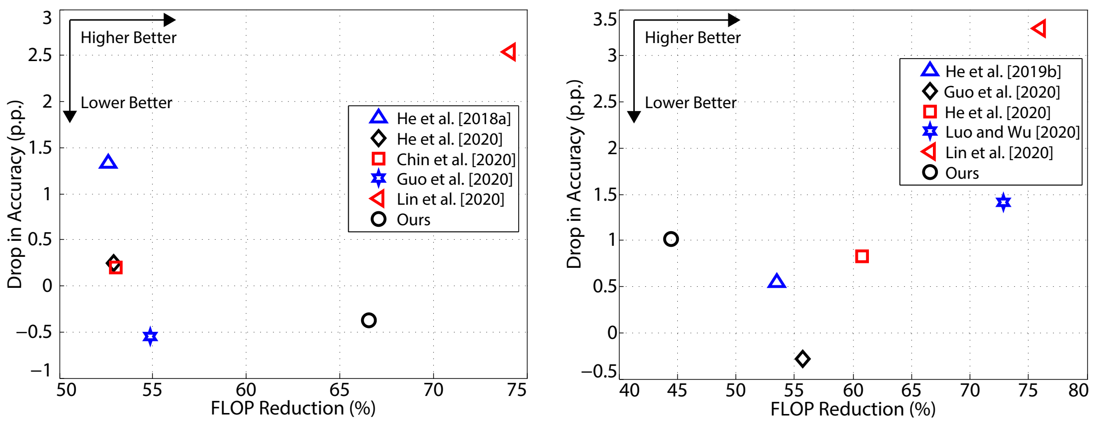


Figure 5.8. Comparison of existing pruning methods. **Left.** Results on the CIFAR-10 dataset. On this dataset, our method always provides a better solution (i.e., it is a non-dominated solution) considering one of the performance metrics: accuracy drop (y-axis) or FLOP reduction (x-axis). **Right.** Results on the ImageNet (224 × 224) dataset. In both figures, negative values in the y-axis denote improvement regarding the original, unpruned, network. The arrows indicate which direction is better.

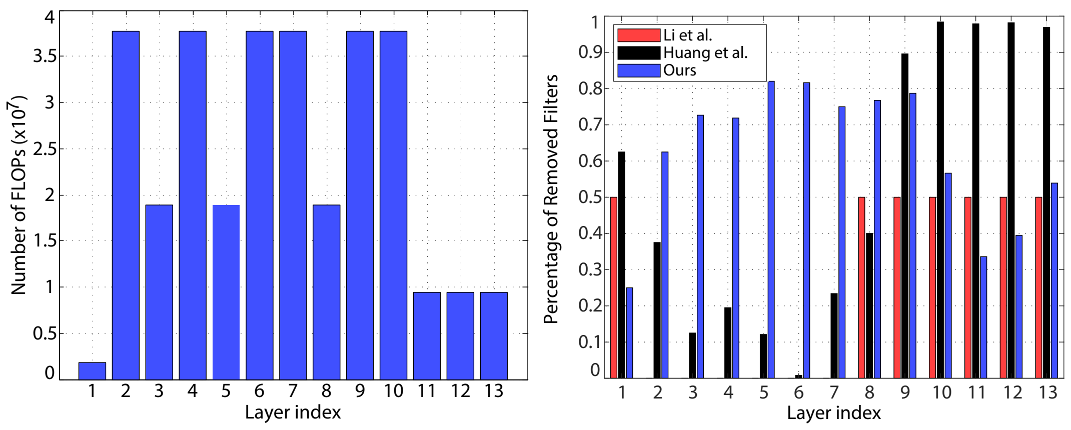


Figure 5.9. **Left.** Number of float point operations (FLOPs) per layer of the VGG16 network. **Right.** Percentage of removed filters in each layer using different pruning methods. Values computed from the VGG16 network on the CIFAR-10 dataset.

the higher number of FLOPs. In general, the existing methods fail to eliminate filters from these layers. For instance, the methods proposed by Li et al. [2017] and Huang et al. [2018] remove a large number of filters from the layers 9 to 13 (Figure 5.9 (right)), but they remove a small number of filters from other layers. On the contrary, our method eliminates a large number of filters from all layers, as shown in Figure 5.9 (right). In particular, we eliminate more than 50% of filters from layers 2 to 10, which are the ones with the large number of FLOPs, and more than 25% from the other layers. Hence, we are able to achieve a higher FLOPs reduction than existing state-of-the-art methods, which are biased in eliminating filters of particular layers.

5.3.1.6 Pruning Lightweight Networks

In this experiment, we assess the performance of removing filters from lightweight networks. Table 5.7 shows the results when using one, three and five iterations of pruning on MobileNetV1 and MobileNetV2. On the CIFAR-10 dataset, we were able to remove up to 60% FLOPs with a negligible drop in accuracy, i.e., it is within one percentage point. To achieve a similar FLOP reduction on ImageNet, the drop in accuracy was substantially higher. In particular, on this dataset, MobileNetV1 had a significant drop in accuracy even when considering few (i.e., three) iterations of pruning, thus, suggesting that it is a bit sensitive to pruning on ImageNet.

Table 5.7. Results when pruning filters from lightweights architectures. Negative values denote improvement regarding the original network. The best FLOP reduction and accuracy drop, for each dataset, are shown in bold. The arrows indicate which direction is better.

	Iteration	FLOP Reduction↑	Accuracy Drop↓
MobileV1 on CIFAR-10	1	17.73	−0.22
	3	42.89	0.04
	5	**60.99**	0.12
MobileV2 on CIFAR-10	1	13.24	**-0.37**
	3	24.23	−0.18
	5	48.20	−0.07
MobileV1 on ImageNet (224x224)	1	14.16	0.44
	3	37.81	2.67
	5	**59.71**	5.61
MobileV2 on ImageNet (224x224)	1	15.15	**-0.51**
	3	34.77	0.70
	5	50.52	2.61

5.3.1.7 Time Issues

In this experiment, we demonstrate the improvement in prediction time provided by our strategy for removing filters. For this purpose, we compare the average prediction time of the original VGG16 and its prediction time after running one, five and ten iterations of pruning. Figure 5.10 shows the average time considering 30 executions for predicting a single sample from the CIFAR-10 dataset.

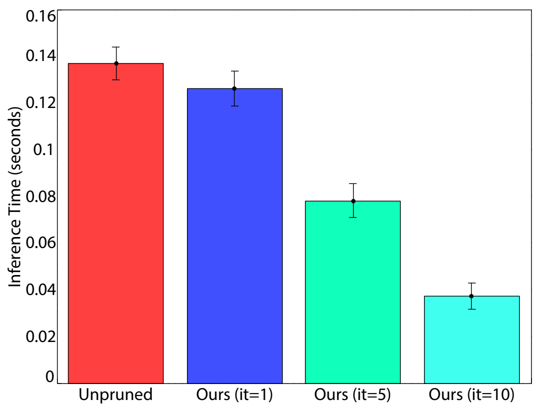


Figure 5.10. Average prediction time (lower is better) of the original network and it after running one, five and ten pruning iterations. Black bars denote the confidence interval. Values are computed using VGG16 architecture on CIFAR-10.

According to Figure 5.10, after one iteration of pruning, the improvement in prediction time is marginal, but statistically different. On the other hand, with five and ten iterations of pruning the improvement is visually higher. In particular, on the paired t-test, all the times were statistically different, indicating that for different pruning iterations, the improvement in prediction time, de-facto, happens.

5.3.2 Generalization Ability

Our next experiment evaluates the generalization ability of pruned models when transferred to other datasets. For this purpose, we prune the ResNet56 architecture on ImageNet 32×32 employing different pruning iterations. Then, we fine-tune (i.e., adjust the weights) and evaluate the pruned architectures on the CIFAR-10 dataset. Table 5.8 summarizes the results.

According to Table 5.8, when applied to transfer learning, the pruned models obtain similar accuracy to the original model (see the last column in the table). Specifically, the difference between the accuracy of the original and the pruned models is less

Table 5.8. Generalization ability (transfer learning) of pruned models. The arrows indicate which direction is better.

Architecture	Accuracy on Transfer Learning$\uparrow$	Difference to (unpruned) ResNet56$\downarrow$
ResNet56	95.44	–
ResNet56 + Pruning (it=1)	95.20	0.23
ResNet56 + Pruning (it=3)	95.12	0.31
ResNet56 + Pruning (it=5)	94.90	0.53

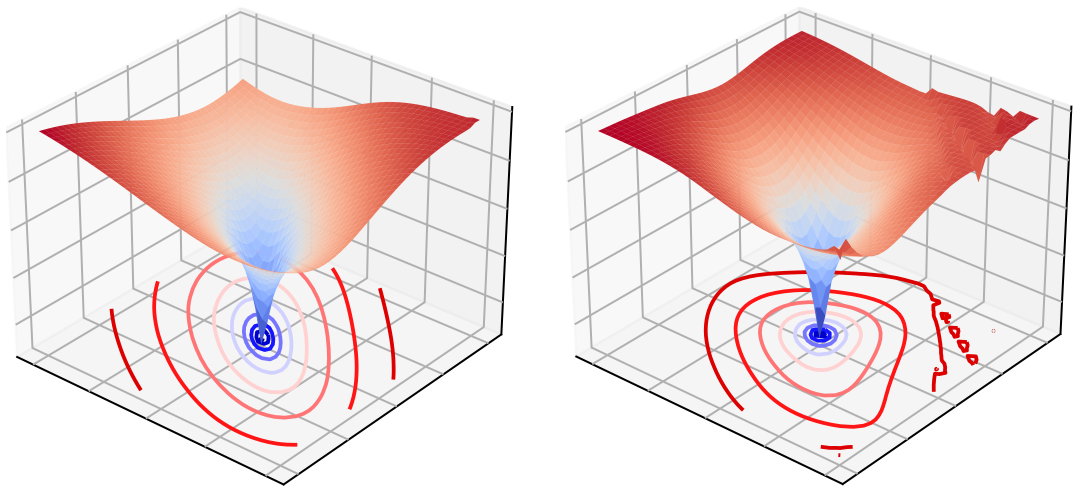


Figure 5.11. Loss Landscape of ResNet56 (left) and its pruned version (right). Both models exhibit similar landscapes, which indicates that they tend to achieve the same generalization ability.

than one percentage point, which means that it is negligible. In particular, these results are consistent even when we consider more iterations of pruning [Blalock et al., 2020]. Additionally, they demonstrate that pruning preserves the generalization power of convolutional networks. To reinforce this claim, we plot the loss landscape of ResNet56 and its pruned version in Figure 5.11. According to previous works [Li et al., 2018; Guo et al., 2020b], the loss landscape is capable of showing the generalization power of convolutional networks, for which the flatter landscape the better generalization. Based on this observation, Figure 5.11 reinforces that pruning preserves the generalization ability of convolutional networks, as the loss landscape of both ResNet56 and its pruned version exhibits similar dynamics.

5.3.2.1 Qualitative Results

Our last experiment shows that the regions in the image which are important to predict the class label are preserved after pruning a network with our method.

Figure 5.12 shows the attention maps of the VGG16 network on images from the ImageNet dataset. It is possible to note that our method preserves the important regions (warmer regions), which are the ones where the object is located. In addition, sometimes, our method locates class-discriminative regions better than the original network, e.g., Figure 5.12 (a)-(c). This is an effect of PLS+VIP, which focuses on keeping filters with high relationship with the class label.

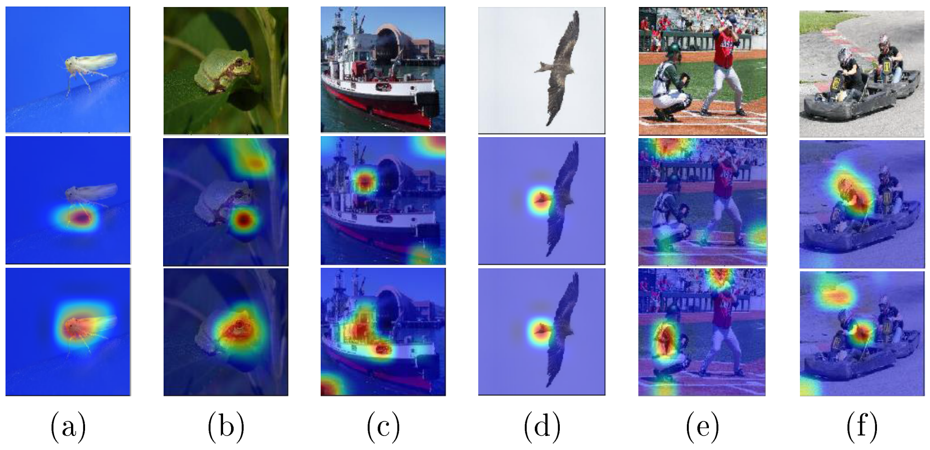


Figure 5.12. Attention maps of the VGG16 network. From top to down. Input images; Attention maps from the original network; Attention maps from the pruned network.

5.3.2.2 Final Remarks

We demonstrate that is possible to remove unimportant, or least important, filters by estimating their importance using PLS. These results confirm our hypothesis that the relationship between filters and the class label on a low-dimensional space (PLS criterion) can be employed to identify potential filters to be removed.

Compared to existing criteria for determining filter importance as well as state-of-the-art feature selection techniques, PLS achieves the lowest drop in accuracy. Compared to state-of-the-art pruning approaches, our strategy for removing filters achieves one of the best trade-offs between FLOP reduction and accuracy drop. In particular, on some architectures, we obtain a large FLOP reduction while improving network accuracy.

Limitations. One concern about the proposed method to remove filters is the large memory consumption when applied to large datasets. This limitation takes place because we generate a matrix describing the filter (features) representation for each sample of the dataset, and this matrix needs to be in memory in advance. Specifically, the dimensions of this matrix are #features (filters) $\times$ # samples. Intuitively, on large datasets, its second dimension becomes large, thus requiring more memory consumption.

5.3.3 Pruning Layers in Convolutional Networks

In this section, we introduce the experiments of the proposed strategy for removing layers from convolutional networks. We first introduce two ways of removing layers:

iterative or single pruning. Then, we compare the proposed criterion for defining layer importance with existing pruning criteria and state-of-the-art pruning approaches, respectively. Next, we assess the effectiveness of removing layers from lightweights architectures. Afterward, we present results when layers and filters are removed from the network. Finally, we demonstrate improvements in inference time and final remarks.

Throughout this section, the terms layers and modules are used interchangeably.

5.3.3.1 Iterative Pruning vs. Single Pruning

Similar to remove filters, when removing layers, we can consider two strategies. (i) Iterative pruning: remove a low percentage of layers iteratively. (ii) Single pruning: remove a high percentage of layers at once.

In this experiment, we demonstrate the behavior of pruning on these two strategies. For this purpose, as in Section 5.3.1.2, we perform some iterations of the iterative pruning, where for each iteration we remove 10% of the modules, and measure the percentage of structures (modules) removed per iteration. Then, we use these percentages to set a single iteration of pruning. Table 5.9 shows the results. According to this table, iterative pruning provided better results than single pruning, as it removed a large percentage of modules with the lowest drop in accuracy. Specifically, iterative pruning was able to remove up to 55% of the modules with improvement in accuracy (-0.65 p.p.) while single pruning decreased accuracy by 0.71 p.p.. Such results are expected since iterative pruning performs more fine-tuning stages. For example, to prune 55% of the modules, iterative pruning requires ten stages of fine-tuning while single pruning requires only one iteration of fine-tuning. In particular, single pruning always requires only one stage of fine-tuning regardless of the percentage of modules removed. Interestingly, the drop in accuracy of the single pruning strategy is less than one percentage point. Thereby, single pruning can be more appropriate to large datasets and deeper

Table 5.9. Drop in accuracy (in percentage points) when executing our method with few iterations and a low pruning ratio (Iterative Pruning), and when executing a single iteration with a high pruning ratio (Single Pruning). Results on CIFAR-10 (test set). The arrows indicate which direction is better.

Percentage of Removed Modules (%)	Iterative Pruning Accuracy Drop↓	Single Pruning Accuracy Drop↓
10	−0.84 (it=1)	−0.84
22	−0.93 (it=3)	−0.34
37	−0.76 (it=5)	0.03
55	−0.65 (it=10)	0.71

networks, where fine-tuning is time-consuming. We refer the reader to Appendix A for details of the time for fine-tuning.

5.3.3.2 Comparison with other Pruning Criteria

In this experiment, we compare our criterion (PLS+VIP) for assigning layer importance with other criteria and state-of-the-art feature selection techniques, which can be employed to define layer importance as well. To this end, we follow the same process when comparing PLS+VIP with other criteria for pruning filters, which consists of using a single pruning iteration and modifying only the criterion for assigning importance.

Table 5.10 shows the results obtained by different pruning criteria on the CIFAR-10 and ImageNet datasets. According to the results, on CIFAR-10, our criterion for determining layer importance achieved the best drop in accuracy. On the low-resolution version of ImageNet, our criterion outperformed only the criterion by Lin et al. [2020]. On the 224 × 224 version of ImageNet, all criteria had similar performance with improvement in accuracy, but our criterion obtained the lowest increase in accuracy. In general, the criterion by Luo and Wu [2020] achieved one of the best drops in accuracy across the datasets.

Even though our criterion underperforms some criteria, it is important to emphasize that PLS+VIP is computationally more attractive. For example, the feature selection techniques (infFS, ilFS, infsFSU) require an adjacency matrix representing all pairs of features, consuming substantial computational resources. The rank approach by Lin et al. [2020] (HRank) is time-consuming since the feature map rank is estimated using the SVD technique. Finally, KL-divergence (KL) is one of the most computa-

Table 5.10. Drop in accuracy using different criteria for determining layer importance. Negative values denote improvement regarding the original (unpruned) network. The best results are in bold. The arrows indicate which direction is better.

Layer Importance	CIFAR-10	ImageNet 32 × 32	ImageNet224 × 224
Criterion	Acc. Drop↓	Acc. Drop↓	Acc. Drop↓
infFS [Roffo et al., 2015]	−0.68	1.50	−2.03
ilFS [Roffo et al., 2017]	−0.46	1.12	**−2.11**
infFS$_U$ [Roffo et al., 2020]	−0.50	2.03	−2.03
KL [Luo and Wu, 2020]	−0.32	1.00	−2.06
HRank [Lin et al., 2020]	−0.73	2.35	−2.03
ABS [Tan and Motani, 2020]	−0.54	**0.96**	**−2.11**
PLS+VIP	**−0.84**	2.25	−1.92

tionally expensive criteria since it requires a forward prediction for each structure of the network.

5.3.3.3 Comparison with Existing Pruning Approaches

In this experiment, we compare the proposed strategy for pruning layers with state-of-the-art pruning approaches that focus on removing the same structure, Table 5.11. We discuss the results employing the single prune iteration since it is more suitable for deeper networks and large datasets, as we discussed before. In addition, we restrict our discussion to ResNet110 on CIFAR-10 and ResNet50 on ImageNet (224×224) since these settings are the most reported in the context of pruning layers.

According to Table 5.11, on both datasets, our method is the one with the best trade-off between drop in accuracy and FLOP reduction. On CIFAR-10, our method achieved a surprisingly 74.75 FLOP reduction with an accuracy drop within one percentage point. On ImageNet, we achieve around 3× more FLOP reduction compared to Veit and Belongie [2020] and, Huang and Wang [2018]. It is worth mentioning that the approaches by Veit and Belongie [2020] and, Wu et al. [2018] are dynamics strategies, which means that the FLOP reduction is conditioned to the input presented to the network; thereby, they can be prohibitive to fixed-resources environments. For example, consider a scenario where it is possible to execute only 35% of the FLOPs of a network, which means that we need to reduce 65% of the FLOPs to run it on such a scenario. To the images where the pruned network surpasses this value, the system can present fails as well as compromising other components. On the other hand, our

Table 5.11. Comparison of existing pruning methods that focus on removing layers. Negative values denote improvement regarding the original network. The symbol * indicates mean FLOP reduction. The best FLOP reduction and accuracy drop are shown in bold. The arrows indicate which direction is better.

	Method	FLOP Reduction↑	Accuracy Drop↓
ResNet110 on CIFAR-10	Veit and Belongie [2020]	18.00*	0.62
	Huang and Wang [2018]	50.47	0.26
	Wu et al. [2018]	65.00*	**-0.39**
	Ours	**74.75**	0.82
ResNet50 on ImageNet(224×224)	Wang et al. [2018b]	12.00*	0.00
	Veit and Belongie [2020]	15.00*	**-0.20**
	Huang and Wang [2018]	31.00	0.95
	Ours	**45.28**	0.67

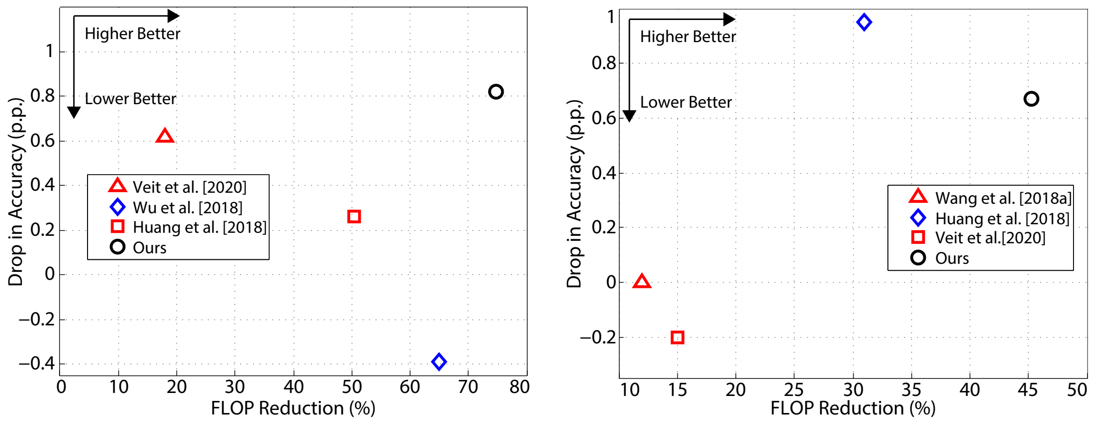


Figure 5.13. Comparison of existing pruning methods. **Left.** Results on the CIFAR-10 dataset. **Right.** Results on the ImageNet (224×224) dataset. On both datasets, our method always provides a better solution (i.e., it is a non-dominated solution) considering one of the performance metrics: accuracy drop (y-axis) or FLOP reduction (x-axis). Negative values in the y-axis denote improvement regarding the original, unpruned, network. The arrows indicate which direction is better.

strategy does not suffer from this problem since the FLOP reduction is the same for all images.

To better visualize the trade-offs between accuracy and FLOP reduction, we plot the results of Table 5.11 in Figure 5.13. From this figure, it is possible to note that our method always provides a better solution (i.e., it is a non-dominated solution) considering accuracy drop or FLOP reduction.

5.3.3.4 Pruning Lightweight Networks

In this experiment, we assess the performance of removing modules from lightweight networks, Table 5.12. Different from previous experiments on lightweight architectures, here, we consider only MobileNetV2. It turns out that MobileNetV1 is a plain network and to prune layers the architecture needs to be residual, as we explain in Section 3.2.2.

To reduce the computational cost, we discuss the results considering only the single pruning strategy with different pruning ratio. Specifically, due to implementation details, MobileNetV2 has only five possible modules to be removed. Thus, we set the prune ratio such that it removes 1, 2, 3, and 4 modules. It is worth mentioning that MobileNetV2 on ImageNet has two downsampling layers at the beginning of architecture, implying in different FLOP reduction when compared to architecture applied to CIFAR-10.

According to Table 5.12, on both datasets, the proposed approach is able to

Table 5.12. Results when pruning layers from lightweights architectures. Negative values denote improvement regarding the original network. The best FLOP reduction and accuracy drop are shown in bold. The arrows indicate which direction is better.

	Pruning Ratio	FLOP Reduction↑	Accuracy Drop↓
MobileV2 on CIFAR-10	0.06	4.60	−0.14
	0.13	9.20	0.16
	0.19	16.31	**-0.30**
	0.26	**26.60**	0.04
MobileV2 on ImageNet (224×224)	0.06	6.84	**-0.83**
	0.13	16.74	−0.21
	0.19	21.16	−0.08
	0.26	**25.66**	0.31

remove more than 26% of the modules with a negligible drop in accuracy. More importantly, these results suggest that lightweight networks are not sensitive to layer removal, thus enabling further improve their performance in terms of latency.

It is important to mention that when pruning filters from MobileNetV2, the FLOP reduction is higher (up to 48.20%, see Table 5.7). It turns out that we can remove, at most, five modules from MobileNetV2 while pruning filters can eliminate filters from all modules, thus obtaining a higher FLOP reduction. Despite this, we shall see that pruned networks obtained from the removal of layers attain considerably better prediction time than those from the removal of filters.

5.3.3.5 Pruning Multiple Structures

Pruning filters and layers are orthogonal strategies and, therefore, they could benefit each other. In this experiment, we assess the effectiveness of removing both structures. For this purpose, we follow a two-step mechanism: we first remove layers and, then, we remove filters. In practice, we use ResNet56 with 55% of its modules removed (the network of Table 5.9, last row) as input to our pruning filter method introduced in Section 4.1.1. It is worth mentioning we could remove filters first, however, the removal of layers provides a network with lower computational cost and latency, which reduces memory requirements and speeds-up the fine-tuning stage. Additionally, our assessment is restricted to ResNet110 on the CIFAR-10 dataset, as it is the most reported setting for both layer and filter pruning.

Table 5.13 summarizes the results. Compared to the top-performance strategies that remove either filters or layers, our strategy for pruning both structures attains

Table 5.13. Comparison of our strategy for removing multiple structures with methods that remove filters or layers, but not both. The best FLOP reduction and accuracy drop are shown in bold. Negative values denote improvement regarding the original network. The arrows indicate which direction is better.

	Method	FLOP Reduction↑	Accuracy Drop↓
ResNet110 on CIFAR-10	Wu et al. [2018]	65.00	−0.39
	Lin et al. [2020]	68.70	0.85
	Ours (Filters only)	60.17	**-0.93**
	Ours (Layers only)	74.75	0.82
	Ours (Filters + Layers)	**76.37**	0.98

significantly higher FLOP reduction with negligible loss in accuracy.

5.3.3.6 Time Issues

Our last experiment shows the improvement in prediction time by pruning layers. To this end, we compare the average prediction time of original, unpruned, ResNet50 with its pruned version considering different pruning ratio. Figure 5.14 (left) shows the prediction time average time considering 30 executions for predicting a single sample from the ImageNet dataset.

In practice, remove layers provides pruned networks with better prediction time than removing filters, even taking into account the same FLOP reduction. To demon-

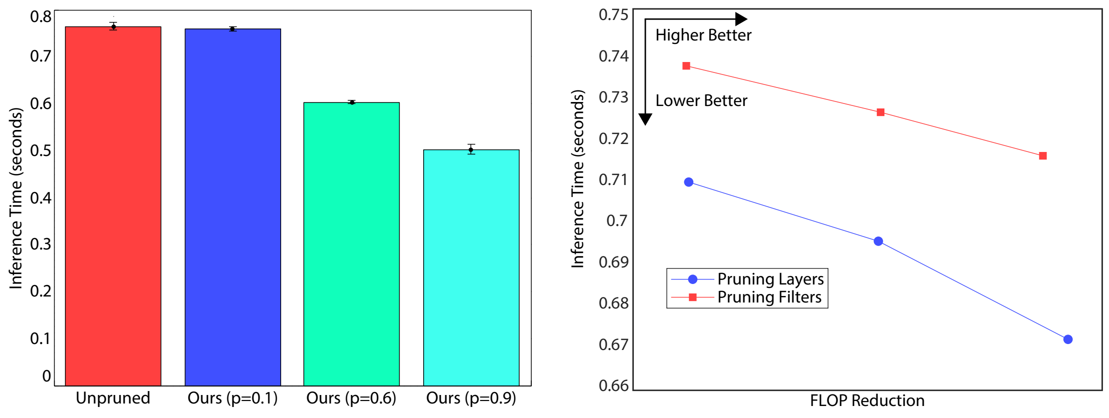


Figure 5.14. Left. Average prediction time (lower is better) of the original (unpruned) network and it after removing layers using different pruning ratio (p). Black bars denote the confidence interval. **Right.** Prediction time of pruned networks where layers or filters, but not both, are removed. On the same FLOP reduction, by removing layers provides substantially better prediction time.

strate that, in Figure 5.14 (right), we plot the prediction time resulting from the removal of filters and layers on the same FLOP reduction. It is possible to observe that removing layers provides better inference time than removing filters, even when the latter obtains more FLOP reduction. It turns out that removing layers reduces prediction latency — to compute convolutions in one layer, the model must wait for the output of all previous layers. In other words, less sequential processing improves parallelization, leading to faster prediction time. Therefore, by removing layers we achieve substantially better predictive time than removing filters, as shown in Figure 5.14 (right).

5.3.3.7 Final Remarks

We demonstrate that is possible to remove unimportant layers by estimating their importance using PLS. These results confirm our hypothesis that the relationship between layers and the class label on a low-dimensional space (PLS criterion) can be employed to identify unimportant layers to be removed. Compared to other criteria for assigning layer importance, PLS achieves competitive results while being more efficient.

In this category of pruning, most approaches are dynamic strategies, where the computational cost is conditioned to input presented to the network. Such strategies can be prohibitive to environments with a fixed computational budget. On the other hand, we show that it is possible to obtain pruned networks to which the computational cost is not conditioned to the input.

Finally, we show that strategies that remove layers and filters could benefit from each other, leaning to substantial improvements in computational performance.

Limitations. Despite the remarkable results in compressing networks, it is not possible to remove all layers composing a network. This problem takes place due to inconsistent dimensions between consecutive layers, i.e., the output of a layer is incompatible with successive layers. For example, we can remove only 33% of the modules of MobileNetV2. We believe that more sophisticated networks such as the ones generated by neural architecture search can be even more prohibitive, thus preventing us from improving their efficiency by pruning layers.

5.4 Neural Architecture Search

In this section, we introduce the experiments of the proposed strategy for designing convolutional architectures automatically. We first introduce the influence of the initial architecture (specifically its depth) on our search space, compare the proposed criterion for defining stage importance with state-of-the-art feature selection techniques, and assess the effectiveness of the proposed weight transfer mechanism, respectively. Then, we compare the built convolutional networks with human-design and NAS architectures. Next, we compare the proposed NAS method with state-of-the-art NAS approaches and demonstrate how to build architectures on large datasets, respectively. Finally, we show the generalization ability of our discovered architectures, their performance when combined to compose an ensemble and final remarks, in this order.

Throughout this section, training from scratch refers to train the architectures with random initialization for 200 epochs. Additionally, when using the proposed weight transfer technique, the architectures are fine-tuned for 50 epochs. For fairness with previous works, which adjust their final architecture using additional epochs [Dong and Yang, 2019; Brock et al., 2018; Elsken et al., 2018], at the end of each iteration of Algorithm 4 the candidate architecture is further trained for 100 epochs. Finally, unless stated otherwise, we are considering our NAS approach with residual modules.

5.4.1 Influence of Initial Depth

Our first experiment evaluates the influence of the depth b_0 of the initial architecture ($\mathcal{F}$ in step 1 of Algorithm 4). To this end, we vary b_0 from 2 to 10 in steps of 2 and measure the performance of the resulting architecture after running one iteration of our method.

According to Table 5.14, we observe that large values of b_0 lead to accurate architectures, but the computational cost increases substantially as well. For example, for $b_0 = 10$ the candidate architecture after one iteration of Algorithm 4 achieves an accuracy of 92.12 with 1.10 million parameters and 149 million FLOPs. With $b_0 = 6$, on the other hand, the first candidate architecture obtains an accuracy of 92.03 leading to significantly fewer parameters and FLOPs. Note that this behavior also occurs in residual networks. For example, ResNet56 ($b = 9$) is only 0.2 percentage points (p.p)

Table 5.14. Influence of the initial number of modules b_0 on the first candidate architecture. The arrows indicate which direction is better.

b_0	Depth	Parameters↓ (Million)	FLOPs↓ (Million)	Memory↓ (MB)	Accuracy ↑ (200 epochs)
2	23	0.36	45	3.81	91.23
4	31	0.40	64	5.31	91.57
6	43	0.60	92	7.58	92.03
8	63	0.95	139	11.56	91.98
10	67	1.10	149	12.34	92.12

Table 5.15. Accuracy on CIFAR-10 (200 epochs) of our method using different criteria for determining stage importance. The best accuracy for each iteration is shown in bold.

	Iteration				
Criterion	1	2	3	4	5
infFS [Roffo et al., 2015]	91.59	92.09	92.02	92.36	92.45
ilFS [Roffo et al., 2017]	91.94	92.06	92.10	92.08	92.52
infFS$_U$ [Roffo et al., 2020]	90.42	92.26	91.95	92.41	**92.64**
PLS+VIP	**92.03**	**92.38**	**92.62**	**92.53**	92.58

more accurate than ResNet44 ($b = 7$), see Table 5.16 (first and fourth rows).

Based on Table 5.14, the initial model using b=6 achieves a good compromise between accuracy and computational cost. Specifically, this model was the one that obtained an accuracy above 92% with the lowest computational cost; thus, we employ it as the initial model in the remaining experiments.

5.4.1.1 Importance Criteria

This experiment assesses the quality of the candidate architectures discovered applying PLS and the infFS framework to measure the importance of the stages.

According to Table 5.15, the proposed method using PLS designs more accurate candidate architectures. While the superiority of PLS could be attributed at first to the fact that it is supervised, in contrast to infFS and infFSU, we also assessed a supervised variant of infFS, ilFS, and observed the same trend (Table 5.15). This suggests that PLS is more suitable for measuring the importance of the stages.

Figure 5.15 shows the distribution of modules resulting from different criteria for measuring the importance of architecture stages. The results show that the approach

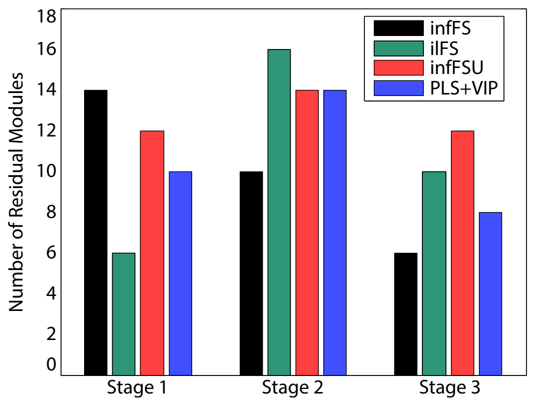


Figure 5.15. Number of residual modules, per stage (i.e. b_i), after five iterations of the proposed method using different criteria for determining stage importance.

Table 5.16. Comparison with human-designed architectures. Our architectures achieve superior accuracy and are more efficient. W. transfer indicates our weight transfer technique. The best values are shown in bold. The arrows indicate which direction is better.

Architecture	Depth	Param.↓ (Million)	FLOPs↓ (Million)	Memory↓ (MB)	Accuracy↑ (300 epochs)
ResNet44	44	0.66	97	8.14	92.83
Ours (it=1), scratch	43	0.60	92	7.41	**93.38**
Ours (it=1), W. transfer	39	**0.56**	**83**	**7.00**	93.32
ResNet56	56	0.86	125	10.42	93.03
Ours (it=3), scratch	59	**0.69**	130	10.32	93.36
Ours (it=3), W. transfer	51	0.90	**111**	**9.16**	**93.61**
ResNet110	110	1.7	253	20.67	93.57
Ours (it=5), scratch	67	**0.88**	149	11.65	**94.27**
Ours (it=5), W. transfer	59	1.23	**139**	**11.31**	93.51

used to measure importance has significant impact on the final architecture. In addition, our method applying PLS, ilFS and InfFS$_U$ inserted more modules on the middle stage, which means increasing its depth brings improvements to the architecture. Importantly, this behavior is consistent with the work by Wang et al. [2018b], where they demonstrate that removing layers from the middle stage degrades accuracy more than other stages. This supports the fact that our approach is capable of identifying stages that need become deeper (most important) and the ones that could be kept shallow (least important).

Based on these results, in the next experiments, we report results considering one, three and five iterations and 300 epochs of training. We observe that increasing the number of iterations above five does not improve accuracy substantially enough to justify the increase in computational cost. In particular, to the criteria infFS$_U$ and PLS+VIP, the candidate architectures discovered using more than five iterations obtained accuracy inferior to architectures in Table 5.15. We refer the reader to Appendix C (Tables C.1 and C.2) for the results of other iterations.

5.4.1.2 Weight Transfer

Our next experiment evaluates the behavior of the proposed NAS method when transferring knowledge (weights) from existing networks to our candidate architectures. For this purpose, we first define an existing (pre-trained) network to provide the weights for the modules of the candidate architectures. In this work, we employ ResNet110 due to its high accuracy. We highlight that since our candidate architectures are very shallow, we could employ shallower ResNets (e.g., ResNet56) and still avoid the restriction that the depth of the candidate architecture cannot exceed the depth of the network providing the weights (see Section 4.2).

Table 5.16 summarizes the results. Compared to training from scratch, our method with the weight transfer technique yields higher performance architectures in terms of depth, memory usage, number of FLOPs and parameters. The reason for different architectures when using training from scratch and the weight transfer technique (fine-tuning) is that the weights of the networks are different and influence the importance score directly. This leads to the insertion of modules on different stages throughout iterations.

Besides designing higher-performance architectures, an advantage of weight transfer is that it reduces the computational burden of training the architectures from scratch. Specifically, this strategy reduces the average time for each iteration from 17 to 3 hours. We emphasize that the time for generating our architectures is faster than previous works, which require many days on several GPUs even when training for a few epochs (i.e., 20) [Zoph et al., 2018; Baker et al., 2017; Real et al., 2017]. For example, the approaches by Baker et al. [2017] and, Real et al. [2017] require 10 days to discover competitive architectures. It is worth mentioning that our method could be made even faster by training/fine-tuning architectures for only a few epochs, as suggested by previous works. On the other hand, this strategy can yield poor architectures, harming the search process [Dong and Yang, 2020; Sciuto et al., 2020].

In summary, the proposed weight transfer technique speeds-up our NAS approach considerably, which enables searching architectures directly on large datasets. However, when using this technique the candidate architectures can take advantage of well-trained networks. Therefore, to make a fairer comparison with other NAS, unless stated otherwise, we are considering our method with training from scratch.

5.4.1.3 Comparison with human-designed architectures

As we mentioned previously, human-designed architectures are generally composed of stages with uniform depth. Our method, on the other hand, designs architectures by adjusting the depth for each stage based on its importance. To demonstrate that this process leads to more efficient and accurate networks, in this experiment, we compare our discovered architectures to their human-designed counterpart.

Table 5.16 compares our architectures with residual modules to existing residual networks [He et al., 2016]. Compared to these networks, our architectures achieve superior performance in terms of the number of parameters, FLOPs, memory usage

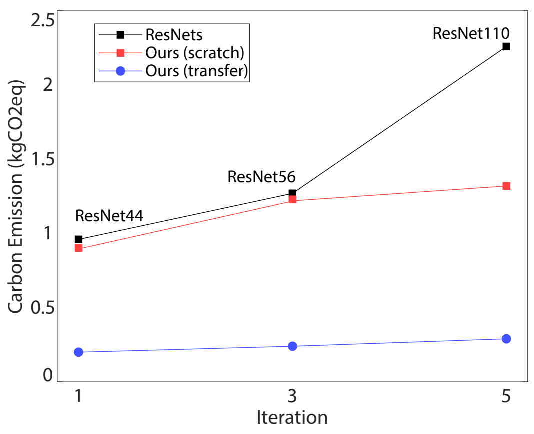


Figure 5.16. Carbon emission for training architectures (the lower the better). From iteration one to five, our architectures have their carbon emission increased slightly whereas from ResNet44 to ResNet110 the increase is notably higher. CO_2eq indicates the global warming potential of various greenhouse gases as a single number.

and accuracy. In particular, with one iteration our discovered architecture achieves comparable accuracy to ResNet110, having less than half of its computational cost. More importantly, these results show that adjusting depth on a stage-by-stage basis enables increasing capacity (reflected by accuracy) of networks without compromising their efficiency.

Following a recent trend [Lacoste et al., 2019; Schwartz et al., 2020; Strubell et al., 2019], we also measure the carbon emission for training architectures. For this purpose, we use the Machine Learning Emissions Calculator[4] and report the CO_2-equivalents (CO_2eq), which indicates the global-warming potential of various greenhouse gases as a single number. According to Figure 5.16, our candidate architectures emit notably less carbon, even taking into account shallow versions of ResNet. Compared to ResNet110, our final architecture trained from scratch emits 41% less CO_2. Observe that, from iteration one to five, our architectures have their carbon emission increased slightly whereas from ResNet44 to ResNet110 the increase is notably higher. This occurs because our architectures are computationally more efficient, leading to a considerably faster training stage. Moreover, ResNets (as well as most human-designed architectures) have the same depth for all stages. Our architectures, on the other hand, had the depth of stages adjusted according to its importance, avoiding unnecessary growth in computational cost.

[4]https://mlco2.github.io/impact/

Table 5.17. Performance of networks built with the proposed method and cell modules discovered by NAS. The best values are shown in bold. The arrows indicate which direction is better.

Architecture	Depth	Param.↓ (Million)	FLOPs↓ (Billion)	Memory↓ (MB)	Accuracy↑ CIFAR-10
NASNet169	169	2.3	2.7	71.26	94.34
Ours (it=1)	109	**1.3**	**1.8**	**45.20**	**94.55**
NASNet205	205	2.8	3.2	86.37	94.37
Ours (it=3)	133	**1.5**	**2.2**	**56.96**	**94.63**
NASNet241	241	3.3	3.8	101.47	94.51
Ours (it=5)	181	**2.3**	**2.9**	**78.87**	**94.74**

5.4.2 Combination with other NAS approaches

As we mentioned earlier, our method can employ modules discovered by other NAS approaches. We highlight that the NAS approaches that focus on discovering cells define the depth of stages uniformly (similar to human-designed architectures). In this experiment, we show that using these modules coupled with our strategy provides even better architectures.

Table 5.17 shows the results of our method applying the cells by Zoph et al. [2018] as modules. Similar to residual modules, our architectures based on cells outperform those based on stages with uniform depth. Compared to the original NASNet ($b = 6$ for all stages — 241 layers deep) by Zoph et al. [2018], with one iteration, our discovered architecture achieves superior accuracy having 60%, 52% and 55% fewer parameters, FLOPs and memory usage.

5.4.3 Comparison with state-of-the-art NAS

This experiment compares our method with state-of-the-art NAS approaches.

According to Table 5.18, our method is the more cost-effective NAS approach in terms of the number of evaluated models and amount of GPUs required. Compared to Baker et al. [2017] and, Real et al. [2017], our method designs competitive architectures by evaluating a significantly smaller number of models, enabling the proposed method to run in a few hours on a single GPU. Specifically, our method evaluates one order of magnitude fewer models. This is because instead of analyzing a large pool of architectures like most approaches, we increment previous architectures iteratively while taking into account the importance of the components to be inserted. This advantage enables our method to scale to large datasets, while most NAS approaches might

Table 5.18. Comparison with state-of-the-art NAS approaches. Results taken from previous works. The best values are shown in bold. '–' indicates the metric is not reported by the original paper. The arrows indicate which direction is better.

Model	Evaluated↓ Models	GPUs↓	Param.↓ (Million)	Accuracy↑ CIFAR-10
Zoph et al. [2018]	20,000	800	2.5	94.51
Baker et al. [2017]	1,500	10	11.1	93.08
Real et al. [2017]	1,000	250	5.4	94.60
Brock et al. [2018]	300	**1**	4.6	94.47
Dong and Yang [2019]	240	**1**	2.5	96.25
Yang et al. [2020b]	128	1	3.6	97.38
Jin et al. [2019]	≈60	**1**	–	88.56
Elsken et al. [2018]	40	5	19.7	94.80
Kandasamy et al. [2018]	**10**	4	–	91.31
Chen et al. [2019]	–	**1**	10.5	**97.75**
Li et al. [2020b]	–	1	3.90	96.21
Ours (Res. modules)	11	**1**	**1.7**	94.27
Ours (Cell modules)	11	**1**	2.3	94.74
Ours (Ensemble)	–	–	7.27	95.68

be prohibitive even when employing morphism and other optimization techniques [Ha et al., 2017; Dong and Yang, 2020]. Compared to approaches that also evaluate a small number of models [Elsken et al., 2018; Kandasamy et al., 2018; Jin et al., 2019], our method achieves the best tradeoff between accuracy and number of GPUs required.

In summary, our method achieves competitive results using both residual and cell modules. When considering our best setting (cell modules with five iterations, see Table 5.17), only the most recent approaches [Elsken et al., 2018; Dong and Yang, 2019; Chen et al., 2019; Yang et al., 2020b; Li et al., 2020b] obtain superior accuracy. We emphasize that our training process employs a simple SGD optimizer with standard data augmentation, while other NAS approaches employ sophisticated optimizers and regularization techniques (e.g., SGDR [Loshchilov and Hutter, 2017] and Scheduled-DropPath [Zoph et al., 2018]). Although we could use these setups, they render it more difficult to identify which aspects actually lead to the improvement in NAS, as argued by Dong and Yang [2020]. Thus, we prefer to maintain the training as simple as possible.

Finally, our method built more parameter-efficient architectures even without considering the computational cost in the searching process. This occurs since most

NAS approaches focus on discovering components of the architecture while keeping a uniform distribution of depth of the stages. Instead, our method adjusts this depth based on its importance leading to shallower, and hence more efficient, architectures.

5.4.4 Learning Architectures on Large Datasets

Since our approach explores few candidate architectures, it is scalable to large datasets. To reinforce this, we apply the proposed method to discover architectures on the large ImageNet (224×224) dataset. We use bottleneck residual blocks [He et al., 2016] as modules and the weight transfer technique. Due to limitations for training NAS-based architectures, we do not consider cell modules.

Our final architecture obtained a top-5 accuracy of 90.23, which is less than one percentage point inferior to the architecture by Dong and Yang [2019]. Although it achieved a lower accuracy, an advantage of our method is that it can be applied on large datasets without requiring careful parameter setting since we are using the same parameters found on CIFAR-10. This advantage is desirable when no resources are available for tuning such parameters. We highlight that the approach by Dong and Yang [2019] fails to design networks directly on ImageNet, as it needs careful tuning and different hyper-parameters.

5.4.5 Generalization Ability

An alternative to learning architectures on large datasets is to design them on small datasets and then transfer them to large datasets. The accuracy of the resulting architecture (trained from scratch) can be used to estimate its generalization ability (i.e., transferability), which is a desirable property in NAS [Zoph et al., 2018]. In this experiment, we assess this generalization ability of our architectures. For this purpose, we follow the same process by Zoph et al. [2018], which consists of taking an architecture found for CIFAR-10 and training it from scratch on other datasets.

Table 5.19 shows the top-5 accuracy obtained on the Tiny ImageNet and ImageNet 32×32 datasets. For both datasets, when using residual modules, our architectures outperformed those based on stages with uniform depth. Specifically, our architecture obtained an accuracy up to 2.4 p.p superior to ResNet110. With cell modules, our architecture and NASNet241 achieved similar results. In summary, these results show that our architectures present high generalization ability.

Table 5.19. Accuracy of networks transferred from a small dataset (i.e., CIFAR-10) to large datasets. The higher the accuracy the higher the generalization ability. The best accuracy is shown in bold.

Architecture	TinyImageNet	ImageNet 32×32
ResNet110	69.94	68.89
Ours (Res. modules)	**72.34**	**70.09**
NASNet241	79.20	**80.67**
Ours (Cell modules)	**79.23**	79.39

5.4.6 Ensemble of Architectures

Motivated by the fact that an ensemble of candidate architectures can obtain better accuracy than the final architecture alone [Elsken et al., 2018], our last experiment shows the performance of our method employing this strategy.

Our ensemble is composed of the candidate architectures presented in Tables 5.16 and 5.17, achieving an accuracy of 95.68 with 7.27 million parameters. Compared to the ensemble of Elsken et al. [2018], which obtains an accuracy of 95.60 with 88 million parameters, our ensemble is marginally more accurate having 12× fewer parameters. In particular, our ensemble is more parameter-efficient even when compared to a single architecture of Elsken et al. [2018] (see Table 5.18).

5.4.7 Final Remarks

We demonstrate that it is possible to design high-performance convolutional architectures by inserting layers (i.e., adjusting the depth) based on their importance. This importance is assigned by PLS and confirms our hypothesis that the relationship between layers and the class label on a low-dimensional space (PLS criterion) can be employed to define layer importance. Compared to previous NAS approaches, our method is significantly more efficient since it evaluates one order of magnitude fewer models. Our discovered architectures are on par with the state of the art and present high generalization ability, as they can be learned on a small dataset and successfully transferred to large datasets.

Limitations. Since our NAS relies on predefined modules, we are not able to discover all components of the architecture (e.g., number of filters and connections) from scratch. Additionally, due to the incremental essence of our NAS (an architecture i must wait for the architecture $i - 1$), we cannot parallelize the search. Thus, even when more resources (i.e., GPU) are available for parallel processing, our method is not able to take advantage of such resources.

5.5 Latent HyperNet

In this section, we introduce the experiments of our strategy for combining multiple layers from convolutional networks. We first describe the convolutional networks used to assess the quality of the proposed method and how to define what layers to combine, respectively. Next, we present the improvements achieved by combining multiple layers and the computational cost of this process. Afterward, we show the importance of project the multiple features onto a compact space and demonstrate the performance of our method on lightweight architectures, in this order. Finally, we discuss time issues and final remarks.

5.5.1 Convolutional Networks

To demonstrate that our LHN is effective across different architectures, throughout the experiments, we employ it on different convolutional networks such as VGG [Simonyan and Zisserman, 2015], ResNet [He et al., 2016] and MobileNet [Sandler et al., 2018]. Unfortunately, these architectures are unsuitable for activity recognition based on wearable sensor data due to the structure of the data. Therefore, to this task, we propose three different architectures (*Arch1*, *Arch2* and *Arch3*), which vary in the number of filters and layers, and filter dimensions, as shown in Table 5.20. Following the work by Chen and Xue [2015], after each convolutional layer, we apply a 2×1 max-pooling operation.

We highlight that the proposed convolutional architectures are only used on the activity recognition task, where there are no well-defined architectures.

Table 5.20. Configurations of the convolutional architectures apply to activity recognition. After each convolutional layer, a 2×1 max-pooling operation is employed.

	Number of Layers	Number of Filters per Layer	Filter Dimensions per Layer (height × width)
Arch1	2	$24, 32$	$12 \times 2, 12 \times 2$
Arch2	3	$24, 32, 40$	$6 \times 1, 8 \times 1, 10 \times 1$
Arch3	4	$24, 32, 40, 48$	$12 \times 1, 12 \times 1, 6 \times 1, 2 \times 1$

5.5.2 Defining Layers to be Combined

Because modern architectures are very deep (dozens or even hundreds of layers), a typical step in HyperNet approaches is to select only a subset of the layers to be

Table 5.21. Accuracy on CIFAR-10 (validation set) using different combinations of layers. FC denotes the employment of the first fully connected layer of the VGG16 architecture. The better accuracy is shown in bold. The symbol '–' denotes that it was not possible to execute the method in the respective setting due to incompatible dimensions between 2D (layers 41, 45, 49) and 1D (FC layer) features maps.

(a) VGG16 layers.

Layers	HyperNet	LHN
$41 + 45$	87.02	87.00
$41 + 49$	87.46	87.06
$45 + 49$	**87.64**	87.90
$41 + 45 + 49$	87.44	86.94
$41 + 45$+FC	–	86.70
$41 + 49$+FC	–	**88.12**
$45 + 49$+FC	–	87.40
$41 + 45 + 49$+FC	–	86.78

(b) ResNet20 layers.

Layers	HyperNet	LHN
$53 + 60$	86.08	79.04
$53 + 67$	86.48	85.88
$60 + 67$	**87.10**	86.10
$53 + 60 + 67$	86.36	**86.68**

combined. Besides reducing computational cost, this step might provide better results since adjacent layers are correlated [Kong et al., 2016].

To determine what layers to combine, we employ the same procedure as suggested by Kong et al. [2016], which consists of generating a small set of possible combinations and evaluating the accuracy of the HyperNet for each combination. To further reduce the number of combinations, we examine only intermediary and deep layers. Table 5.21 (a) and (b) show the accuracy of the HyperNets when combining different layers of VGG16 and ResNet20, respectively.

According to Table 5.21 (a), on the VGG16 network, the HyperNet by Kong et al. [2016] attained the best results combining the layers 45 and 49. Our method, on the other hand, obtained the best results combining the layers 41, 49 and the first fully connected layer (FC). On the ResNet20 network, Table 5.21 (b), the HyperNet by Kong et al. [2016] achieved the best results combining the layers 60 and 67. Similarly, our LHN obtained the best accuracy combining the layers 60, 67 and 53. Note that, while our method enables combining any layer that composes the network, the approach of Kong et al. [2016] is limited to combine convolutional layers only.

The results above consider one PLS model per layer (as explained in Section 4.3). An alternative to this modeling is to concatenate all the feature maps provided by all the layers (the ones to be combined) and then, learn a single PLS model, as illustrated in Figure 5.17. However, the memory consumption increases significantly since the result of this concatenation is a high dimensional space and could unfeasible the em-

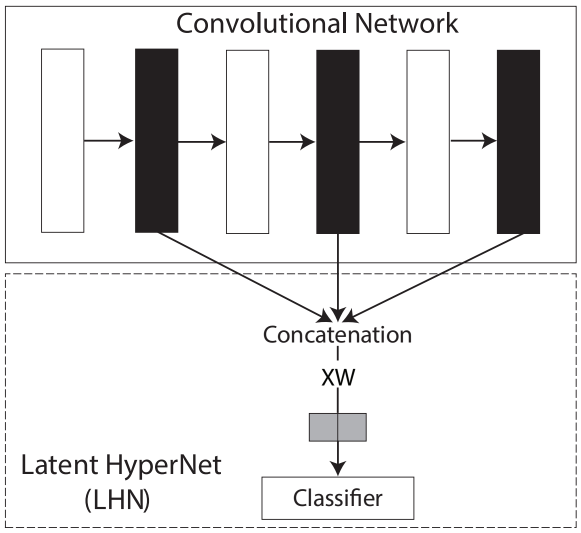


Figure 5.17. Latent HyperNet considering a single PLS projection. After setting the layers to be combined (represented by black boxes), we concatenate their features maps yielding a data matrix X. Then, we learn a single PLS projection (matrix W) using X. Finally, we project (XW) and present the low-dimensional feature maps to a classifier. In this example, PLS projects the high-dimensional feature maps onto two dimensions.

ployment of LHN on resources-constrained applications. For example, on VGG16, the concatenation of the layers $41, 49$ and FC yields a feature space with around $3\times$ more dimensions when compared to the scheme layer-by-layer. We observe that the accuracy of both strategies is similar; therefore, it is more efficient to learn PLS layer-by-layer.

Once found the best combinations of layers on CIFAR-10, we apply them in the remaining experiments, including experiments on the ImageNet dataset.

Different from convolutional networks designed to image applications, in activity recognition, the convolutional networks are very shallow. For example, the deepest architectures do not exceed 3-4 layers [Chen and Xue, 2015; Ha et al., 2015; Ha and Choi, 2016; Xu et al., 2018]. Therefore, on activity recognition, we combine all the layers composing the convolutional network.

5.5.3 HyperNet Improvements

In this experiment, we evaluate the improvements achieved by the HyperNets approaches. To make a fair comparison and show the improvement obtained by the exploration of different layers only, we use the same classifier employed by the original convolutional network, an MLP classifier (fully-connected layers with softmax activation). In this way, the improvements are not biased by the classifier.

Image Classification. Table 5.22 shows the improvements in accuracy achieved by

Table 5.22. Improvements in accuracy achieved by the HyperNets. Negative values denote a **decrease** in accuracy regarding the original network. The better method is shown in bold. The arrows indicate which direction is better.

		CIFAR-10↑	ImageNet (32 × 32)↑
VGG16	HyperNet [Kong et al., 2016]	−0.22	0.01
	LHN (Ours)	**0.05**	**0.66**
ResNet20	HyperNet [Kong et al., 2016]	**-0.02**	**3.60**
	LHN (Ours)	−0.13	2.65

the HyperNets approaches using multiple layers from VGG16 and ResNet20. According to this table, on CIFAR-10, the approach by Kong et al. [2016] was not able to improve the accuracy compared to the original network. In contrast, our LHN obtained a marginal improvement.

On the ImageNet dataset, the approach by Kong et al. [2016] improved the accuracy of VGG16 and ResNet20 in 0.01 p.p. and 3.60 p.p., respectively. On the other hand, LHN improved the accuracy of VGG16 and ResNet20 in 0.66 p.p. and 2.65 p.p.. We emphasize that the layers combined in ImageNet are the ones selected on CIFAR-10 (see Table 5.21).

In summary, HN and LHN attained similar results, which reinforces the ability of PLS to model multiple levels of features when compared to time-consuming operations, such as the ones used in HyperNets (e.g., convolutions). To reinforce this claim, we show the behavior of the softmax distribution (output of the softmax layer) when both strategies assign a sample to its correct and incorrect label. Figure 5.18 shows these

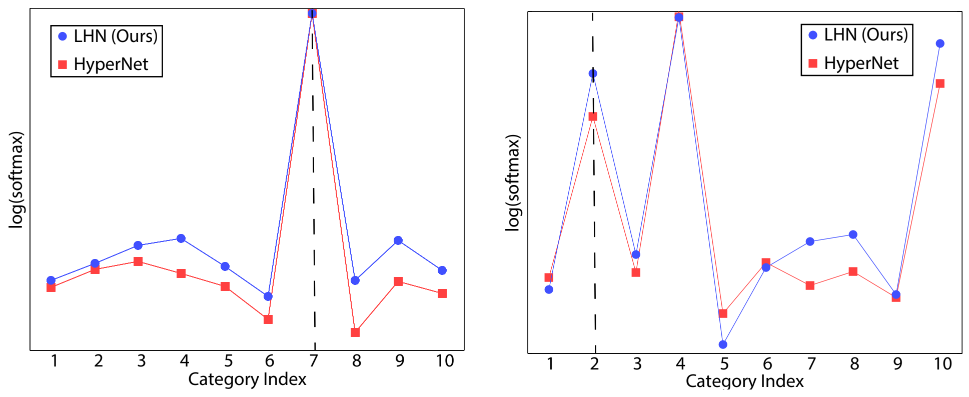


Figure 5.18. Left. Distribution of the softmax layer when a sample is assigned to its correct category. **Right.** Distribution of the softmax layer when a sample is assigned to an incorrect category. Dashed line indicates the expected (corrected) label. We omitted the y-axis values due to different scales.

distributions. From these figures, it is evident that both strategies obtain similar softmax distributions. Even though the behavior in Figure 5.18 has been observed across different samples, we measure the divergence between the distributions considering all samples of CIFAR-10. For this purpose, similar to Luo and Wu [2020], we compute the KL-divergence between the softmax of HN and LHN for each sample and average its values. This value was significantly small, 0.0001, confirming that the distributions are very similar.

Activity Recognition. On the activity recognition task, it was not possible to compare our method with the approach by Kong et al. [2016] due to the design of the convolutional networks. For example, convolutional networks applied to activity recognition often employ large filters [Chen and Xue, 2015; Ha et al., 2015; Xu et al., 2018], yielding feature maps much smaller in subsequent layers and prohibiting the combination among layers. On the contrary, LHN enables us to employ layers that provide feature maps of different sizes, as we show by combining convolutional and fully connected layers (see Table 5.21 last rows).

Table 5.23 shows the improvements achieved by LHN on different architectures. In this table, the ith LHN represents LHN using the proposed ith architecture (defined in Table 5.20). As shown in Table 5.23, the proposed method was able to enhance all networks, except Arch2 on USCHAD. On the paired t-test, LHN obtained an accuracy statistically superior to the original network except on the USCHAD dataset. In particular, LHN was able to improve up to 9.57 p.p. the accuracy. Considering all datasets in Table 5.23, LHN was able to improve the architectures 1 and 3, on average, 4.30 p.p. and 1.17 p.p., respectively. Moreover, on the architecture by Chen and Xue [2015] (LHNC), the use of LHN improved the accuracy, on average, in 1.01 p.p..

Table 5.23. Improvements in accuracy achieved by LHN on different architectures. The ith LHN represents LHN using the proposed ith architecture (defined in Table 5.20). LHNC indicates LHN using the architecture by Chen and Xue [2015]. The numbers enclosed in square brackets denote confidence interval (95% confidence). The symbol '–' denotes that it was not possible to evaluate the architecture on the respective dataset due to its design — the feature map achieves zero size in deep layers. The better accuracy improvement is shown in bold.

	USCHAD	WISDM	UTD-MHAD1	UTD-MHAD2
LHN1	**2.96** [1.7, 4.1]	0.22 [−0.5, 0.9]	**9.57** [7.1, 12.0]	**4.45** [1.3, 7.5]
LHN2	−1.04 [−6.8, 4.7]	0.13 [−0.7, 0.9]	–	–
LHN3	0.80 [−2.1, 3.7]	**1.55** [0.9, 2.1]	–	–
LHNC	1.55 [−1.7, 4.8]	0.45 [−0.1, 1.0]	–	–

5.5.4 Computational Cost

Even though improving accuracy, HyperNet approaches incur a high computational cost. In this experiment, we compare the computational cost, measured by FLOPs, of the HyperNets approaches.

According to Table 5.24, considering all settings (architectures $\times$ datasets), our LHN requires fewer FLOPs than Kong et al. [2016]. Specifically, our method has, on average, $1,690,368$ fewer FLOPs than Kong et al. [2016]. It turns out that we use simple projections to combine layers while Kong et al. [2016] apply convolution operations, which increases the number of FLOPs substantially. We empathize that, compared to Kong et al. [2016], our method always uses one additional layer (see Table 5.21 - bold values), which reinforces its efficiency in combining layers.

Table 5.24. Floating point operations (FLOPs) of HyperNets approaches. Values are in million. The lowest FLOPs is shown in bold. The arrows indicate which direction is better.

	Method	CIFAR-10↓	ImageNet (32 × 32)↓
VGG16	HyperNet [Kong et al., 2016]	313.54	314.05
	LHN (Ours)	**313.22**	**313.72**
ResNet20	HyperNet [Kong et al., 2016]	43.91	44.42
	LHN (Ours)	**40.85**	**41.36**

5.5.5 Importance of Dimensionality Reduction

The core of our LHN is the dimensionality reduction step, which projects high-dimensional feature maps onto a compact space. In this experiment, we show the importance of this step. To this end, we measure the results of LHN without the dimensionality reduction step on the CIFAR-10 and USCHAD datasets. More concretely, we present the feature maps from layers directly to the classifier rather than projecting them using PLS.

By removing the dimensionality reduction, the accuracy decreased 30 and 0.09 p.p., on average, on the activity recognition and image classification tasks, respectively. This behavior takes place because of the high dimensionality generated from the concatenation of the feature maps, rendering the learning stage more complex since the network needs to learn a larger number of parameters. For example, employing the VGG16 architecture on LHN without dimensionality reduction, the number of parameters increased from 15 million to 17 million. In contrast, by using the dimensionality

Table 5.25. Accuracy of LHN on CIFAR-10 (test set) using different combinations of layers from lightweight networks. The combinations of layers that achieved improvement regarding the original network are shown in bold.

(a) MobileNetV1 layers.

Layers	LHN
$67 + 70$	87.46
$67 + 73$	87.59
$67 + 76$	**87.73**
$70 + 73$	87.52
$70 + 76$	**87.71**
$73 + 76$	**87.62**
$67 + 70 + 73$	87.48
$67 + 70 + 76$	**87.75**
$67 + 73 + 76$	**87.72**
$70 + 73 + 76$	**87.62**
$67 + 70 + 73 + 76$	87.58

(b) MobileNetV2 layers.

Layers	LHN
$131 + 137$	89.48
$131 + 143$	89.43
$131 + 148$	89.37
$137 + 143$	89.65
$137 + 148$	89.43
$143 + 148$	89.51
$131 + 137 + 143$	89.48
$131 + 137 + 148$	89.39
$131 + 143 + 148$	89.46
$137 + 143 + 148$	89.55
$131 + 137 + 143 + 148$	89.48

reduction technique, we generate a low-dimensional feature space, which aids the learning phase and reduces the computational cost as well.

5.5.6 Latent HyperNet on Lightweight Networks

Our next experiment evaluates the performance of LHN on lightweight networks. To this end, we employed the popular MobileNetV1 [Howard et al., 2017] and MobileNetV2 [Sandler et al., 2018] architectures. As before, we examine the accuracy of LHN when considering different combinations of layers, as shown in Table 5.25. In this table, bold values indicate the combinations where LHN achieved better accuracy than the original network.

On the MobileNetV1 architecture, LHN was able to improve accuracy in 54% combinations. However, on MobileNetV2, LHN did not obtain any improvement considering all combinations. It worth noting that MobileNetV1 is a plain network while MobileNetV2 is a residual-based architecture. Thereby, these results are similar to the ones found in CIFAR-10 (see Table 5.22), where LHN improved accuracy over VGG16 (plain network) but underperformed accuracy over ResNet20 (residual network).

Besides MobileNets, we also assess LHN on a pruned network, which in turn has become a lightweight (efficient) architecture. We consider the pruned network in which we remove layers and filters of its architecture (specifically, the network of Table 5.13

last row). On different combinations of layers, LHN was not able to improve accuracy. According to previous results, these results are not surprising since LHN on residual architectures (ResNet and MobileNetV2) presented poor results. However, we observe that the pruned architecture was more sensitive to the combination of layers, as it obtained substantial inferior accuracy.

5.5.7 Time Issues

Since LHN performs a projection (dimensionality reduction) after each layer, it introduces an extra cost at the prediction stage. In this experiment, we show that this cost is negligible, rendering LHN computationally efficient compared to the traditional convolutional network. To demonstrate that, we perform the paired t-test on the prediction time considering 30 executions.

Figure 5.19 shows the average prediction time and the confidence interval of the original convolutional network and the ones using LHN. On the paired t-test, the prediction times of the original network and LHN were statistically equivalent, indicating that the employment of LHN does not compromise the prediction time.

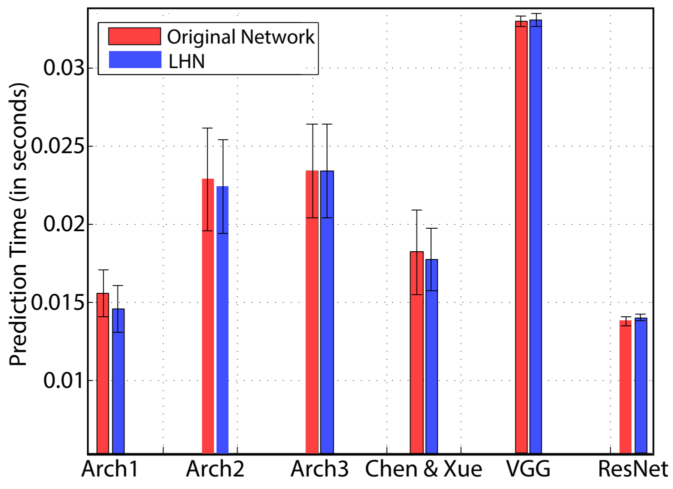


Figure 5.19. Average prediction time (lower values are better) of the original network (red bars) and LHN (gray bars). The proposed LHN does not compromise the prediction time since its time is statistically equivalent to the original convolutional network time. Black bars denote the confidence interval.

5.5.8 Final Remarks

We demonstrate that an efficient yet effective way of combining multiple levels of features is to project them on the latent space of PLS. Compared to time-consuming operations, we demonstrate that PLS projection enhances data representation (reflected by accuracy) at negligible additional cost. More importantly, these results confirm our hypothesis that the relationship between network structure and the class label on a low-dimensional space (PLS criterion) can be employed to combine multiple levels of representation distributed across the network.

Limitations. Despite promising results, our strategy for exploring multiple layers exhibits some limitations. For example, to employ a network with LHN to other datasets (or tasks), we need to retrain the PLS from scratch since the number of categories and the average of data (Z-score) are different, which might lead to notable divergence in the latent space. In other words, the PLS model learned on one dataset does not ensure the maximum covariance between data and labels in another dataset, thus the latent data might not be discriminative. Other HyperNets, however, can only recondition the weights of the operations (i.e., convolutions) applied to multiple layers. Furthermore, our LHN also does not allow that the weights of the network and the projection matrix be trained jointly. Thus, it is suitable for applications where the network can be used simply as a feature extractor.

5.6 Covariance-free Partial Least Squares

In this section, we compare the proposed incremental Partial Least Squares, named *Covariance-free Incremental Partial Least Squares* (CIPLS), with other methods as well as with the traditional PLS. Afterward, we present the influence of higher-order components on the classification performance. Finally, we discuss the time complexity of the methods, their performance on a streaming scenario and compare our method on the feature selection context.

5.6.1 Comparison with Incremental Methods

This experiment compares the proposed CIPLS with other incremental dimensionality reduction methods. Table 5.26 summarizes the results.

Table 5.26 shows that, on the LFW dataset, CIPLS outperformed SGDPLS and IPLS in 1.18 and 1.48 p.p., respectively. Similarly, on the YTF dataset, CIPLS outperformed SGDPLS and IPLS in 0.88 and 1.88 p.p., in this order. In particular, on these datasets, the results of CIPLS were statistically superior to IPLS and SGDPLS. As we argued before, to perform the paired t-test on face verification, we use 90% confidence. However, by using 95% confidence, our CIPLS still presented results statistically superior.

Table 5.26. Comparison of existing incremental methods in terms of accuracy. The symbol '–' denotes that it was not possible to execute the method on the respective dataset due to memory constraints or convergence problems (see the text). PLS denotes the use of the traditional PLS. The closer to the accuracy of the baseline (PLS), the better. The numbers enclosed in square brackets denote confidence interval (95% confidence).

	LFW	YTF	ImageNet 32×32	ImageNet 224×224
CCIPCA	89.87 [89.17, 90.55]	81.48 [80.07, 82.88]	40.30	52.58
SGDPLS	90.60 [89.95, 91.24]	83.22 [82.07, 84.36]	–	–
IPLS	90.30 [89.60, 90.99]	82.22 [80.96, 83.47]	43.24	65.74
CIPLS (Ours)	91.78 [91.08, 92.47]	84.10 [82.82, 85.37]	43.31	67.09
PLS	92.47 [91.87, 93.05]	85.96 [84.47, 87.44]	–	–

On the ImageNet dataset, the difference in accuracy compared to IPLS was of 0.07 and 1.35 p.p., for the 32× and 224×224 versions, respectively. It is important to mention that we do not consider SGDPLS on these datasets due to convergence problems and the high computational cost. Also, due to memory constraints, it was not possible to run the traditional PLS on the ImageNet datasets.

5.6.2 Comparison with Partial Least Squares

As suggested by Weng et al. [2003], we compare the incremental methods with the traditional approach as baseline (in our case, traditional PLS). According to Table 5.26, besides providing better results than IPLS and SGDPLS, CIPLS achieved the closest results to traditional PLS. For instance, on LFW, the difference in accuracy between PLS and CIPLS was 0.69 p.p. while on YTF it was 1.86 p.p.. In contrast, the difference in accuracy between PLS and SGDPLS is higher — 1.87 p.p. on LFW and 2.74 p.p. on YTF. In addition, the difference in accuracy between PLS and IPLS is among the highest, 2.17 and 3.74 p.p. for the LFW and YTF datasets, respectively. In particular, the results for PLS and CIPLS are statistically equivalent, while IPLS and SGDPLS present results statistically inferior compared to PLS.

It should be noted that the results of IPLS are closer to CCIPCA than PLS since only the first component of IPLS maintains the relationship between independent and dependent variables. On the other hand, the proposed method preserves this relation along higher-order components, which provides better discriminability, as seen in our results.

5.6.3 Higher-order Components

In this experiment, we assess the discriminability of the higher-order components of CIPLS compared to each of the other incremental methods. For this purpose, we follow a process suggested by Martínez and Kak [2001], which consists of removing the first component of the latent space before presenting the projected data to the classifier. This evaluates the performance of the remaining components, not only the first one which tends to be better. By performing this process, our method outperformed IPLS on average[5] in 32.48 p.p.. Observe that when all the components are used, CIPLS outperformed IPLS in 1.77 p.p.. This larger difference when removing the first component is an effect of the better discriminability achieved by the components extracted

[5]Value computed considering the accuracy of all the datasets.

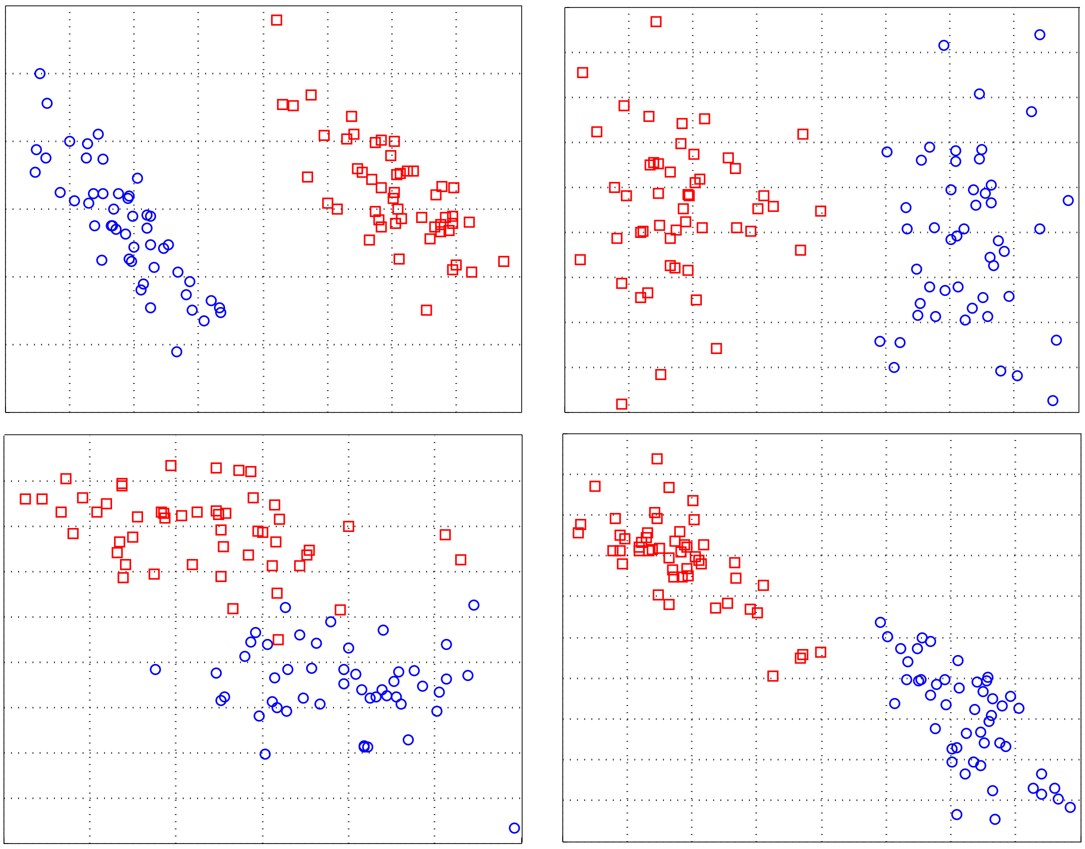


Figure 5.20. Projection on the first (x-axis) and second (y-axis) components using different dimensionality reduction techniques. **Top-left.** PLS projection. **Top-right.** IPLS projection. **Bottom-left.** SGDPLS projection. **Bottom-right.** CIPLS projection. Our CIPLS separates the feature space better than IPLS and SGDPLS, which are state-of-the-art incremental PLS-based methods. Blue and red points denote positive and negative samples, respectively.

by CIPLS. As we have argued, CIPLS preserves the relationship between dependent and independent variables across higher-order components, yielding more accurate results. Compared to SGDPLS, CIPLS outperforms it in 24.83 p.p. when using only the higher-order components.

Figure 5.20 reinforces the above discussion, where it is possible to note that CIPLS yields more discriminative components when compared to other incremental Partial Least Squares methods.

5.6.4 Time Issues

To demonstrate the efficiency of CIPLS, in this experiment, we compare its time complexity to compute the projection matrix with the incremental methods evaluated. Following Weng et al. [2003] and, Zeng and Li [2014], we report this complexity w.r.t. dimensionality of the original data (m), number of samples (n), number of components (c) and number of PCA components (L - required only by IPLS and CCIPCA). Table 5.27 shows the time complexity of the methods.

According to Table 5.27, our method presents a low time complexity for estimating the projection matrix. The complexity of CIPLS is not only on the same class as CCIPCA, which is the fastest among the compared methods, but it also has a very small constant factor. This constant factor is the number of components, c for CIPLS and L for CCIPCA. In our experiments, we found that the optimal constant factor for the former is negligible (i.e., $c = 2$). In other words, $c < L$ on practical applications and this is a known advantage of PLS, where it has been shown to require substantially less components to achieve its optimal accuracy than PCA [Schwartz et al., 2009].

To show the efficiency (on practical terms) of our CIPLS, we report the average computation time (considering 30 executions) of the methods for estimating the projection matrix to one new sample. To make a fair comparison, we set $c = 2$ for all methods

Table 5.27. Comparison of incremental dimensionality reduction methods in terms of time complexity for estimating the projection matrix. m, n denote dimensionality of the original data and number of samples, while c, L and T denote number of PLS components, number of PCA components and convergence steps, respectively.

	Time Complexity
CCIPCA [Weng et al., 2003]	$O(nLm)$
SGDPLS [Arora et al., 2016]	$O(Tcm)$
IPLS [Zeng and Li, 2014]	$O(nLm + c^2m)$
CIPLS (Ours)	$O(ncm)$

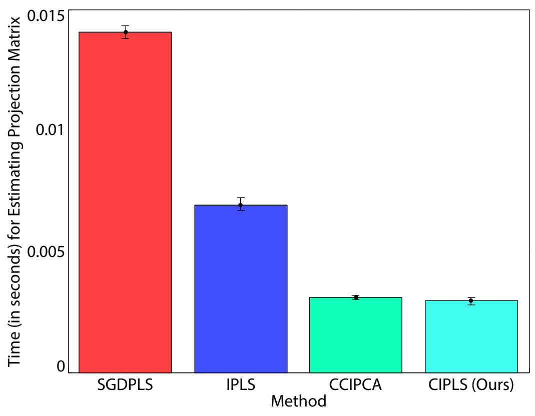


Figure 5.21. Average prediction time (in seconds) for estimating the projection matrix, lower values are better. Black bars denote the confidence interval.

and for the other parameters we use the values where the methods achieved the best results in validation. As shown in Figure 5.21, SGDPLS is the slowest incremental PLS method, which is a consequence of its strategy for estimating the projection matrix, where for each sample the convergence step is run T times. Our experiments showed that $T \geq 100$ is required for good results.

By performing the paired t-test, the time for estimating the projection matrix of our method was equivalent to CCIPCA, which is the fastest incremental dimensionality reduction. Also, our method statistically faster than IPLS and SGDPLS. Therefore, CIPLS is the fastest among the compared incremental PLS methods.

5.6.5 Incremental Methods on Streaming Scenario

As we argued before, incremental methods can be employed on streaming applications, where the training data are continuously generated. To demonstrate the robustness of our method on these scenarios, we evaluate the methods on a synthetic streaming context, as proposed by Zeng and Li [2014]. The procedure works as follows. First, the training data are divided into k blocks, where $k = 20$. The idea behind this process is to interpret each block as a new instance of arriving data. Then, we create a new training set and insert each kth block at a time. Each time we insert a new block, we learn the projection method and evaluate its accuracy on the testing set. For instance, when adding the tenth block, all the $1, 2, ...10$ blocks are being used as training. It is important to mention that a block contains more than one sample, however, this does not modify the strategy of the incremental methods, which is to estimate the projection matrix by using a single sample at a time.

Figure 5.22 (left) and (right) show the results on LFW and YTF, respectively. On

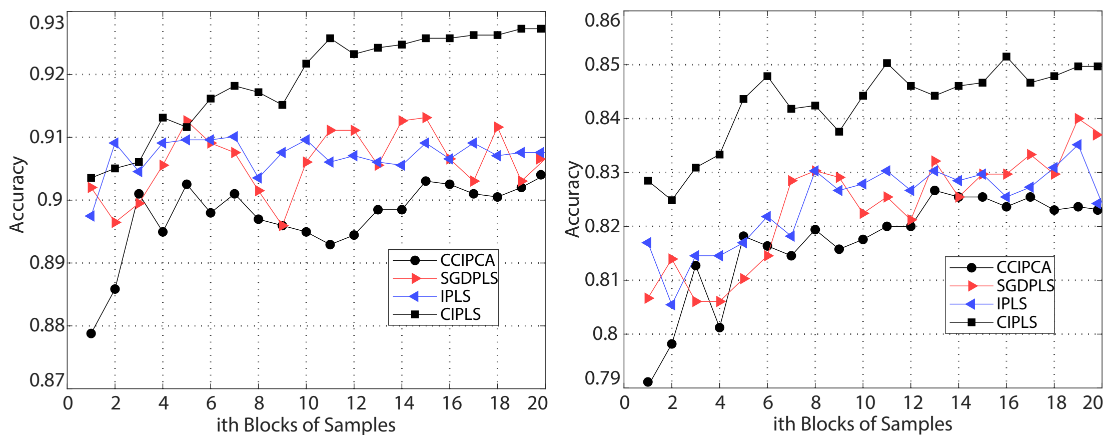


Figure 5.22. Comparison of incremental methods on a streaming scenario. **Left.** Results on Labeled Faces in the Wild (LFW). **Right.** Results on Youtube Faces (YTF). The x-axis denotes the data arriving sequentially (see the text).

the LFW dataset, until the fifth block, it is not possible to determine the best method since the accuracy presents high variance, however, from the sixth block onwards, our method outperformed all other methods. On the YTF dataset, our method achieved the highest accuracy for all blocks. These results show that the proposed method is more adequate for streaming applications than existing incremental PLS methods.

5.6.6 Comparison with Feature Selection Methods

Our last experiment evaluates the performance of CIPLS as a feature selection method. Table 5.28 shows the results for different percentages of kept features on LFW and YTF.

According to Table 5.28, CIPLS is on par with the state-of-the-art feature selection techniques. For example, on LFW the difference in accuracy, on average, from CIPLS to infFS and ilFS is of 0.15 and 0.25 p.p., respectively. Compared to infFS_S and infFS_U, this difference is 0.05 and 0.26 p.p., in this order. Interestingly, on YTF for some percentages of kept features (e.g., 15% and 50%), CIPLS outperforms ilFS, infFS_S and infFS_U. We highlight that these methods were designed specifically for feature selection.

Finally, the difference, on average, between CIPLS and PLS is of 0.26 and 0.14 p.p. on the LFW and YTF datasets, respectively. Moreover, the largest accuracy difference between PLS and CIPLS is only 0.4 p.p., on LFW with 15% of features kept. This result reinforces that the proposed decompositions to extend the NIPALS and enable the employment of VIP are a good approximation of the original method.

Based on the results shown, it is possible to conclude that, besides dimensionality

Table 5.28. Comparison of feature selection methods using different percentages of kept features. The better method is shown in bold.

	LFW			YTF		
	Percentage of Kept Features			Percentage of Kept Features		
	15	20	50	15	20	50
infFS [Roffo et al., 2015]	91.58	92.03	92.23	86.68	87.14	87.30
ilFS [Roffo et al., 2017]	91.67	92.25	92.23	**86.94**	86.84	87.54
infFS$_U$ [Roffo et al., 2020]	**91.70**	**92.30**	92.15	86.60	87.14	87.16
infFS$_S$ [Roffo et al., 2020]	91.62	91.62	92.33	86.50	86.80	87.22
PLS+VIP	91.67	92.13	**92.38**	86.82	**87.18**	**87.68**
CIPLS (Ours)+VIP	91.55	91.80	92.18	86.92	87.02	87.40

reduction, our CIPLS achieves state-of-the-art results in the context of feature selection.

5.6.7 Final Remarks

We demonstrate that is possible to extend the PLS algorithm for incremental operation and enable computation of the projection matrix using one sample at a time while still presenting the main property of traditional PLS, namely preserving the relation between dependent and independent variables. These results confirm our hypothesis that using simple algebraic decomposition it is possible to preserve the properties of traditional PLS in its incremental version. Compared to existing incremental partial least squares methods, CIPLS achieves superior performance besides being computationally efficient. In the context of feature selection, the proposed method is able to achieve comparable results to the state of the art.

Limitations. The major limitation of CIPLS is the numerical instability when employing many components. It turns out that during the algebraic decomposition (Equation 4.2) of the first components the numbers become extremely small leading to unstable computation that is propagated and accumulated to higher-order components. We observe these issues mainly when using features from deep networks, which often present high numerical precision (32 bit floating point representation). Moreover, we note that IPLS exhibited even worse numerical instability. Such issues are not restricted to CIPLS, other incremental methods also suffer from this problem [Maalouf et al., 2019; Madras et al., 2020]. In particular, even convolutional networks can present numerical instability when using large learning rates [He et al., 2019a].

Chapter 6

Conclusions

Modern convolutional networks are computationally expensive, hindering applicability on resource-constrained environments. Motivated by this, in this thesis, we proposed strategies for reducing the computational cost of convolutional architectures by removing, inserting and combining their structures. Throughout our work, we showed that Partial Least Squares is a powerful tool for measuring the importance of the structures of convolutional networks. With the estimation of this importance, we were able to identify and remove unimportant structures composing the convolutional networks. We showed that the relationship between a specific structure and its class label on a low-dimensional space (PLS criterion) can be employed to determine potential structures to be removed. Besides, by using this importance, we were able to insert structures to design high-performance convolutional networks. We demonstrated that it is possible to discover efficient architectures by inserting layers in the stages of an architecture based on their importance. In this context, we showed that estimating such importance using the PLS criterion is an effective way of determining how deep an architecture should be. Finally, with the importance estimated by PLS, we were capable of combining multiple levels of representation distributed across the network to improve data representation. We showed that an effective way of projecting multiple levels of features is to project them on a compact space provided by PLS.

The results achieved by the proposed strategies confirm our central hypothesis that the relationship between structures (specifically their outputs) and the class label, on a low-dimensional space (PLS criterion), can be effectively employed to estimate the importance of the structures composing a convolutional network. From a practical perspective, the proposed strategies promote efficient convolutional networks to a broad range of supervised computer vision tasks. In addition, they can be employed according to different hardware budgets. For example, our strategies for removing layers, designing architectures automatically and combining multiple layers are more proper to memory-constraint scenarios and large datasets, as they are more efficient to estimate the importance of the structures composing a convolutional network. When more resources (i.e., memory and time for fine-tuning) are available, our strategy for removing neurons is a clear alternative to obtain high-performance architectures since it significantly reduces the computational cost with a negligible drop in predictive ability. In particular, for some datasets, this approach reduces the computational cost while improving the predictive ability.

Despite promising results, our strategies are not without their limitations. Our method to remove neurons demands large memory consumption when applied to large datasets. This is because it generates a representation describing the neurons for each sample of a dataset, which becomes computationally expensive in terms of memory when a large number of samples is available. Our method to eliminate layers prohibits us to consider removing all layers from a convolutional network. This limitation takes place due to incompatible dimensions between the remaining layers. Our method to discover architectures does not take advantage of parallel processing. It turns out that due to its incremental essence, we cannot parallelize the neural architecture search even when more hardware resources (i.e., GPUs) are available. Our method to combine multiple layers is not suitable for transfer learning tasks, which means the LHN learned on a dataset (i.e., task) is not applicable to another dataset. The reason for this limitation is that the number of categories and the average of data can be different, which might lead to notable divergence in the latent space. Regardless of these drawbacks, our strategies promote efficient convolutional networks, facilitating the applicability of such models on resource-constrained scenarios.

Besides the drawbacks and limitations above, one major concern about our strategies is that PLS is infeasible on large datasets since it requires all the data to be in memory in advance, which is often impractical due to hardware limitations. To handle this, we proposed an incremental Partial Least Squares that learns a compact representation of the data using a single sample at a time. The results achieved by this strategy confirm our hypothesis that using simple algebraic decomposition it is possible to preserve the properties of traditional PLS in its incremental version, thus enabling applicability on large datasets while maintaining discriminability.

It is worth mentioning that, due to the nature of Partial Least Squares, all strategies developed in this thesis is limited to supervised tasks.

Throughout our research, we assessed the effectiveness of our approaches on several convolutional architectures and supervised tasks for computer vision. Our results are on par with the state of the art and, in most cases, they have the best trade-off between accuracy and computational cost. More specifically, our pruning approaches achieved a high computational cost reduction with negligible accuracy loss. Our approach to automatically design convolutional networks built high-performance architectures requiring considerably fewer computational resources than existing neural architecture search approaches. Our strategy for combining layers of convolutional networks attained the best trade-off between accuracy and computational cost. Our incremental Partial Least Squares achieved the best performance in both accuracy and complexity compared to other incremental versions of Partial Least Squares.

Future Work

The current research presented encouraging results, but many promising problems remain open to be addressed. Here, we discuss some possible future directions for each strategy of this research.

Pruning Convolutional Networks. During our investigation into removing structures of convolutional networks, we considered architectures such as plain and residual networks. Recent works have proposed more sophisticated and efficient networks, for example, the ones provided by neural architecture search [Howard et al., 2019; Dong and Yang, 2020]. It is unknown if such architectures are more sensitive to pruning or if they present a high redundancy in their structures. Thereby, we believe that removing structures from architectures designed automatically is an interesting direction for future work. In this context, similar to Cai et al. [2020], another branch for research is to combine neural architecture search and pruning, where instead of evaluating a large set of candidate architectures, we could create a dense architecture (e.g., taking into account all possible connections between layers) and, then, remove their structures.

Neural Architecture Search. To limit the search space, our neural architecture search explores only depth. However, we believe that other components, such as the number of filters and sample resolution, could also be learned stage-by-stage. Throughout our experiments, we consider two types of modules to compose the stages of the candidate architectures: residuals and cells discovered by Zoph et al. [2018]. We believe that exploring other types of modules, as well as their combinations, is another potential area for investigation.

HyperNet. While our HyperNet aims at improving data representation, we believe that multiple levels of information can aid to identify easy and hard samples. This is motivated by previous observations that several classifiers, when placed on different levels of the network, can discover hard and easy samples [Dhurandhar et al., 2020].

Once discovered hard and easy samples, it is possible to adjust the training budget based on this characteristic of the samples. For example, we can dedicate less training resources (i.e., epochs) to easy samples and make the opposite to hard samples. In this context, we believe that the proposed Latent HyperNet can be an efficient and effective way of identifying these samples.

Incremental Partial Least Squares. Our incremental Partial Least Squares (named *Covariance-free Incremental Partial Least Squares* — CIPLS) is restricted to binary problems. To operate on multi-class problems, it is necessary to learn one CIPLS model for each class following a one-versus-rest scheme. Therefore, we believe that extending CIPLS to multiclass problems is a promising line of research. Finally, despite the positive results of linear models, we believe that investigating the behavior of nonlinear (i.e., its kernel version) incremental Partial Least Squares is an interesting direction for future research.

Bibliography

Abdi, H. (2010). Partial least squares regression and projection on latent structure regression (pls regression). *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):97--106.

Alakkari, S. and Dingliana, J. (2019). An acceleration scheme for mini-batch, streaming PCA. In *British Machine Vision Conference (BMVC).*

Andrew, A. L. and Tan, R. C. E. (1998). Computation of derivatives of repeated eigenvalues and the corresponding eigenvectors of symmetric matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 20(1):78--100.

Arora, R., Mianjy, P., and Marinov, T. V. (2016). Stochastic optimization for multiview representation learning using partial least squares. In *International Conference on International Conference on Machine Learning (ICML).*

Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., and Carlsson, S. (2016). Factors of transferability for a generic convnet representation. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(9):1790--1802.

Badia, A. P., Guoand, B. P. S. K. P. S. A. V. Z., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International Conference on International Conference on Machine Learning (ICML).*

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2017). Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations (ICLR).*

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *Transactions on Information Theory*, 39(3):930--945.

Bartoldson, B., Morcos, A. S., Barbu, A., and Erlebacher, G. (2020). The generalization-stability tradeoff in neural network pruning. In *Neural Information Processing Systems (NeurIPS).*

Bell, S., Zitnick, C. L., Bala, K., and Girshick, R. B. (2016). Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Computer Vision and Pattern Recognition (CVPR).*

Bishop, C. M. (2007). *Pattern recognition and machine learning.* Springer.

Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Guttag, J. V. (2020). What is the state of neural network pruning? In *Conference on Machine Learning and Systems (MLSys).*

Brendel, W. and Bethge, M. (2019). Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations (ICLR).*

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2018). SMASH: one-shot model architecture search through hypernetworks. In *International Conference for Learning Representations(ICLR).*

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (2018). Efficient architecture search by network transformation. In *Conference on Artificial Intelligence (AAAI).*

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

Cai, W., Li, Y., and Shao, X. (2008). A variable selection method based on uninformative variable elimination for multivariate calibration of near-infrared spectra. *Chemometrics and Intelligent Laboratory Systems*, 90(2):188–194.

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep clustering for unsupervised learning of visual features. In *European Conference on Computer Vision (ECCV).*

Chatterji, N. S., Neyshabur, B., and Sedghi, H. (2020). The intriguing role of module criticality in the generalization of deep networks. In *International Conference on Learning Representations (ICLR).*

Chen, C., Jafari, R., and Kehtarnavaz, N. (2015). UTD-MHAD: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *International Conference on Image Processing (ICIP).*

Chen, T., Goodfellow, I. J., and Shlens, J. (2016). Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *International Conference on Computer Vision (ICCV)*.

Chen, Y. and Xue, Y. (2015). A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer. In *International Conference on Systems, Man, and Cybernetics*.

Chin, T., Ding, R., Zhang, C., and Marculescu, D. (2020). Towards efficient model compression via learned global ranking. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chun, S., Oh, S. J., Yun, S., Han, D., Choe, J., and Yoo, Y. (2019). An empirical evaluation on robustness and uncertainty of regularization methods. In *International Conference on International Conference on Machine Learning (ICML)*.

Cybenko, G. (1992). Approximation by superpositions of a sigmoidal function. *Mathematics ofCon- trol, Signals and Systems*, 5(4):455.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR)*.

de Geer, S. A. V. (2008). High-dimensional generalized linear models and the lasso. *The Annals of Statistics*, 36(2):614--645.

de Melo, V. H. C., Leao, S., Campos, M., Menotti, D., and Schwartz, W. R. (2013). Fast pedestrian detection based on a partial least squares cascade. In *International Conference on Image Processing (ICIP)*.

de Souza, J. S., Santos, J. B., and Schwartz, W. R. (2018). Multiscale DCNN ensemble applied to human activity recognition based on wearable sensors. In *European Signal Processing Conference (EUSIPCO)*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition (CVPR)*.

Dhurandhar, A., Shanmugam, K., and Luss, R. (2020). Enhancing simple models by exploiting what they already know. In *International Conference on Machine Learning (ICML)*.

Diniz, M. A. and Schwartz, W. R. (2020). Face attributes as cues for deep face recognition understanding. In *International Conference on Automatic Face and Gesture Recognition(FG)*.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*.

Dong, X. and Yang, Y. (2019). Searching for a robust neural architecture in four GPU hours. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dong, X. and Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*.

dos Santos, F. P. and Ponti, M. A. (2019). Alignment of local and global features from multiple layers of convolutional neural network for image classification. In *Conference on Graphics, Patterns and Images (SIBGRAPI)*.

dos Santos Junior, C. E., Kijak, E., Gravier, G., and Schwartz, W. R. (2016). Partial least squares for face hashing. *Neurocomputing*, 213:34--47.

Eger, S., Youssef, P., and Gurevych, I. (2018). Is it time to swish? comparing deep learning activation functions across NLP tasks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Simple and efficient architecture search for convolutional neural networks. In *International Conference on Learning Representations (ICLR)*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:55:1--55:21.

Evci, U., Pedregosa, F., Gomez, A. N., and Elsen, E. (2019). The difficulty of training sparse neural networks. In *International Conference on International Conference on Machine Learning (ICML)*.

Fan, A., Grave, E., and Joulin, A. (2020). Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations (ICLR)*.

Fleuret, F. (2004). Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*.

Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations (ICLR)*.

Geladi, P. and Kowalski, B. (1986). Partial least-squares regression: a tutorial. *Analytica Chimica Acta*, 185:1–17.

Ghorbani, B., Xiao, Y., and Krishnan, S. (2019). The effect of network depth on the optimization landscape. In *International Conference on International Conference on Machine Learning (ICML)*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017a). Lstm: A search space odyssey. *Transactions on Neural Networks and Learning Systems*, 28(10):2222--2232.

Greff, K., Srivastava, R. K., and Schmidhuber, J. (2017b). Highway and residual networks learn unrolled iterative estimation. In *International Conference on Learning Representations (ICLR)*.

Guo, J., Ouyang, W., and Xu, D. (2020a). Multi-dimensional pruning: A unified framework for model compression. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Guo, S., Alvarez, J. M., and Salzmann, M. (2020b). Expandnets: Linear over-parameterization to train compact convolutional networks. In *Neural Information Processing Systems (NeurIPS)*.

Gupta, S. and Tan, M. (2020). *EfficientNet-EdgeTPU: Creating Accelerator-Optimized Neural Networks with AutoML*. Accessed: 2020-07-09.

Ha, D., Dai, A. M., and Le, Q. V. (2017). Hypernetworks. In *International Conference on Learning Representations (ICLR)*.

Ha, S. and Choi, S. (2016). Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In *International Joint Conference on Neural Networks (IJCNN).*

Ha, S., Yun, J., and Choi, S. (2015). Multi-modal convolutional neural networks for activity recognition. In *International Conference on Systems, Man, and Cybernetics.*

Han, D., Kim, J., and Kim, J. (2017). Deep pyramidal residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Han, K., Wang, Y., Zhang, Q., Zhang, W., XU, C., and Zhang, T. (2020). Model rubiks cube: Twisting resolution, depth and width for tinynets. In *Neural Information Processing Systems (NeurIPS).*

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Neural Information Processing Systems (NIPS).*

Hariharan, B., Arbeláez, P. A., Girshick, R. B., and Malik, J. (2015). Hypercolumns for object segmentation and fine-grained localization. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Hasegawa, K., Miyashita, Y., and Funatsu, K. (1997). GA strategy for variable selection in QSAR studies: Ga-based PLS analysis of calcium channel antagonists. *Journal of Chemical Information and Computer Sciences*, 37(2):306--310.

Hasegawa, R. and Hotta, K. (2016). Plsnet: A simple network using partial least squares regression for image classification. In *International Conference on Pattern Recognition (ICPR).*

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR).*

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2019a). Bag of tricks for image classification with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., and Yang, Y. (2020). Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. (2018a). Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI).*

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. (2018b). Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV).*

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019b). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

He, Y., Zhang, X., and Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV).*

Hendrycks, D. and Gimpel, K. (2017). Early methods for detecting adversarial images. In *International Conference on Learning Representations (ICLR).*

Hendrycks, D., Lee, K., and Mazeika, M. (2019). Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning (ICML).*

Hiraoka, K., Yoshizawa, S., Hidai, K., Hamahira, M., Mizoguchi, H., and Mishima, T. (2000). Convergence analysis of online linear discriminant analysis. In *International Joint Conference on Neural Network (IJCNN).*

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251--257.

Hornik, K., Stinchcombe, M. B., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359--366.

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

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *In arXiv.*

Hu, H., Peng, R., Tai, Y., and Tang, C. (2016). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *In arXiv.*

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Huang, G., Liu, Z., Pleiss, G., Van Der Maaten, L., and Weinberger, K. (2019). Convolutional networks with dense connectivity. *Transactions on Pattern Analysis and Machine Intelligence (PAMI).*

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

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

Huang, G. B., Mattar, M. A., Lee, H., and Learned-Miller, E. G. (2012). Learning to align from scratch. In *Neural Information Processing Systems (NIPS).*

Huang, Q., Zhou, S. K., You, S., and Neumann, U. (2018). Learning to prune filters in convolutional neural networks. In *Winter Conference on Applications of Computer Vision (WACV).*

Huang, Z. and Wang, N. (2018). Data-driven sparse structure selection for deep neural networks. In *European Conference on Computer Vision (ECCV).*

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML).*

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* Wiley professional computing. John Wiley & Sons.

Jin, H., Song, Q., and Hu, X. (2019). Auto-keras: An efficient neural architecture search system. In *International Conference on Knowledge Discovery & Data Mining (SIGKDD).*

Kandasamy, K., Neiswanger, W., Schneider, J., Póczos, B., and Xing, E. P. (2018). Neural architecture search with bayesian optimisation and optimal transport. In *Neural Information Processing Systems (NeurIPS).*

Ke, Q., Bennamoun, M., An, S., Sohel, F. A., and Boussaïd, F. (2017). A new representation of skeleton sequences for 3d action recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Kloss, R. B., Jordão, A., and Schwartz, W. R. (2017). Boosted projection: An ensemble of transformation models. In *Iberoamerican Congress on Pattern Recognition (CIARP)*.

Kloss, R. B., Jordao, A., and Schwartz, W. R. (2018). Face verification strategies for employing deep models. In *International Conference on Automatic Face & Gesture Recognition (FG)*.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In *European Conference on Computer Vision (ECCV)*.

Kong, T., Yao, A., Chen, Y., and Sun, F. (2016). Hypernet: Towards accurate region proposal generation and joint object detection. In *Computer Vision and Pattern Recognition (CVPR)*.

Kornblith, S., Shlens, J., and Le, Q. V. (2019). Do better imagenet models transfer better? In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Krizhevsky, A., Nair, V., and Hinton, G. (2009). Cifar-10 (canadian institute for advanced research).

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems (NIPS)*.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86.

Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J. R. R., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T., and Ferrari, V. (2020). The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision (to appear)*.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. In *Neural Information Processing Systems (NeurIPS)*.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541--551.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. *International Conference for Learning Representations(ICLR).*

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Neural Information Processing Systems (NeurIPS).*

Li, J., Qi, Q., Wang, J., Ge, C., Li, Y., Yue, Z., and Sun, H. (2019a). OICSR: out-in-channel sparsity regularization for compact deep neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Li, M., Yumer, E., and Ramanan, D. (2020a). Budgeted training: Rethinking deep neural network training under resource constraints. In *International Conference on Learning Representations (ICLR).*

Li, X., Wang, W., Hu, X., and Yang, J. (2019b). Selective kernel networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Li, Y., Yang, M., and Zhang, Z. (2019c). A survey of multi-view representation learning. *Transactions on Knowledge and Data Engineering*, 31(10):1863--1883.

Li, Z., Xi, T., Deng, J., Zhang, G., Wen, S., and He, R. (2020b). GP-NAS: gaussian process based neural architecture search. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. (2020). Hrank: Filter pruning using high-rank feature map. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Liu, H., Simonyan, K., and Yang, Y. (2019a). DARTS: differentiable architecture search. In *International Conference on Learning Representations (ICLR).*

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision (ICCV).*

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019b). Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR).*

Lockhart, J. W., Weiss, G. M., Xue, J. C., Gallagher, S. T., Grosner, A. B., and Pulickal, T. T. (2011). Design considerations for the wisdm smart phone-based sensor mining architecture. In *International Workshop on Knowledge Discovery from Sensor Data.*

Loshchilov, I. and Hutter, F. (2017). SGDR: stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*.

Lu, G., Zou, J., and Wang, Y. (2012). Incremental learning of complete linear discriminant analysis for face recognition. *Knowledge-Based Systems*, 31:19--27.

Lu, J. and Tong, K. (2019). Robust single accelerometer-based activity recognition using modified recurrence plot. *IEEE Sensors Journal*, 19(15):6317–6324.

Luo, J., Zhang, H., Zhou, H., Xie, C., Wu, J., and Lin, W. (2019). Thinet: Pruning CNN filters for a thinner net. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 41(10):2525--2538.

Luo, J.-H. and Wu, J. (2020). Neural network pruning with residual-connections and limited-data. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Maalouf, A., Jubran, I., and Feldman, D. (2019). Fast and accurate least-mean-squares solvers. In *Neural Information Processing Systems (NeurIPS)*.

Mackey, L. W. (2008). Deflation methods for sparse PCA. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Neural Information Processing Systems (NIPS)*.

Madras, D., Atwood, J., and D'Amour, A. (2020). Detecting extrapolation with local ensembles. In *International Conference on Learning Representations (ICLR)*.

Martínez, A. M. and Kak, A. C. (2001). PCA versus LDA. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(2):228--233.

Mehmood, T., Liland, K. H., Snipen, L., and Saebo, S. (2012). A review of variable selection methods in partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2020). Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations (ICLR)*.

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *British Machine Vision Conference (BMVC)*.

Ramachandran, P., Zoph, B., and Le, Q. V. (2018). Searching for activation functions. In *International Conference on Learning Representations (ICLR)*.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: An astounding baseline for recognition. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPR)*.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *International Conference on International Conference on Machine Learning (ICML)*.

Renda, A., Frankle, J., and Carbin, M. (2020). Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations (ICLR)*.

Roffo, G. and Melzi, S. (2016a). Online feature selection for visual tracking. In Wilson, R. C., Hancock, E. R., and Smith, W. A. P., editors, *British Machine Vision Conference (BMVC)*.

Roffo, G. and Melzi, S. (2016b). Ranking to learn: Feature ranking and selection via eigenvector centrality. In Appice, A., Ceci, M., Loglisci, C., Masciari, E., and Ras, Z. W., editors, *New Frontiers in Mining Complex Patterns (NFMCP)*.

Roffo, G., Melzi, S., Castellani, U., and Vinciarelli, A. (2017). Infinite latent feature selection: A probabilistic latent graph-based ranking approach. In *International Conference on Computer Vision (ICCV)*.

Roffo, G., Melzi, S., Castellani, U., Vinciarelli, A., and Cristani, M. (2020). Infinite feature selection: a graph-based feature filtering approach. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

Roffo, G., Melzi, S., and Cristani, M. (2015). Infinite feature selection. In *International Conference on Computer Vision (ICCV)*.

Rooyen, B. V., Menon, A. K., and Williamson, R. C. (2015). Learning with symmetric label noise: The importance of being unhinged. In *Neural Information Processing Systems (NIPS)*.

Rosenfeld, J. S., Rosenfeld, A., Belinkov, Y., and Shavit, N. (2020). A constructive prediction of the generalization error across scales. In *International Conference on Learning Representations (ICLR)*.

Rueda, F. M., Grzeszick, R., Fink, G. A., Feldhorst, S., and ten Hompel, M. (2018). Convolutional neural networks for human activity recognition using body-worn sensors. *Informatics*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:696–699.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*.

Sankararaman, K. A., De, S., Xu, Z., Huang, W. R., and Goldstein, T. (2020). The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *International Conference on International Conference on Machine Learning (ICML)*.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2018). How does batch normalization help optimization? In *Neural Information Processing Systems (NeurIPS)*.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12):54--63.

Schwartz, W. R., Kembhavi, A., Harwood, D., and Davis, L. S. (2009). Human detection using partial least squares analysis. In *International Conference on Computer Vision (ICCV)*.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. (2020). Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations (ICLR)*.

Shafahi, A., Saadatpanah, P., Zhu, C., Ghiasi, A., Studer, C., Jacobs, D. W., and Goldstein, T. (2020). Adversarially robust transfer learning. In *International Conference on Learning Representations (ICLR)*.

Shafiee, M. S., Shafiee, M. J., and Wong, A. (2019). Dynamic representations toward efficient inference on deep neural networks by decision gates. In *Conference on Computer Vision and Pattern Recognition Workshops(CVPR)*.

Sharma, A. and Jacobs, D. W. (2011). Bypassing synthesis: PLS for face recognition with pose, low-resolution and sketch. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.

Sindagi, V. and Patel, V. M. (2019). Multi-level bottom-top and top-bottom feature fusion for crowd counting. In *International Conference on Computer Vision (ICCV)*.

Singh, S. and Shrivastava, A. (2019). Evalnorm: Estimating batch normalization statistics for evaluation. In *International Conference on Computer Vision (ICCV)*.

Smith, S. L., Kindermans, P., Ying, C., and Le, Q. V. (2018). Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations (ICLR)*.

Song, H., Thiagarajan, J. J., Sattigeri, P., Ramamurthy, K. N., and Spanias, A. (2017). A deep learning approach to multiple kernel fusion. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Stott, A. E., Kanna, S., Mandic, D. P., and Pike, W. T. (2017). An online NIPALS algorithm for partial least squares. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Conference of the Association for Computational Linguistics (ACL)*.

Suau, X., Zappella, L., and Apostoloff, N. (2020). Filter distillation for network compression. In *Winter Conference on Applications of Computer Vision (WACV)*.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision (ICCV)*.

Tan, C. M. J. and Motani, M. (2020). Dropnet: Reducing neural network complexity via iterative pruning. In *International Conference on International Conference on Machine Learning (ICML)*.

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

Vahdat, A., Mallya, A., Liu, M., and Kautz, J. (2020). UNAS: differentiable architecture search meets reinforcement learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Vareto, R., Silva, S., Costa, F., and Schwartz, W. R. (2017a). Towards open-set face recognition using hashing functions. In *International Joint Conference on Biometrics (IJCB)*.

Vareto, R. H. and Schwartz, W. R. (2020). Unconstrained face identification using ensembles trained on clustered data. In *International Joint Conference on Biometrics (IJCB)*.

Vareto, R. H., Silva, S. S. D., de Oliveira Costa, F., and Schwartz, W. R. (2017b). Face verification based on relational disparity features and partial least squares models. In *Conference on Graphics, Patterns and Images, (SIBGRAPI)*.

Veit, A. and Belongie, S. J. (2020). Convolutional networks with adaptive inference graphs. *International Journal of Computer Vision (IJCV)*, 128(3):730--741.

Veit, A., Wilber, M. J., and Belongie, S. J. (2016). Residual networks behave like ensembles of relatively shallow networks. In *Neural Information Processing Systems (NIPS)*.

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

Wang, C., Zhang, G., and Grosse, R. B. (2020). Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR)*.

Wang, R. J., Li, X., and Ling, C. X. (2018a). Pelee: A real-time object detection system on mobile devices. In *Neural Information Processing Systems (NeurIPS)*.

Wang, X., Yu, F., Dou, Z., Darrell, T., and Gonzalez, J. E. (2018b). Skipnet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision (ECCV)*.

Weng, J., Zhang, Y., and Hwang, W. (2003). Candid covariance-free incremental principal component analysis. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(8):1034--1040.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2017). Automatic frankensteining: Creating complex ensembles autonomously. In Chawla, N. V. and Wang, W., editors, *International Conference on Data Mining (ICDM)*.

Wolf, L., Hassner, T., and Maoz, I. (2011). Face recognition in unconstrained videos with matched background similarity. In *Computer Vision and Pattern Recognition (CVPR)*.

Wu, Y. and He, K. (2018). Group normalization. In *European Conference on Computer Vision (ECCV).*

Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K., and Feris, R. S. (2018). Blockdrop: Dynamic inference paths in residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Xie, C. and Yuille, A. L. (2020). Intriguing properties of adversarial training at scale. In *International Conference on Learning Representations (ICLR).*

Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR).*

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *In arXiv.*

Xu, W., Pang, Y., Yang, Y., and Liu, Y. (2018). Human activity recognition based on convolutional neural network. In *International Conference on Pattern Recognition (ICPR).*

Xu, Z. and Li, P. (2019). Towards practical alternating least-squares for CCA. In *Neural Information Processing Systems (NeurIPS).*

Yan, J., Wan, R., Zhang, X., Zhang, W., Wei, Y., and Sun, J. (2020). Towards stabilizing batch statistics in backward propagation of batch normalization. In *International Conference on Learning Representations (ICLR).*

Yang, H. H. and Moody, J. E. (1999). Data visualization and feature selection: New algorithms for nongaussian data. In Solla, S. A., Leen, T. K., and Müller, K., editors, *Neural Information Processing Systems (NIPS).*

Yang, L., Han, Y., Chen, X., Song, S., Dai, J., and Huang, G. (2020a). Resolution adaptive networks for efficient inference. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Yang, L. and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295--316.

Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., and Xu, C. (2020b). CARS: continuous evolution for efficient neural architecture search. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Neural Information Processing Systems (NIPS)*.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. (2018). NISP: pruning networks using neuron importance score propagation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *British Machine Vision Conference (BMVC)*.

Zeng, X. and Li, G. (2014). Incremental partial least squares analysis of big streaming data. *Pattern Recognition*, 47:3726--3735.

Zhang, C., Bengio, S., and Singer, Y. (2019). Are all layers created equal? In *International Conference on International Conference on Machine Learning (ICML)*.

Zhang, M. and Sawchuk, A. A. (2012). Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *ACM Conference on Ubiquitous Computing*.

Zhou, W., Xu, C., Ge, T., McAuley, J. J., Xu, K., and Wei, F. (2020). BERT loses patience: Fast and robust inference with early exit. In *Neural Information Processing Systems (NeurIPS)*.

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

Appendix A

Time for Training/Fine-tuning

Table A.1 and A.2 show the time (in hours) for training/fine-tuning the architectures employed in our research. For the ImageNet dataset, Table A.1, this time was computed with the data stored in a Solid State Drive (SSD). For the CIFAR-10 dataset, Table A.2, this time was computed with all the data in memory.

Table A.1. Time for training/fine-tuning different architectures on ImageNet 224×224.

Architecture	Time (hours) for Fine-Tuning (1 epoch)
VGG16	3.63
MobileNetV1	3.67
MobileNetV2	4.52
ResNet50	5.34

Table A.2. Time for training/fine-tuning different architectures on CIFAR-10.

Architecture	Time (hours) for Fine-Tuning (1 epoch)
VGG16	0.0283
MobileNetV1	0.0250
MobileNetV2	0.0402
ResNet56	0.0458
ResNet110	0.0827

Appendix B

Pruning Structures

Current deep learning frameworks implement the convolutional operation as tensor operations to take advantage of parallel processing of the GPUs. For this purpose, a layer i store its input (the output of $i-1$ layer) and output tensor. Below we explain how these tensor operations influence in removing filters and layers.

Pruning Filters. When pruning filters from the layer i, we need to remove the connections of tensors and ensure that the dimensions will match. Such a requirement prohibits us to prune some layers, as illustrated in Figure B.1. In this figure, blue boxes indicate the layers we can prune and the red box indicates the layer we cannot.

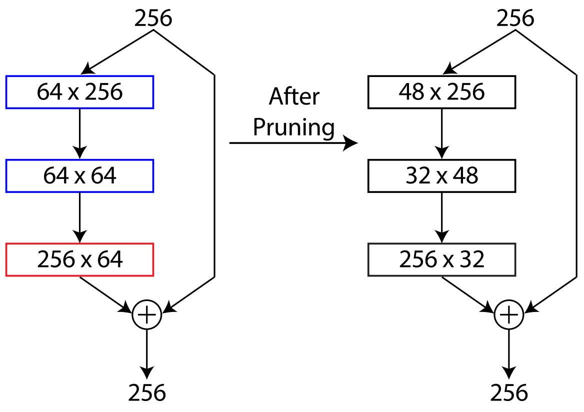


Figure B.1. Left. Layers before removing filters. **Right.** Layers after removing filters. In both figures, left and right values in each box represent the output (i..e, the number of filters composing the layer) and input tensor, in this order.

To the layers that we can prune, we need to remove the filters of tensors of input and output. For example, in Figure B.1, we remove 16 and 32 filters from the first and second layers, respectively. Note that, when removing such filters, the next $(i+1)$ layer had its input tensor changed. Finally, we are not able to prune the last layer because it will generate an incompatible match in the element-wise addition operation (represented by '+'). For example, let us consider we remove 56 filters from this layer. It turns out that we would have 200 filters in this layer and the element-wise will fail because it expects an operation between two tensors with 256 filters each.

Pruning Layers. The process to remove layers consists in connecting the output tensor of a layer i to the input tensor of a layer $i+j$. For example, Figure B.2 illustrates a single module of ResNet50 (top leftmost) and the removal of the first (case 1), second (case 2) and third (case 3) layer of this module. In this figure, the cases 1 and 3 generate incompatible dimensions between the input and output tensor; thus we cannot remove such layers. In practice, considering this module, we can remove only the second layer (case 2). Following this implementation, to a convolutional architecture with six modules we can remove only six layers. Instead, we can prune entire modules. To this end, we need to connect the output tensor of a module to the input tensor of another module, as shown in Figure B.2 (bottom). In this implementation, to a convolutional architecture with six modules, we can remove up to 12 layers (4 modules with 3 layers each). It is important to mention that the first and last module of the stage cannot be removed due to incompatible dimensions of tensors. Thus, given a

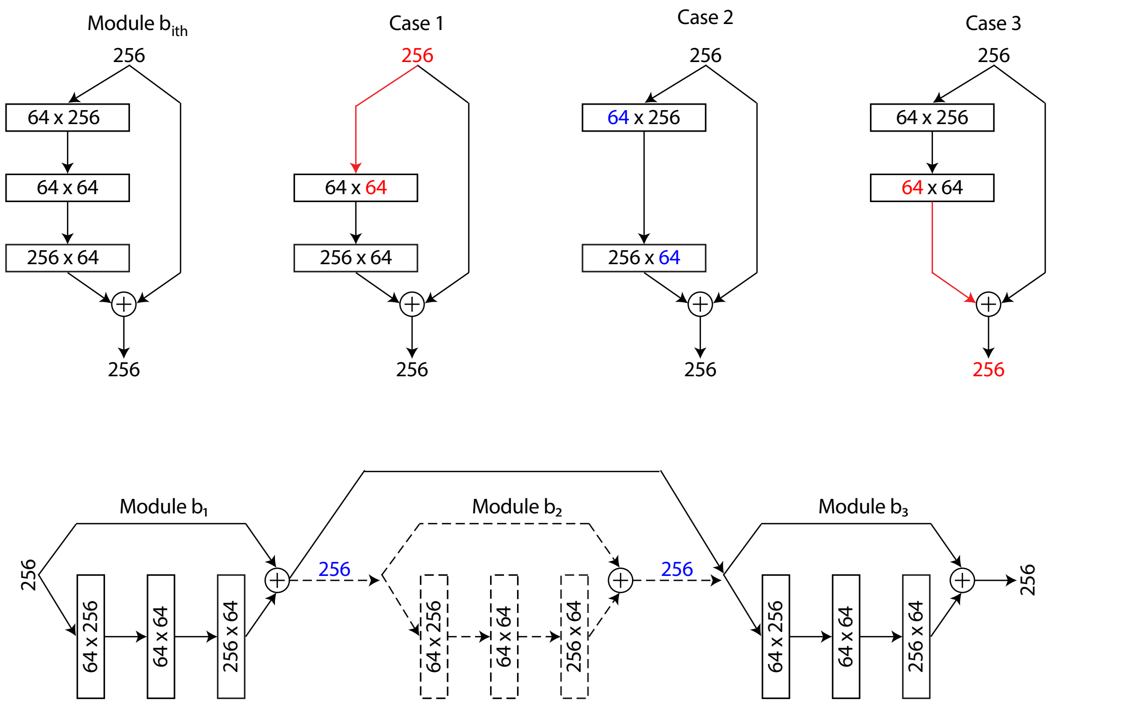


Figure B.2. **Top.** Module before removing layers (leftmost) and possible layers to be removed. Red arrows indicate the is impossible to execute the removal. **Bottom.** Example of pruning considering entire modules instead of layers. In this example module b_2 was removed. To this end, we connect the output of the module b_1 to the input of module b_3.

stage of k modules, we can remove at most $k - 2$ modules. Finally, this limitation restricted us to remove only five modules of MobileNetV2.

Appendix C

Neural Architecture Search

Tables C.1 and C.2 show our candidate architectures considering residual and cells by Zoph et al. [2018] as modules, respectively.

Table C.1. Performance of our discovered architectures considering residual modules. W. transfer indicates our weight transfer mechanism. The best accuracy is shown in bold. The arrows indicate which direction is better.

Iteration	Training Strategy	Depth	Param.↓ (Million)	FLOPs↓ (Million)	Memory↓ (MB)	Accuracy↑ (200 epochs)
1	Scratch	43	0.60	92	7.41	92.03
	W. transfer	39	0.56	83	7.00	92.88
2	Scratch	51	0.65	111	8.88	92.38
	W. transfer	43	0.71	92	7.61	92.64
3	Scratch	59	0.69	130	10.32	92.62
	W. transfer	51	0.90	111	9.16	**92.92**
4	Scratch	63	0.84	139	11.09	92.53
	W. transfer	51	1.08	130	10.52	92.76
5	Scratch	67	0.88	149	11.65	92.58
	W. transfer	59	1.23	139	11.31	92.39

Table C.2. Performance of our discovered architectures considering cell modules. The best accuracy is shown in bold. The arrows indicate which direction is better.

Iteration	Depth	Param.↓ (Million)	FLOPs↓ (Billion)	Memory↓ (MB)	Accuracy↑ (200 epochs)
1	109	1.3	1.8	45.20	92.06
2	121	1.4	2.0	52.81	91.93
3	133	1.5	2.2	56.96	92.03
4	157	1.9	2.5	67.92	92.60
5	181	2.3	2.9	78.87	**92.78**