Sparse, Hierarchical and Shared-Factors Priors for Representation Learning

Thèse

Ludovic Trottier

Doctorat en informatique
Philosophiæ doctor (Ph. D.)

Québec, Canada

Sparse, Hierarchical and Shared-Factors Priors for Representation Learning

Thèse

Ludovic Trottier

Sous la direction de:

Philippe Giguère, directeur de recherche
Brahim Chaib-draa, codirecteur de recherche

Résumé

La représentation en caractéristiques est une préoccupation centrale des systèmes d'apprentissage automatique d'aujourd'hui. Une représentation adéquate peut faciliter une tâche d'apprentissage complexe. C'est le cas lorsque par exemple cette représentation est de faible dimensionnalité et est constituée de caractéristiques de haut niveau. Mais comment déterminer si une représentation est adéquate pour une tâche d'apprentissage ? Les récents travaux suggèrent qu'il est préférable de voir le choix de la représentation comme un problème d'apprentissage en soi. C'est ce que l'on nomme l'apprentissage de représentation.

Cette thèse présente une série de contributions visant à améliorer la qualité des représentations apprises. La première contribution élabore une étude comparative des approches par dictionnaire parcimonieux sur le problème de la localisation de points de prises (pour la saisie robotisée) et fournit une analyse empirique de leurs avantages et leurs inconvénients. La deuxième contribution propose une architecture réseau de neurones à convolution (CNN) pour la détection de points de prise et la compare aux approches d'apprentissage par dictionnaire. Ensuite, la troisième contribution élabore une nouvelle fonction d'activation paramétrique et la valide expérimentalement. Finalement, la quatrième contribution détaille un nouveau mécanisme de partage souple de paramètres dans un cadre d'apprentissage multitâche.

Abstract

Feature representation is a central concern of today's machine learning systems. A proper representation can facilitate a complex learning task. This is the case when for instance the representation has low dimensionality and consists of high-level characteristics. But how can we determine if a representation is adequate for a learning task? Recent work suggests that it is better to see the choice of representation as a learning problem in itself. This is called Representation Learning.

This thesis presents a series of contributions aimed at improving the quality of the learned representations. The first contribution elaborates a comparative study of Sparse Dictionary Learning (SDL) approaches on the problem of grasp detection (for robotic grasping) and provides an empirical analysis of their advantages and disadvantages. The second contribution proposes a Convolutional Neural Network (CNN) architecture for grasp detection and compares it to SDL. Then, the third contribution elaborates a new parametric activation function and validates it experimentally. Finally, the fourth contribution details a new soft parameter sharing mechanism for multitasking learning.

Contents

List of Tables

List of Figures

List of Algorithms

À mes parents.

Per aspera ad astra.

Remerciements

Il va sans dire que mes premiers remerciements reviennent à mon directeur de recherche Philippe Giguère et à mon codirecteur de recherche Brahim Chaib-draa.

Merci Philippe de m'avoir pris sous ton aile alors que toi-même tu débutais ton nouveau poste de professeur. Tu as été un directeur, professeur et mentor dévoué envers moi et tu n'as jamais hésité à investir ton temps et ton énergie pour me guider dans mes recherches. Je suis reconnaissant de l'aide et du support que tu m'as offert, autant moral, technique que financier, et de m'avoir aidé à m'épanouir tout au long de ces dernières années.

Merci Brahim d'avoir cru en mon potentiel et de m'avoir donné une place au DAMAS au tout début de mon baccalauréat. Tu m'as donné l'opportunité de m'immerger dans le domaine de la recherche universitaire en apprentissage automatique, ce qui constituait pour moi tout un privilège. Tu m'as aussi inculqué le rôle et les responsabilités de la recherche scientifique et m'as permis de grandir en tant que chercheur.

J'aimerais également remercier tous mes collègues au DAMAS, anciens comme nouveaux.

D'abord merci Patrick Dallaire. J'ai savouré toutes les discussions en Bayésien non-paramétrique, en apprentissage automatique et sur la science en général que nous avons eues. J'ai vécu l'expérience incroyable de construire de toutes pièces une nouvelle distribution de probabilité. C'est pas quelque chose à laquelle on assiste à tous les jours, alors je t'en remercie.

Merci Jean-Philippe Mercier. Grâce à toi, j'ai eu l'opportunité de réaliser un de mes souhaits de chercheur lors de notre collaboration. Tu m'as fait l'honneur d'appliquer mes travaux de recherche sur ton problème de grasping. Ce fut une grande étape dans mon cheminement, car je voyais concrètement mes efforts récompensés.

Merci Alexandre Gariépy pour m'avoir initié à Arch Linux, Vim et Spacemacs. Tu m'as également permis de revivre à travers tes yeux la découverte du merveilleux domaine de l'apprentissage automatique et du *deep learning*. Merci Hamid Chinaei pour m'avoir aidé avec la reconnaissance vocale au tout début lorsque je débutais à la maîtrise. Merci Alejandro Sanchez qui m'a fait découvrir et aimer Python, et qui a été un partenaire impeccable pour les travaux personnels.

Sans vous, mon expérience au DAMAS aurait été bien différente !

Je veux également souligner toutes les personnes avec qui j'ai eu de belles discussions. Entre autres, Ming Hou, Mathieu Carpentier, David Landry, Jonathan Gingras, Maxime Latulippe, Jean-François Tremblay, Philippe Babin, Marc-André Gardner, Yannick Hold-Geoffroy, Jean-Francois Lalonde, Amar Alibey, Sébastien Michaud et Alexandre Lemire-Paquin.

Merci à tous les professeurs qui ont été marquant lors de mes études. Merci Christian Gagné, Mario Marchand, Claude Bolduc, Thierry Eude, Claude-Guy Quimper, Fançois Laviolette, Hugo Larochelle, Béchir Ktari et Pascal Tesson.

Et enfin, je veux dire merci à ma mère Maude Gagnon, mon père Marc Trottier, mon frère Olivier Trottier et ma sœur Mireille Trottier, ainsi qu'à ceux de la famille éloignée, autant chez les Trottier, Gagnon, Gaudreault, Chagnon, Porter, Ferland et Brunelle. Je suis extrêmement privilégié d'avoir un noyau familial qui me permet de grandir et de m'épanouir. Vous m'avez donné tous les outils nécessaires pour construire mon avenir et m'établir comme personne. On ne choisit pas les membres de sa famille lorsqu'on vient au monde, mais si j'avais eu à le faire, je vous aurais choisi sans hésitation.

Introduction

The history of mankind has thought us that our civilization has been shaped by many technological revolutions. These revolutions were propelled by seemingly different factors, but most of them revolved around our desire for *automation*. For instance, the advent of the modern computer made possible the creation of automated manufacturing processes (Hitomi, 1994) that accelerated manufacturing speed. The advent of the Internet pushed forward the field of communication to a point where we can connect devices worldwide (Kim, 2005). The modern cell phone has evolved considerably such that we now consider it as a Personal Data Assistant (PDA), which functions as a personal automated information manager.

We have gone through several such revolutions throughout the years, but the current one is different in an important aspect: it is propelled by Artificial Intelligence (AI). The previous one was concerned with digitalization, where we understood the importance of shifting from mechanical and analogue electronic technology to digital electronics. The AI revolution of today rather focuses on developing *intelligent automation*, where we try to develop intelligent components with abilities to learn and adapt. We are seeing more and more of this desire with the advent of the Internet of Things (IoT) (Shah and Yaqoob, 2016), the *smart factory* of Industry 4.0 (Chen et al., 2018a), AI as a Service (AIaaS) (Rouse, 2018), as well as Software 2.0 (Karpathy, 2018), where programmers of today are seen as data scientists rather than software engineers.

Despite the fact that these novelties are propelled by intelligent automation, *intelligence* still remains an abstract and obscure notion. What makes intelligent automation possible is the fact that AI relies on specific instantiations driven by theoretically-founded principles, rather than our abstract definitions of intelligent behaviors. In particular, it relies on a learning theory that evolved from the study of pattern recognition and statistics, which is known as *Machine Learning* (Bishop, 2006).

Machine Learning is a field of computer science dedicated to the study of algorithms that

have the ability to learn and make predictions on data. A specific branch of machine learning is interested in the problem of learning a parametric mapping from input observation to output values. In this specific case, training a machine learning approach comes down to optimizing the parameters of the mapping. The goal is to improve the parametric mapping so that the output value predicted by the algorithm corresponds to the output value (ground truth) associated with the given input observation.

Many learning problems can be defined is such a way. In particular, problems related to real-world images. These problems are usually challenging because of the inherent difficulty of processing high-dimensional images. As a first step to tackle this problem, it is common to apply a preprocessing transformation on the input image. The goal is to reduce its dimensionality to make the learning problem easier.

Various preprocessing transformations, known as feature extraction pipelines, have been proposed to accomplish this task. They usually come in the form of a series of hand-designed input-output mappings that transform the high-dimensional input image into low-dimensional features (Liu et al., 2004; Nixon and Aguado, 2012). They aim to bring out the important underlying factors of variation from the data (Choras, 2007), while dismissing the unimportant ones (Hyvärinen et al., 2004; Jolliffe, 2011). When the design of the preprocessing is appropriate for a given task, it can extract informative high-level features.

The success of these hand-designed approaches can however be limited (Bengio et al., 2009). The engineering of preprocessing pipelines is based on human inventiveness, and they can struggle to extract good representations on complex tasks. When an expert tries to leverage his expertise, he can incorporate restricting priors – often unwillingly – about the data generating process. These priors appear in various forms, such as assumptions, approximations or fixed numerical values, and generally do not appear as restrictive. They make the preprocessing pipeline simple and elegant, but in return, they overshadow the restrictions they impose on the representation computed from the data (Bengio et al., 2009).

More and more experts are now considering the use of a learning algorithm for feature representation (Bengio et al., 2013). Instead of investing efforts on designing better feature engineering pipelines, they now invest efforts on designing better *representation learning* algorithms. Representation learning algorithms adapt to the data distribution, which make them more flexible than feature engineering pipelines. A typical feature engineering approach would process the input image using a fixed sequence of trans-

formations. These transformations are predetermined in advance and have little to no flexibility. In contrast, representation learning has a learning phase during which they try to uncover the intricacy of the structure underlying the data. They can learn to capture the important aspects of the observations and generate features to represent them (Bengio et al., 2013).

However, representation learning can also be limited by expert knowledge, like feature engineering. A good understanding of the data generating process may no longer be as critical, but there are design choices that affect the overall quality of the computed representation. Experts designing representation learning approaches still need to follow guiding principles that also appear in the form of priors. Whether they be about properties of the extracted representations, the design of the learning approach, or other considerations like computational speed or hardware limitations, they play different roles on the ability to perform representation learning.

In this thesis, our aim is to better understand the effect of these priors by studying representation learning approaches applied to images. With novel contributions and in-depth empirical evaluations, we will attempt to deepen our understanding and broaden the range of application of representation learning.

Motivating Example

The choice of feature representation is a central concern in machine learning. A good feature representation can leverage simple machine learning models, such as the linear model. A linear model expresses the target output value y as a linear combination of the input features x. It is defined as $y = \theta^\top x$, where θ is the vector of parameters. A linear model can only make simple predictions from its input x and will struggle when x exhibits nonlinear dependencies. For instance, on classification problems, if we provide input features where classes are not linearly separable, a linear model will fail. However, if we provide input features where classes are linearly separable, then the linear model will be as accurate as a nonlinear model.

The example shown in Figure 1 illustrates that point. On the left of the figure, we show a binary classification problem. We generate observations on the unit circle by varying the angle. Observations on the top right and bottom left quadrant form the first class, and observations on the top left and bottom right quadrant form the second class. The problem is not linearly separable and suggests that we should opt for a

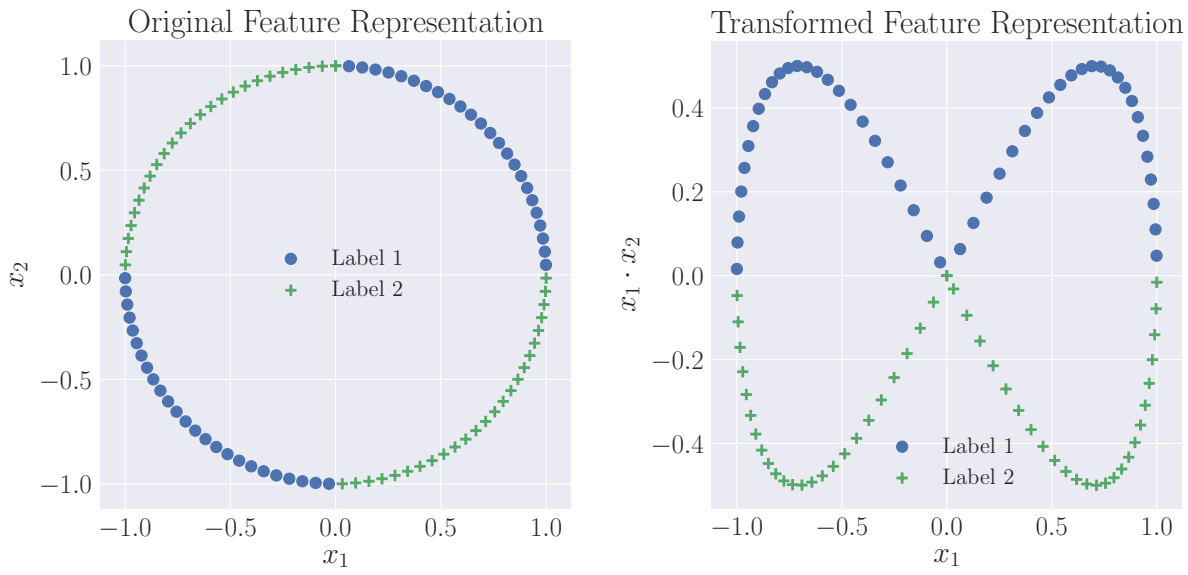


Figure 1: Example motivating the importance of feature representation. The left figure shows a classification problem that is not linearly separable. To obtain good performance, we would need a complex nonlinear classifier. To alleviate this constraint, we can apply the transformation $(x_1, x_2) \mapsto (x_1, x_1 \cdot x_2)$ and obtain a linearly separable classification problem, as shown on the right figure. A simple linear model can now obtain a high accuracy. This example illustrates the importance of leveraging input transformation to create a new feature representation, in which the problem becomes easier.

nonlinear classifier to be accurate. On the right figure, we show the same classification problem, but this time using a new feature representation. We applied the transformation $\Phi : [x_1, x_2] \mapsto [x_1, x_1 \cdot x_2]$ and created a representation $\phi = [x_1, x_1 \cdot x_2]$ in which the classes are linearly separable. The linear model $y = \theta^\top \phi$ can now perform in par with the nonlinear classifier.

Although the example shown in Figure 1 is simple, it can help us understand the importance of feature representation on harder tasks. High-dimensional multi-class classification tasks, for instance, have inputs x with more nonlinear dependencies, and more complex ones. These additional dependencies make the design of transformation Φ challenging, since now Φ must use a series of elaborated and complex transformations to reflect the increase in complexity of the data generating process. It then becomes difficult to select the proper transformations to capture the relevant factors of variation. In the end, the generated feature representation is likely to be incompatible with the linear model.

Representation learning can overcome many of these difficulties. A representation learn-

ing approach can learn from the data the proper mapping to apply to x, and transform x into a new representation ϕ compatible with a linear model. It can make the initial learning problem easier.

Contributions

The key contributions of this thesis are divided into four parts. The first two contributions are more of an experimental nature, while the last two contributions are more of a theoretical nature. The contributions are as follows:

A comparative study of sparse dictionary learning on the problem of grasp localization (Chapter 5)

We present in this contribution a comparative study of sparse dictionary learning, which is a framework for representation learning that focuses on sparsity. We perform this study on the problem of grasp localization. Given the image of an object, the goal is to identify where to position a two-plates parallel gripper, in order to maximize the grasp success probability. To do so, we use the grasp rectangle, parameterized by a center, width, height and angle, to represent the physical dimensions of the gripper in image space. We use the rectangle to extract the image of the graspable region, and train a sparse dictionary learning approach in an unsupervised manner. The trained approach is used to transform the image into a feature vector, which is fed to a linear support vector machine classifier.

Using the Cornell dataset (Jiang et al., 2011), we evaluated a total of thirty-six sparse dictionary learning approaches on the problem of rectangle classification, and keep the five best ones for the problem of rectangle detection. Results show that the ability of sparse dictionary learning to learn good representations on noisy data helps to obtain better performance than the previously proposed neural network approaches. This also allowed us to better understand the advantages and disadvantages of neural networks, as well as provide us with more insight on certain limitations of the Cornell dataset.

A residual network for grasp localization (Chapter 5)

As second contribution, we extended the network architecture of Redmon and Angelova (2015) for grasp detection to include residual learning (He et al., 2016a,b). The network is designed to perform multitask learning, where one head produces a confidence map indicating the grasp success probability in forty-nine distinct regions, while the

second one predicts one grasp rectangle in each region. We further address the spatial information loss in the error signal on the confidence map caused by global average pooling (Lin et al., 2013).

To evaluate our approach, we present an empirical evaluation on the problem of grasp localization. We then compare our network to the previous one of Redmon and Angelova (2015), as well as the sparse dictionary learning approaches implemented for our comparative study. Results on the Cornell dataset (Jiang et al., 2011) show that our network with residual connections is easier to train than the one without residual connections and obtains better accuracy. We also performed better than the sparse dictionary learning approaches when considering processing speed.

A parametric activation function (Chapter 6)

In this contribution, we propose a new parametric activation function for deep neural networks. Even though many activation functions have been proposed over the past few years (Klambauer et al., 2017; Ramachandran et al., 2017; Scardapane et al., 2017), it is still an active domain of research. In this work, we looked at the advantage of parameterizing the Exponential Linear Unit (ELU). Parameterization can improve convergence and help find better solutions, as it was observed with the Parametric ReLU (PReLU) (He et al., 2015). Indeed, the PReLU learns a non-negative slope for negative entries and helps reduce the number of dead units.

We propose learning two parameters, each one controlling a different aspect of ELU. The first one adapts the saturation negative value for negative entries, and the second one changes the exponential decay towards the saturation. We evaluated our activation function on object recognition using the standard MNIST, CIFAR 10/100 and ImageNet 2012 datasets. Results showed that networks using our Parametric ELU (PELU) obtained better performance than using the non-parametric ELU. We observed that the networks learned different forms of parameterization. Some PELU looked like ReLU and favored sparse outputs, while others had large negative saturations to deal with bias shift.

A soft parameter sharing network architecture for multitask learning (Chapter 7)

In this contribution, we introduce a new soft parameter sharing mechanism for deep neural networks. An interesting property of a good representation is that it contains

factors of variation shared across more than one tasks. A standard way to adapt a deep neural network to this multitask learning setting is to have a shared central section with one head per task. Task-specific features will compete together and those relevant to all tasks will be favored. The architecture has however some constraints. Features relevant to all tasks may not be able to extract high-level features specific to a particular task. Such high-level features may be needed to obtain a good representation for the said task.

Based on the recent advances in residual learning, we propose a new soft parameter sharing mechanism integrating lateral skip connections. This mechanism uses two non-linear transformations. The first one aggregates the features of the task-specific hidden layers using a concatenation and creates a set of global features. The second one is a local aggregation, which we integrate with each layer of each task. It first transforms the global features and those of the specific layer into local features, and then integrates them with a residual connection. The global aggregation using concatenation allows combining different concepts from specific tasks into new shared features, and the local aggregation with a residual connection allows integrating back this new shared features only if it helps the specific task.

We evaluated our approach on facial landmark detection in a multitask setting. Our results show that our architecture obtains state-of-the-art performance. We also confirm with an ablation study, that was purposefully designed to impede knowledge sharing between the tasks, the known fact that landmark localization depends on face orientation. This constitutes an empirical evidence of information transfer between domains.

Thesis Outline

This thesis is organized as follows. Chapter 1 presents the background material on representation learning. It defines the task of representation learning and presents important *prior beliefs* to guide the search for the relevant factors of variation. Then, the following chapters present three practical frameworks that each implements one or more of these prior beliefs. Chapter 2 covers Sparse Dictionary Learning, Chapter 3 covers Deep Learning and Chapter 4 covers Multitask Learning. Afterwards, the following chapters correspond to our contributions. Chapter 5 presents our empirical study on the problem of grasp localization, where we compare thirty-six sparse dictionary approaches and evaluate our proposed adaptation of residual network. Chapter 6 introduces a novel parametric activation function. Based on previous work on the Parametric

Rectified Linear Unit, we propose learning a parameterization of the Exponential Linear Unit. Chapter 7 presents our soft parameter sharing mechanism for deep neural networks. In a multitask setting, our aggregation blocks allow extracting high-level features specific to a particular task as well as learn features relevant to all tasks. Finally, the thesis is concluded with a summary of our contributions and research avenues for future work.

Publication Notes

In Chapter 5, the work related to sparse dictionary learning appeared in (Trottier et al., 2017d), and the work related to residual network appeared in (Trottier et al., 2017b). The work of Chapter 6 appeared in (Trottier et al., 2017c), while the work of Chapter 7 is published online (Trottier et al., 2017a). We have also published works that we did not include in the thesis, since they are related to speech recognition. They appeared in (Trottier et al., 2015b) and in (Trottier et al., 2015a).

Chapter 1

Background Material on Representation Learning

In this chapter, we present background material on Representation Learning. We start in Section 1.1 with its definition, which can be summarized as the task of learning a representation that captures the underlying factors of variation that generated the data. In Section 1.2, we explain that representation learning approaches can rely on prior beliefs to guide their search. To this end, we review important prior beliefs that have been proposed over the last few years. We finally conclude the chapter in Section 1.3.

1.1 Definition

The difficulty of an information processing task depends on the way information is presented. This is a general principle that is not only applicable in machine learning, but also in computer science and in science in general. For instance, take the polar and Cartesian coordinate systems. A point P in space can be described with standard Cartesian coordinates as $P = (x, y)$, or can be converted into polar coordinates using the relations that $x = r\cos\theta$ and $y = r\sin\theta$, where r is the radial distance and θ is the angular coordinate. We often use the Cartesian coordinate system for our computations because we are more familiar with it. But it is not always the best system for all tasks. For example, the polar coordinate system can be more appropriate when performing integrals over a region D that cannot be described in terms of simple functions in the Cartesian coordinate system. This is the case when D is a circle or an intersection of more than one circle. The use of Cartesian coordinates in this case makes the double integral more difficult to compute due to the peculiar limits of the inner integral.

The overall idea of *Representation Learning* is similar to the idea of using polar coordinates when integrating over an unusual region D: to make the initial problem easier. The difference is that representation learning is developed in a context of machine learning, where the goal is to learn a feature representation of the original input observations that will facilitate a machine learning approach to obtain statistical generalization. In this case, the feature representation must capture the underlying factors of variation that generated the data to make the initial problem (that is, the learning problem) easier.

The difficulty of this task is to disentangle the factors of variation that are relevant from those that are irrelevant. To better illustrate this difficulty, consider the following problem. Suppose we define a classification problem where the goal is to classify geometric figures. For the sake of the argument, consider only binary classification where the task is to classify between squares and rectangles. In this simple case, we can define two factors of variation to generate the geometric figures: one for the length of the edges, and the other for the angle at which the edges are connected. On one hand, the underlying factors of variation that generate the squares must comply to the following rule: a figure with four straight edges connected perpendicularly, where all four edges have the same length. On the other hand, the underlying factors of variation that generate the rectangles must comply to the following rule: a figure with four straight edges connected perpendicularly, where each pair of opposite edges have the same length. In other words, a square and a rectangle differs only with respect to the length of their edges, since they both have the same angles. Therefore, the factor of variation corresponding to the angle at which the edges are connected is irrelevant for our task. Only the factor of variation corresponding to the length of the edges is important, since it is the only one that can discriminate between a square and a rectangle.

The previous example is simple (it has only two factors), but it can help us better understand the difficulty of disentangling relevant factors of variation from irrelevant ones. Consider again the classification problem between the squares and the rectangles, but this time with factors of variation shared between the geometric figures. For example, there could be a factor of variation for the thickness of the edge, for its color, for the scale of the figure, for its orientation, and so on. These factors would be applied to all geometric figures, yet only the one corresponding to the length of edges would be relevant for our task. This can be complexified further if we allow certain factors to be applied on specific geometric figures. For instance, rectangles could be constrained to always be blue and squares could be constrained to always be green. In this case,

color becomes relevant to our task, since it can be used to discriminate between a square and a rectangle. However, discriminating with color can be risky, because color does not properly characterize a square or a rectangle. It helps for the classification problem, but could be detrimental for other problems where different geometric figures could have the same color. This is a concern because we want our learned feature representations to be usable in most contexts.

How can we then design a learning framework to help a learning algorithm to disentangle the factors of variation that are relevant from those that are irrelevant? One of the main strategy employed to achieve this goal is to introduce clues to guide learning towards the relevant ones. These clues can take many forms, but one clue that is common in machine learning in general is labels associated with an observation. A label indicates that an observation belongs to a specific object category and directly specifies at least one factor of variation that generated the observation. For instance, if we consider the classification problem introduced earlier, using a label indicating the color of the geometric figure would directly specify the factor of variation corresponding to the color of the edges. The labels can also be more generic and determine a group of factors of variation. For instance, the label indicating that a given figure is a square informs that the factors of variation found in the given figure are associated with those that make up a square. Using labels is the most efficient strategy to provide a strong clue about the relevant underlying factors of variation.

Representation learning can also rely on another strategy that makes use of other, more indirect clues about the underlying factors of variation. These clues take the form of *prior beliefs*, which are regularization strategies that reflect a preference for specific characteristics over others. The use of prior beliefs can be motivated by learning principles from biology, psychology or other branches of machine learning, but it can sometimes be motivated by intuition. The fact is that all prior beliefs can be relevant, whether they are motivated by a long-established guiding principle or by intuition alone. This is due to a consequence of the No Free Launch (NFL) theorem (Wolpert and Macready, 1997) that informs us that a universally superior regularization strategy does not exist. The main challenge is rather to find a set of fairly generic prior beliefs that can be applied to a set of tasks that we consider important.

1.2 Review of Important Prior Beliefs

The following sections provide a list of common prior beliefs that can be used to guide the learning algorithm to find the relevant factors of variation. The list has been created mostly by Bengio et al. (2013) and has further been extended by Goodfellow et al. (2016). The list is not exhaustive but reflects most of the priors beliefs that have been proposed over the last few decades.

1.2.1 Multiple Explanatory Factors

The *multiple explanatory factors* prior belief states that data generation is governed by underlying factors of variation. It also suggests that for the most part, what is learned about one factor generalizes to the configuration of the other factors. A representation subject to this prior belief would therefore try to recover or at least disentangle the underlying factors of variation. This is a general prior belief and is at the root of the majority of representation learning approaches.

1.2.2 Distributed Representations

The characteristic of a representation to be *distributed* is fundamental in representation learning. A distributed representation is composed of elements that can be set separately from each other. It contrasts with the *local* or *symbolic* representation, where only one element can be set at a time. For instance, the feature representation computed by the K-Means algorithm (MacQueen et al., 1967) is a binary vector with one at the position of the nearest cluster and zero everywhere else. There can be only one nearest cluster [1], which means that a K-dimensional feature representation is needed to describe K concepts. In contrast, a learning algorithm that extracts a binary feature representation by thresholding K linear functions of the input in $\mathbb{R}^d$ can distinguish between $O(K^d)$ regions (Pascanu et al., 2014).

Distributed representations are more expressive than local representations. In general, the number of concepts that a distributed representation can distinguish is exponential in the input dimensionality d and polynomial in the number K of linear functions. When confronted with complex observations that have an intricate latent structure, the distributed representation will more efficiently describe the underlying factors of variation from the data than a local representation.

[1] Ties are broken at random

1.2.3 Simplicity of Factor Dependencies

The *simplicity of factor dependencies* suggests that the elements of a good representation should be related to each other through simple dependencies. The simplest type of dependency is independence, which states that the value of one element conveys no information about the values of the others. A representation where elements exhibit independence can be useful to leverage simple approaches, like the Naive Bayes classifier (Murphy, 2006).

The linear dependency is also a reasonable type of dependency. It states that a linear variation of an element changes linearly the values of all other elements. A representation where all elements are related to each other through linear dependencies can also be useful to leverage simple approaches, like the Linear model (Seal, 1967). Beside, we used this prior belief in our introductory example in the introduction chapter.

1.2.4 Sparse Representations

Sparsity is an example of a prior belief that is motivated by intuition. It suggests that only a small number of factors of variation are needed to explain an observation x. A feature representation subjected to the sparsity prior belief is constrained to have its binary features (that can either be *inactive* or *active*) to be inactive most of the time. It is also constrained to have any non-binary features to behave as much as possible as binary features by constraining them to be either inactive (a value exactly at zero) most of the time or active (a value largely different from zero) on rare occasions.

Sparsity follows from the more general *Occam's razor* principle, also known as the *law of parsimony* (Blumer et al., 1987). The Occam's razor principle states that: "Among competing hypotheses, the one with the fewest assumptions should be favored." (Wikipedia, Occam's razor). Informally, it states a preference for what is simple over what is complex.

This preference is often used as a guiding principle when developing models in science. It has been observed in humans and is believed to be an intrinsic property of our psychology (Rubin et al., 2010; Alter and Oppenheimer, 2006; Alter et al., 2007; Cannon et al., 2010; Corley et al., 2007). For instance, a study has shown that deliberately increasing the complexity of your vocabulary when writing a text has a negative impact on judged intelligence (Oppenheimer, 2006). Another study has shown that using words that are difficult to pronounce fosters the impression of higher risk (Song and

Schwarz, 2009). Moreover, another study has shown that consumers can be influenced to prefer a product over another by using an easy-to-read font during the presentation of the product (Novemsky et al., 2007). These studies indicate that we have developed a preference for what is simple throughout history, which motivates the use of the Occam's razor principle as a heuristic to guide the development of representation learning approaches.

1.2.5 Hierarchical Representations

The *hierarchical* prior belief states that high-level abstract concepts can be defined in terms of low-level simple concepts. Organizing concepts in such a way forms a *hierarchy* with different levels of abstraction, where low-level concepts appear at the bottom of the hierarchy and high-level ones appear at the top. A hierarchy is a way of organizing things that is often seen in nature. We have observed it, for instance, in the human body, in astronomical objects and in dominance hierarchies of groups of social animals. The idea of this prior is thus to allow a hierarchical organization to naturally occur during representation learning. We will see in Chap 3 an instance of naturally occurring hierarchical organizations in deep neural networks (Zeiler and Fergus, 2014).

1.2.6 Shared Factors Across Tasks

The *shared factors across tasks* prior belief states that a feature representation should capture similar factors of variation from tasks with similar types of observations. This prior belief is motivated by the fact that we can perform complex and unrelated tasks even though we have access to only one *biological* neural network. The brain only uses stimuli from the sensory nervous system, yet it is able to differentiate sensory signatures arising from different contexts. And with these signatures, the brain settles its neural activity to create a consistent representation of the world.

The idea is thus to learn a consistent feature representation from more than one source of information that is usable for more than one task. A certain subset of the features will be relevant for a first task, another subset will be relevant for a second task, and so on, so the overall feature representation is relevant for all tasks.

1.2.7 Smoothness

The *smoothness* prior belief states that the mapping f that we want to learn should be smooth, that is, $f(x) \approx f(x + \epsilon d)$ for a random unit direction d and a small scalar ϵ. In

other words, it states that f should be locally consistent, which means that it should not change abruptly within a small region. Using smooth mappings has the advantage of lowering overfiting (Goodfellow et al., 2016), since their smoothness prohibits them from complying to insignificant variations of the inputs. However, smoothness alone cannot overcome overfitting arising from the curse of dimensionality (Goodfellow et al., 2016).

1.2.8 Causal Factors

The *causal factors* prior belief states that a representation learning approach should compute a feature representation that treats the factors of variation as the causes of the data. Causation is different from standard statistical dependency, from which most representation learning approaches rely on. The difference between causation and statistical dependency can be explained with the following example. It is well-known that smoking is correlated with alcohol abuse, but smoking does not cause alcoholism (Reed et al., 2007). The comorbidity of smoking and alcoholism is a statistical event that should not be treated as a causal event. On the other hand, stressful environments, drinking at an early age or depression can cause alcohol abuse, which can cause alcoholism when the circumstances are propitious (Blanco et al., 2013).

In a similar vein, the *causal factors* prior belief suggests that the factors of variation should be treated as the causes underlying the generation of the data. It proposes to switch from standard statistical learning to causal learning. However, causal learning is more difficult than statistical learning (Pearl et al., 2009), but can also be more rewarding. The feature representation can become invariant to changes in the distribution of the underlying causes when establishing causal relations instead of statistical relations (Goodfellow et al., 2016).

1.2.9 Manifolds

The *manifolds* prior belief states that a feature representation should explicitly learn the structure of the data manifold. The term manifold is often used in statistical learning to denote a region of low dimensionality where a probability distribution concentrates its mass. This assumption has been confirmed in real-world observations, such as images, sounds and texts.

One simple way to show the existence of the manifold of real-world images is to run the following experiment (Goodfellow et al., 2016). Take a high-resolution image of an ob-

ject and try to replicate that image by sampling a uniform distribution over the pixels of the image. After a few tries, it becomes apparent that it would take an overwhelming number of tries to do so. The uniformly sampled images do not look like real-world images, but rather look like static white noise. This reveals that the distribution of real-world images is not well-approximated by the uniform distribution, which points toward the conclusion that real-world images lie in a region of low dimensionality characteristic of a manifold. The fact that real-world observations exhibit a data manifold motivates the idea that the feature representation should directly learn the structure of the manifold.

1.2.10 Natural Clustering

The *natural clustering* prior belief suggests that each connected manifolds in input space should be grouped together and be assigned to a single class. The overall data distribution may be constituted of more than one disconnected manifolds, but all observations within a group of connected manifolds should be conceptually identical. The idea of grouping similar observations together is coherent with how we tend to group objects with similar characteristics into a category.

1.2.11 Temporal and Spatial Coherence

The *temporal and spatial coherence* prior belief states that important factors of variation change slowly over time or space. This prior is appropriate when learning a feature representation on observations like signals that have temporal coherence or images that have spatial coherence. A feature representation subject to temporal or spatial coherence will be able to interpret incoherent changes in raw input space as coherent changes in the space of factors of variation.

1.3 Conclusion

In this chapter, we introduced prior beliefs for representation learning. The difficulty of learning a feature representation capturing the underlying factors of variation is to disentangle those that are relevant from those that are irrelevant. Prior beliefs can be used as a heuristic to reflect a preference for specific factors of variation and can be used to guide the learning algorithm towards them.

So far, the presentation of the prior beliefs has been conceptual and abstract. In the following chapters, we will introduce three practical learning frameworks that make use of one or more of these prior beliefs. In Chapter 2, we start with Sparse Dictionary Learning (SDL). SDL is a subfield of representation learning that implements approaches based on the *sparsity* prior belief. Then, we will introduce Deep Learning in Chapter 3, which focuses on the task of training deep neural networks. A deep neural network implements the *hierarchical*, the *distributed* and the *simplicity of factor dependencies* prior beliefs. Finally, we will introduce Multitask Learning in Chapter 4, which deals with the problem of learning many tasks in parallel. Approaches in this category focus on the *shared factors across tasks* prior belief.

Chapter 2

Background Material on Sparse Dictionary Learning

In this chapter, we present background material on Sparse Dictionary Learning (SDL). SDL is a framework that implements the *sparsity* prior belief that we introduced in Section 1.2.4. We start in Section 2.1 with a definition of this framework. In particular, we detail the difference between feature coding and dictionary learning, and also the importance of whitening and sparsity measures. Then, we present a review of important feature coding approaches in Section 2.2, such as Feature Sign Search, Matching Pursuit, Least Angle Regression and Marginal Regression. Finally, we present a review of important dictionary learning works in Section 2.3, such as Method of Optimal Direction, Gradient Descent, Online Dictionary Learning and Gain Shape Vector Quantization.

2.1 Definition of Sparse Dictionary Learning

Sparse Dictionary Learning (SDL) poses that an observation can be decomposed as a sparse linear combination of atoms from a dictionary. In its generative form, the approach can be formulated as follows:

$$p(D) \propto \prod_{j=1}^{d} Uniform(D_{:,j}) \;,\; \|D_{:,j}\|_2^2 \leq 1 \,, \tag{2.1.1}$$

$$p(w^{(i)}) \propto \prod_{i=1}^{N} \frac{\lambda'}{2} \exp\left(-\lambda' |w^{(i)}|\right) \;, \tag{2.1.2}$$

$$p(x^{(i)}|D, w^{(i)}) = \mathcal{N}(x^{(i)}|Dw^{(i)}, \sigma^2 I) \,. \tag{2.1.3}$$

where $x^{(i)} \in \mathbb{R}^n$, for $i \in \{1 \dots N\}$, are the observations, N is the number of observations, n is the dimensionality of the input $x^{(i)}$, $D \in \mathbb{R}^{n \times d}$ is the dictionary whose columns $D_{:,j}$ represent the atoms, d is the number of atoms, and $w^{(i)} \in \mathbb{R}^d$ is the sparse weight vector describing the linear combination $Dw^{(i)}$.

Eq. (2.1.1) poses *a priori* that the dictionary atoms $D_{:,j}$ are independently and uniformly distributed inside a n-dimensional unit hypersphere. The restriction on the norm is necessary to avoid divergence during optimization.

Eq. (2.1.2) defines sparsity on the weight vector $w^{(i)}$ by assuming that the weights are *a priori* independent and distributed according to a Laplace distribution centered at 0 with an inversed scale parameter λ'. Parameter λ' governs the magnitude of the peak of the distribution and can be tuned to adjust the sparsity.

Finally, Eq. (2.1.3) defines the *likelihood* and poses that observation $x^{(i)}$ can be modelled by a normal distribution centered at the estimate $Dw^{(i)}$ with an isotropic variance σ^2. This distribution expresses the magnitude of acceptable biases between the observation $x^{(i)}$ and its estimate $Dw^{(i)}$.

A standard SDL approach is divided into a *dictionary learning* phase and a *feature coding* phase. The goal of the dictionary learning phase is to train a dictionary D to capture the latent structure of the data, while the goal of the feature coding phase is to compute the feature representation of an observation $x^{(i)}$ using dictionary D. During both phases, the generative formulation given by Eqs. (2.1.1), (2.1.2) and (2.1.3) are used as the underlying principle of the approach. In the following sections, we will introduce them both, starting with feature coding, then dictionary learning.

2.1.1 Feature Coding

In SDL, the feature representation of observation $x^{(i)}$ computed during feature coding is defined as the weight vector $w^{(i)}$ that maximizes the *a posteriori* distribution of Eq. (2.1.3).

Derivation

Assuming we have a dictionary D, the feature coding task can be expressed as the following optimization:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmax}}\, p(w^{(i)}|x^{(i)}, D) \qquad (\textit{a posteriori} \text{ distribution}) \quad (2.1.4)$$

$$= \underset{w^{(i)}}{\text{argmax}}\, p(x^{(i)}|w^{(i)}, D)p(w^{(i)}|D) \qquad \text{(Bayes rule)} \quad (2.1.5)$$

$$= \underset{w^{(i)}}{\text{argmax}}\, p(x^{(i)}|w^{(i)}, D)p(w^{(i)}) \qquad \text{(independence assumption)} \quad (2.1.6)$$

$$= \underset{w^{(i)}}{\text{argmin}} - \log p(x^{(i)}|w^{(i)}, D) - \log p(w^{(i)}) \qquad \text{(properties of log)} \quad (2.1.7)$$

$$= \underset{w^{(i)}}{\text{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\|w^{(i)}\|_1 \,. \qquad \text{(Eq. (2.1.3) and (2.1.2))} \quad (2.1.8)$$

As shown in Eq. (2.1.8), the goal of feature coding is to find the sparsest weight vector $w^{(i)}$ such that $Dw^{(i)}$ is a good estimate of $x^{(i)}$. The problem is defined as a ℓ_1-penalized least square optimization. The quantity to be minimized is the *residual* $\|Dw^{(i)} - x^{(i)}\|_2^2$ (squared Euclidean norm of the residue $r = Dw^{(i)} - x^{(i)}$) and the penalty term is the ℓ_1 norm on the weight vector:

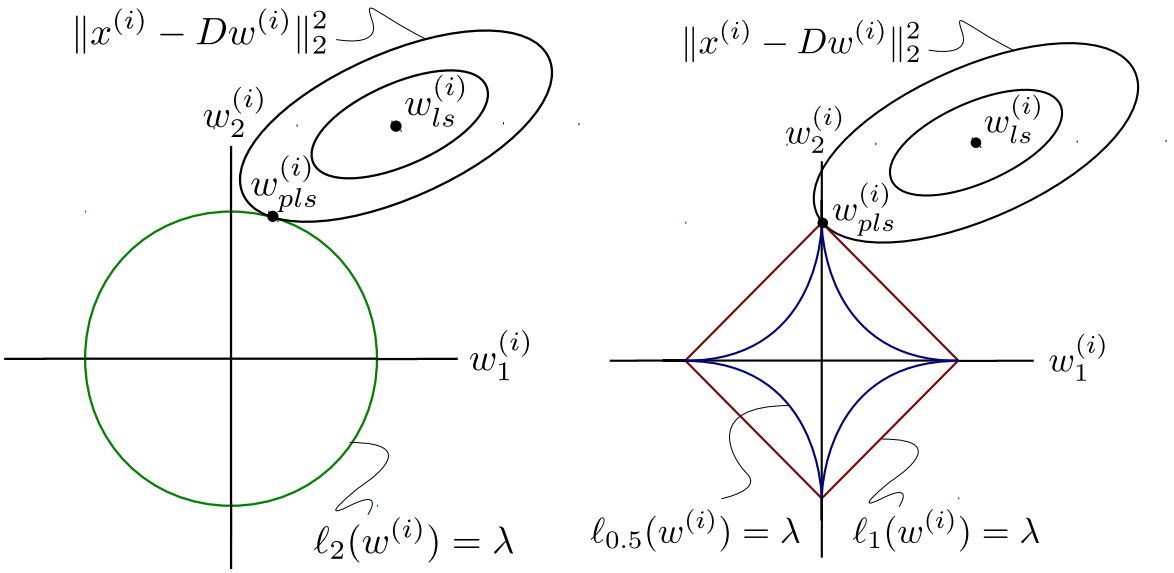
$$\ell_1(w^{(i)}) = \|w^{(i)}\|_1 = \sum_{k=1}^{d} \left|w^{(i)}\right| \,. \qquad (2.1.9)$$

Parameter λ governs the importance of minimizing the penalty term ℓ_1. A large λ value will put more emphasis on minimizing the penalty $\|w^{(i)}\|_1$ (at the expense of having a large residual $\|Dw^{(i)} - x^{(i)}\|_2^2$), while a small λ value will put more emphasis en minimizing the residual $\|Dw^{(i)} - x^{(i)}\|_2^2$ (at the expense of having a large penalty $\|w^{(i)}\|_1$).

Role of Sparsity

The ℓ_1 penalty term in Eq. (2.1.8) has a direct influence on the sparsity of the solution. To understand how, we can look at the geometry of the solution set under different types of penalty. An example using the ℓ_2, ℓ_1 and $\ell_{0.5}$ penalties is shown in Figure 2.1. With a ℓ_2 penalty, the solution set is circular. In this case, the optimal penalized least-square solution will rarely be positioned on the axis where the majority of its elements are zero (Figure 2.1 (a)). With the ℓ_1 penalty, the solution set is rectangular. In this case, the penalized least-square solution will often be localized at the corners where most of its elements are zero. This can also happen with the $\ell_{0.5}$ norm due to its concave solution set (Figure 2.1 (b)).

The sparsest weight solution $w^{(i)}$ that minimizes the penalized least-square problem of Eq. (2.1.8) is an accurate feature representation of observation $x^{(i)}$. This is because the weight values reflect the importance of each atom $D_{:,j}$ in the linear combination. Large weight magnitudes indicate the presence of factors of variation from which $x^{(i)}$



(a) ℓ_2 penalty.

(b) $\ell_{0.5}$ and ℓ_1 penalty.

Figure 2.1: Geometric intuition illustrating why ℓ_p, $p \leq 1$, leads to sparsity on a two-dimensional example. The vector $w_{ls}^{(i)}$ is the optimal least-square solution minimizing $\|x^{(i)} - Dw^{(i)}\|_2^2$, and the vector $w_{pls}^{(i)}$ is the optimal penalized least-square solution minimizing $\|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\ell_p(w^{(i)})$. In (a), the penalty is the ℓ_2 norm, while in (b) the penalties are the ℓ_1 and $\ell_{0.5}$ norms. The solution set in (a) is circular, which means that the solution $w_{pls}^{(i)}$ will rarely be positioned on the axis where the majority of the weights are equal to zero. In (b), the solution set is rectangular, which means that the solution $w_{pls}^{(i)}$ will often occur at the corners corresponding to a sparse $w_{pls}^{(i)}$.

is composed, and low weight magnitudes indicate the absence of factors of variation unrelated to $x^{(i)}$. As a result, observations that share these factors of variation will have weight vectors with similar entries that are non-zero. This is an important characteristic of the representation because observations will be similar in the feature space even though they may look different in the original image space. This allows the classifier trained on top of the features to better capture the relations between the observations.

Apart from constraining the solution set of the least-square optimization problem, sparsity has another important advantage: it restricts the number of factors of variation that can explain the data. This restriction is enforced in the solution $w^{(i)}$ to Eq. (2.1.8) by the requirement to have few active atoms, i.e. atoms with a corresponding weight value not equal to zero. This places constraints on the type of dictionary atoms $D_{:,j}$ that can be used to minimize both the residual and the ℓ_1 penalty. As a result, predefined dictionaries D will rarely contain the necessary atoms to well represent the data. For this reason, we rather learn the dictionary D from the data, which we refer to as *dictionary*

learning. We now introduce this framework in the next section.

2.1.2 Dictionary Learning

The goal of the dictionary learning phase is to learn a dictionary D such that feature coding can find weight vectors $w^{(i)}$ where both the residual and the penalty term are small, for all inputs $x^{(i)}$. One of the key principles of dictionary learning is that atoms $D_{:,j}$ are learned from the data. Previously to SDL, the standard practice was to use analytically predefined dictionaries, such as Discrete Cosine Transform (DCT) or Wavelet Transform (WT). These fixed dictionaries were rarely flexible enough to well represent the data, and were usually used only on specific tasks (Zhang et al., 2015).

Derivation

Similarly to feature coding, SDL defines dictionary learning as finding the dictionary D that maximizes the *a posteriori* distribution of Eq. (2.1.3). Let $X = \{x^{(i)}\}_{i=1}^{N}$ be the set of all N i.i.d observations $x^{(i)}$ and $W = \{w^{(i)}\}_{i=1}^{N}$ be the set of all weight vectors $w^{(i)}$. The task is expressed as the following optimization problem:

$$
\begin{aligned}
D &= \underset{D}{\operatorname{argmax}}\, p(D|X) && (a\ posteriori) && (2.1.10)\\
&= \underset{D}{\operatorname{argmax}}\, p(X|D)p(D) && (\text{Bayes rule}) && (2.1.11)\\
&= \underset{D}{\operatorname{argmax}} \prod_{i=1}^{N} p(x^{(i)}|D) && (\text{i.i.d assumption}) && (2.1.12)\\
&= \underset{D}{\operatorname{argmax}} \prod_{i=1}^{N} \int p(x^{(i)}|w^{(i)}, D)p(w^{(i)}|D)\mathbf{d}w^{(i)} && (\text{sum rule}) && (2.1.13)\\
&\approx \underset{D}{\operatorname{argmax}} \prod_{i=1}^{N} \max_{w^{(i)}} p(x^{(i)}|w^{(i)}, D)p(w^{(i)}) && (\text{approximation}) && (2.1.14)\\
&= \underset{D}{\operatorname{argmin}} \min_{W} \sum_{i=1}^{N} -\log p(x^{(i)}|w^{(i)}, D) - \log p(w^{(i)}) && (\text{properties of log}) && (2.1.15)\\
&= \underset{D}{\operatorname{argmin}} \min_{W} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\|w^{(i)}\|_1 && (\text{Eq. (2.1.3) and (2.1.2)}) && (2.1.16)
\end{aligned}
$$

As shown in Eq. (2.1.16), the goal of dictionary learning is to find a dictionary D such that feature coding can find the sparsest weight vector solutions $w^{(i)}$ having small

residual values. It is defined as a nested optimization of the penalized least-square problem over both D and $w^{(i)}$.

Alternating Minimization

Minimizing (2.1.16) for both the dictionary D and the weights $w^{(i)}$ is however computationally expensive. This is due to the interleaving dependency between D and all $w^{(i)}$ in the nested optimization problem. More viable alternatives rather make use of the *Alternating Minimization* scheme to perform the minimization (Kreutz-Delgado et al., 2003). Approaches based on alternating minimization sequentially improve an estimate of the optimal solution in a greedy fashion until convergence. The greedy step performs approximate optimization over a subset of the unknown variables by considering the others as fixed. They are usually applied on optimization problems that are either too computationally demanding to solve directly or that do not admit an analytical solution.

A general alternating minimization approach usually works as follows. Step 1, determine the initial value of the solution at random. Step 2, minimize the objective with respect to a subset of the variable by considering the variables outside the subset as fixed. Step 3, minimize again the objective, but this time with respect to the variables outside the subset by considering the variables inside the subset as fixed. The algorithm alternates between step 2 and 3 until the loss function agrees with a determined convergence criterion, which is algorithm-specific. During step 2 and 3, the values of the fixed variables are those computed during the previous iteration.

Alternating Minimization for Dictionary Learning

In the case of dictionary learning, step 1 corresponds to initializing both the dictionary D and the weights $w^{(i)}$ at random. Then, step 2 corresponds to minimizing Eq. (2.1.16) with respect to all $w^{(i)}$ considering D fixed. Finally, step 3 corresponds to minimizing Eq. (2.1.16) with respect to D considering all $w^{(i)}$ fixed. The convergence criterion can be either when the loss function of Eq. (2.1.16) reaches a target value or when the magnitude of the dictionary updates are smaller than a target value (Kreutz-Delgado et al., 2003).

Note that when minimizing Eq. (2.1.16) over all $w^{(i)}$ considering D fixed (in step 2), we can invoke the i.i.d. assumption of the observations $x^{(i)}$ to simplify the optimization. Instead of minimizing the penalized least-square jointly over all $w^{(i)}$ simultaneously,

Algorithm 2.1 Dictionary Learning with Alternative Minimization.

Require: Dataset X, penalty λ and number of iterations T.
Ensure: $\|D_{:,j}\|_2 = 1\,,\ \forall j \in \{1 \dots d\}$

1: Initialize dictionary D: $D_{:,j} \sim Uniform(0,1),\ \forall j \in \{1 \dots d\}$
2: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2},\ \forall j \in \{1 \dots d\}$
3: **for** $t = 1 \dots T$ **do**
4: Minimize Eq. (2.1.16) over all $w^{(i)}$ considering D as fixed:

$$w^{(i)} \leftarrow \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\|w^{(i)}\|_1\,, \qquad \forall i \in \{1, \dots, N\}$$

5: Minimize Eq. (2.1.16) over D considering all $w^{(i)}$ as fixed:

$$D \leftarrow \underset{D}{\operatorname{argmin}} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2$$

6: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2},\ \forall j \in \{1 \dots d\}$
7: **end for**
8: **return** D

we can minimize the penalized least-square over each i individually. The task can be defined as follows:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\|w^{(i)}\|_1\,, \qquad \forall i \in \{1, \dots, N\}\,. \tag{2.1.17}$$

In other words, the optimization task of step 2 comes down to performing feature coding N times over all observations $x^{(i)}$ using the most recent updates of D. As for the minimization of Eq. (2.1.16) over D considering all $w^{(i)}$ as fixed (in step 3), this is simply a least-square optimization:

$$D = \underset{D}{\operatorname{argmin}} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2\,. \tag{2.1.18}$$

Overall Dictionary Learning Algorithm

A description of the overall dictionary learning approach is presented in Algorithm 2.1. The algorithm starts by initializing the dictionary D (line 1) and by constraining the dictionary atoms to the unit hyper-sphere (line 2). Then, the following three steps are performed for T iterations. First, perform feature coding on all observations $x^{(i)}$ (line 4). Second, update the dictionary D by minimizing Eq. (2.1.18) (line 5). Third, constrain

the dictionary atoms to the unit hyper-sphere (line 6). The algorithm ends at line 8 by returning the learned dictionary D.

Convergence Properties of Alternating Minimization

The global dictionary learning problem of Eq. (2.1.16) is non-convex (Mairal et al., 2014). That is, the joint minimization of Eq. (2.1.16) with respect to both the weights $w^{(i)}$ and the dictionary D is non-convex. This means that in general, we have no guarantees that the solution will be the *global* minimum. However, we can still have theoretical guarantees about convergence. Indeed, note that even though Eq. (2.1.16) is jointly non-convex, it is convex in the weights $w^{(i)}$ considering the dictionary D as fixed and it is convex in the dictionary D considering the weights $w^{(i)}$ as fixed (Mairal et al., 2014). In this case, it can be shown that the alternating minimization scheme, from which Algorithm 2.1 is based, always converges to a stationary point (Xu and Yin, 2013). This means that as long as the dictionary learning approach comply to the formulation of Algorithm 2.1, they are ensured to converge. In our review of the important dictionary learning approaches in Section 2.3, most will comply to the formulation of Algorithm 2.1. Thus, we will not specify the proof of convergence for them, only for those that do not comply to Algorithm 2.1.

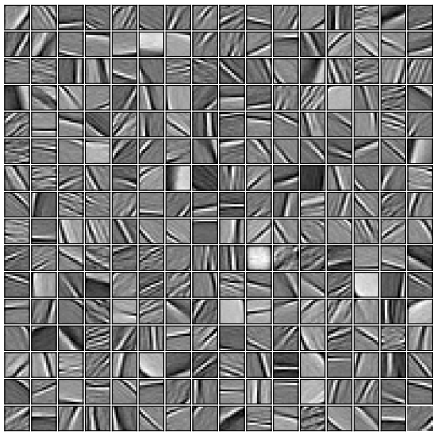
Dictionary Learning Example on Images

To better illustrate what a dictionary D looks like, we trained one on a real-world image and showed it in Figure 2.2. We used as observation the image *Raccoon Procyon Lotor* (PIXNIO, 2018) and optimize the dictionary following Algorithm 2.1 [1]. We first extracted at random 35,000 small 12×12 patches from the image and used their vectorized 144-dimensional vector as observation $x^{(i)}$. We then trained a dictionary D of $d = 256$ atoms with a value of $\lambda = 0.05$ on the penalty term. The image at the bottom left of Figure 2.2 shows all 256 de-vectorized 12×12 dictionary atoms $D_{:,j}$.

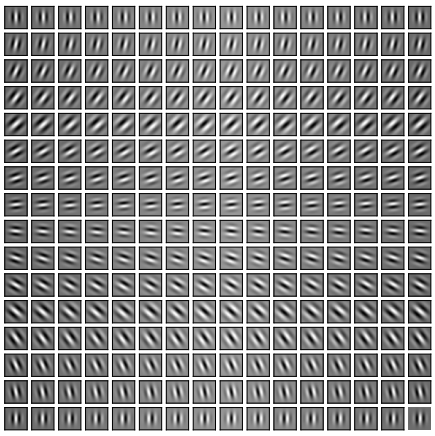
The visualization of the learned dictionary D reveals an interesting structure that was learned from the data. In particular, it shows that the dictionary has learned atoms that look like Gabor filters. For comparison, an example of a dictionary of Gabor filters is shown at the bottom right of Figure 2.2. A Gabor filter is characterized by its impulse response that is defined with a sinusoidal plane wave (in 2D) multiplied by a Gaussian kernel. We can use them to detect the presence of image content with a specific frequency

[1] To be more precise, we used a specific instantiation of Algorithm 2.1 from Mairal et al. (2010)'s work. This approach will be detailed in Section 2.3.4.

(a) *Raccoon Procyon Lotor* Image (PIXNIO, 2018)



(b) Dictionary of Learned Atoms



(c) Dictionary of Gabor Filters

Figure 2.2: Dictionary learning on the *Raccoon Procyon Lotor* image. A total of 35,000 patches with dimensions 12×12 were extracted and a dictionary of 256 atoms was trained with Mairal et al. (2010)'s instantiation of Algorithm 2.1. Figure (a) at the top is the training image, figure (b) at the bottom left shows all 256 learned dictionary atoms, and figure (c) at the bottom right shows 256 Gabor filters. A visual inspection of the learned dictionary reveals that the atoms share similarities with Gabor filters. Gabor filters have been claimed to be similar to those of the human visual system (Marĉelja, 1980; Daugman, 1985, 1980; Mély and Serre, 2017), which suggests that the dictionary has captured factors of variation considered valuable by the human visual system.

at a specific location. For instance, they have been used in previous works on texture analysis with great success (Fogel and Sagi, 1989; Jain and Farrokhnia, 1991).

What makes Gabor filters particularly special in our case is that they have been claimed to be similar to those of the human visual system (Marĉelja, 1980; Daugman, 1985, 1980; Mély and Serre, 2017). The fact that our dictionary has learned atoms that look like

Gabor filters suggests that the dictionary can capture factors of variation considered valuable by the human visual system. A closer evaluation of the learned dictionary also reveals that some atoms are more complex than Gabor filter. These factors cannot be expressed easily as Gabor filters, which shows that the dictionary has learned features that are not easily expressed in simple analytical terms, i.e. a product of a sine with an exponential. This ability to discover factors of variation for which we do not have a clear analytical representation is what we expect from a representation learning approach, like sparse dictionary learning.

2.1.3 ZCA Whitening

Whitening plays an important role when learning a dictionary D. This transformation is a decorrelation transformation that aims to transform a set of observations having a covariance matrix C to a new set of observations having an identity covariance matrix $\mathbb{I}$. In other words, its goal is to remove linear dependencies. This is particularly advantageous in the context of dictionary learning because nonlinear dependencies are easier to capture when the linear ones are removed.

Let $X = [x^{(1)} \dots x^{(N)}]$ be the matrix containing in its columns all observations $x^{(i)}$. Suppose that the data are centered and let the covariance matrix $C = \mathbb{E}\left[XX^\top\right]$ have the eigendecomposition $C = VDV^\top$, where eigenvectors are in the columns of V and the eigenvalues are in the diagonal of D. The whitening transformation is given by matrix $W = D^{-1/2}V^\top$ so that the whitened data $Y = WX$ have identity covariance matrix:

$$\begin{aligned}
\mathbb{E}\left[YY^\top\right] &= \mathbb{E}\left[D^{-1/2}V^\top XX^\top V(D^{-1/2})^\top\right] \\
&= D^{-1/2}V^\top CV(D^{-1/2})^\top \\
&= D^{-1/2}V^\top VDV^\top V(D^{-1/2})^\top \\
&= \mathbb{I}
\end{aligned}$$

This transformation is also known as PCA whitening. However, it is not unique. Indeed, whitened data stay whitened after any rotation. This means that $W = RD^{-1/2}V^\top$, for any orthogonal matrix R, is also a valid whitening transformation. In what is known as ZCA whitening, we use V as rotation matrix:

$$W_{ZCA} = VD^{-1/2}V^\top \tag{2.1.19}$$

The important difference between W_{ZCA}, using $R = V$, and the other W, using any orthogonal matrix R, is that W_{ZCA} makes the whitened data as close as possible to the original un-whitened data. The solution W_{ZCA} can in fact be defined as the solution to the following optimization problem:

$$W_{ZCA} = \arg\min_{W} \left\{ \|X - WX\|_F \mid \mathbb{E}\left[(WX)(WX)^\top\right] = \mathbb{I} \right\} \tag{2.1.20}$$

where $\|\cdot\|_F$ is the Frobenius norm. In other words, W_{ZCA} is the matrix that results in whitened data $Y = WX$ that are as close as possible to the original data X. This way, the whitened data Y keep structural information and nonlinear dependencies related to the original data which allows dictionary learning to learn natural atoms.

2.1.4 Sparsity Measures

The ℓ_1 penalty in Eq. (2.1.8) leads to sparse solutions $w^{(i)}$ due to the geometry of the solution set (we have seen why in Figure 2.1). We also briefly mentioned that the $\ell_{0.5}$ penalty can also lead to sparsity, even though each penalty provides a different *type* of sparsity. Using a similar reasoning, we can generalize these two cases and express a general formulation of ℓ_p norms that lead to sparse solutions:

$$\ell_p(w^{(i)}) = \begin{cases} \#\left\{k \mid w_k^{(i)} \neq 0,\, k = 1, \ldots, n\right\} & p = 0 \\ \left(\sum_{k=1}^{n} |w_k^{(i)}|^p\right)^{1/p} & 0 < p < 1 \end{cases} . \tag{2.1.21}$$

The ℓ_0 penalty norm is often used as alternative to the frequent ℓ_1 norm. It is a direct measurement of sparsity since it computes the number of non-zero entries in a vector. In this case, the feature coding penalized least-square problem of Eq. (2.1.8) is defined as follows:

$$w^{(i)} = \operatorname*{argmin}_{w^{(i)}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda \|w^{(i)}\|_0 \,. \tag{2.1.22}$$

One concern when using ℓ_0 as penalty term is its intractability. Natarajan (1995) has shown that solving Eq. (2.1.22) becomes NP-Hard with a reduction to the NP-Complete subset selection problem. Indeed, to find an optimal solution to Eq. (2.1.22) requires iteratively searching over all candidate weight vectors $w^{(i)}$ having γ or less active atoms. For instance, starting with $\|w^{(i)}\|_0 = 1$, the algorithm would find all candidate solutions with exactly 1 non-zero entry. It would then continue with $\|w^{(i)}\|_0 = 2$ and find

all $\binom{\gamma}{2}$ candidate solutions with exactly 2 non-zero entries. Finally, it would repeat until $\|w^{(i)}\|_0 = \gamma$. At each step, the search for all $\binom{\gamma}{k}$ would require an expensive combinatorial search, which would make the complexity of the whole search exponential.

The ℓ_0 penalty is still often used in practice because we can get around the computational burden by solving Eq. (2.1.22) approximately. In the next section, we will review approaches to do so. In particular, we will review greedy optimization that finds a solution in γ iterations, as well as *one-shot* optimization that finds a solution in a single iteration. We will also cover iterative optimization for solving Eq. (2.1.8) with the ℓ_1 penalty.

2.2 A Review of Important Feature Coding Approaches for the ℓ_0 and ℓ_1 Penalties

In this section, we review important works on feature coding. The goal of feature coding approaches is to solve either the ℓ_1-penalized least-square optimization problem of Eq. (2.1.8) or the ℓ_0-penalized problem of Eq. (2.1.22). We review the most important ones, such as Feature Sign Search (FSS) (Lee et al., 2006), Least Angle Regression (LARS) (Efron et al., 2004) and Marginal Regression (Donoho and Johnstone, 1995) for the ℓ_1 penalty, as well as Matching Pursuit (Mallat and Zhang, 1993) and Orthogonal Marching Pursuit (Pati et al., 1993) for the ℓ_0 penalty.

2.2.1 Feature Sign Search

Lee et al. (2006) proposed the Feature Sign Search (FSS) algorithm for optimizing the penalized least-square of Eq. (2.1.8) problem under a ℓ_1 penalty. The approach is based on iterative optimization, where an estimate of the optimal solution is updated sequentially. At each step, FSS keeps track of the set A of active atoms and consider adding an inactive atom only if it locally improves the objective. Lee et al. (2006) provided a proof that their algorithm always converges, and we refer the reader to their work for details.

The approach is presented in Algorithm 2.2. Starting with an empty set $A = \{\}$ of active atoms (line 1), FSS first selects an inactive atom j as candidate to be added to

Algorithm 2.2 Feature Sign Search

Require: Dictionary D, observation $x^{(i)}$ and parameter λ.

1: Initialize approach: $A = \{\}$, $w_j^{(i)} = 0$ and $\theta_j = 0$, $\forall j \in \{1 \dots d\}$, where $\theta_j \in \{-1, 0, 1\}$.
2: **while** conditions of Eq.(2.2.7) and Eq.(2.2.8) are not met **do**
3: Select candidate based on absolute derivative. See Eq. (2.2.1).
4: Activate j only if it improves objective. See Eq. (2.2.2).
5: Compute new solution. See Eq. (2.2.6).
6: Perform *discrete* line search between $w_{new}^{(i)}$ and $w^{(i)}$ to find the point where the objective is minimum. Update A and $w^{(i)}$ accordingly.
7: **end while**

A (line 3). The candidate atom j is determined using the following gradient rule:

$$j = \underset{j}{\operatorname{argmax}} \left| \frac{\partial \|x^{(i)} - Dw^{(i)}\|_2^2}{\partial w_j^{(i)}} \right| , \tag{2.2.1}$$

In other words, it selects the most influential atom j that has the highest absolute derivative of the residual $\|x^{(i)} - Dw^{(i)}\|_2^2$ with respect to the weight $w_j^{(i)}$. Then, FSS considers activating j only if its absolute derivative is larger than λ (line 4). Activating atom j comes down to adding j to set A and setting θ_j to the negative sign of the derivative:

$$\begin{cases} \theta_j = -1 \text{ and } A \leftarrow A \cup \{j\} & \dfrac{\partial \|x^{(i)} - Dw^{(i)}\|_2^2}{\partial w_j^{(i)}} > \lambda \,, \\ \theta_j = 1 \text{ and } A \leftarrow A \cup \{j\} & \dfrac{\partial \|x^{(i)} - Dw^{(i)}\|_2^2}{\partial w_j^{(i)}} < -\lambda \,, \\ \text{reject} & \text{otherwise.} \end{cases} \tag{2.2.2}$$

where λ is the weight on the ℓ_1 penalty term. After activating (or not) a new atom, FSS solves an unconstrained quadratic program using only the active atoms (line 5). Let $\bar{D}$, $\bar{w}^{(i)}$ and $\bar{\theta}$ represent the dictionary, the weight vector and the sign vector of active atoms only. The quadratic program to be solved is as follows:

$$\bar{w}_{new}^{(i)} = \underset{\bar{w}^{(i)}}{\operatorname{argmin}} \|x^{(i)} - \bar{D}\bar{w}^{(i)}\|_2^2 + \lambda \bar{\theta}^\top \bar{w}^{(i)} \,. \tag{2.2.3}$$

Eq (2.2.3) can be solved analytically by equalling to zero the gradient of the penalized objective with respect to $w^{(i)}$:

$$0 = -2\bar{D}^\top (x^{(i)} - \bar{D}\bar{w}_{new}^{(i)}) + \lambda\bar{\theta} \tag{2.2.4}$$

$$2\bar{D}^\top \bar{D}\bar{w}_{new}^{(i)} = 2\bar{D}^\top x^{(i)} - \lambda\bar{\theta} \tag{2.2.5}$$

$$\bar{w}_{new}^{(i)} = (\bar{D}^\top \bar{D})^{-1}(\bar{D}^\top x^{(i)} - \frac{\lambda}{2}\bar{\theta})\,. \tag{2.2.6}$$

Finally, FSS performs a discrete line search between $\bar{w}_{new}^{(i)}$ and $\bar{w}^{(i)}$, and selects the candidate $\hat{w}$ with the lowest objective (line 6). The set of active atoms A, the sign vector θ and current solution $w^{(i)}$ are then updated according to the active and inactive atoms in the candidate $\hat{w}$. The approach stops when the following two conditions are met:

$$\frac{\partial \|x^{(i)} - Dw^{(i)}\|_2^2}{\partial w_j^{(i)}} + \lambda \text{ sign}(w_j^{(i)}) = 0,\ \forall w_j^{(i)} \neq 0 \tag{2.2.7}$$

$$\left| \frac{\partial \|x^{(i)} - Dw^{(i)}\|_2^2}{\partial w_j^{(i)}} \right| \leq \lambda,\ \forall w_j^{(i)} = 0 \tag{2.2.8}$$

In other words, the approach stops when the active elements have reached a critical point (Eq. (2.2.7)) and when no inactive elements can be added to the active set (Eq. (2.2.8), which is the *reject* case of Eq. (2.2.2)).

An important characteristic of FFS is its computational speed. Indeed, FSS reduces the computational burden by using only the active atoms when optimizing Eq. (2.2.3). This allows the approach to be used with large dictionaries. Indeed, the authors Lee et al. (2006) were able to evaluate their approach on four large datasets with RGB images, speech, stereo images and videos. They were able to extract good feature representations with lower time burden. Their approach has also been used with great success on other tasks, such as image classification (Yang et al., 2009; Long et al., 2013), texture reconstruction (Xu et al., 2011) and 3D facial-expression synthesis (Lin et al., 2012).

2.2.2 Matching Pursuit

Matching pursuit approaches target the penalized least-square problem of Eq. (2.1.22) under the ℓ_0 penalty. They are greedy methods that lower the computational burden of the NP-Hard optimization problem by searching for an approximate solution in a greedy fashion. Instead of exhaustively searching over all solutions $w^{(i)}$ where $\|w(i)\|_0 \leq \gamma$, matching pursuit iteratively updates the solution $w^{(i)}$ one weight at a time until a stop condition is met.

In order to better understand matching pursuit, we first express the penalized least-square problem of Eq. (2.1.22) as the following constrained least-square problem:

$$w^{(i)} = \underset{w^{(i)}}{\text{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 \qquad s.t. \qquad \|w^{(i)}\|_0 \leq \gamma\,. \tag{2.2.9}$$

Algorithm 2.3 Matching Pursuit (Mallat and Zhang, 1993)

Require: Dictionary D, observation $x^{(i)}$, upper bound γ and threshold ϵ.

1: Initialize the weight solution: $w^{(i)} = 0$
2: Initialize the residue: $r = x^{(i)}$
3: **while** $\|w^{(i)}\|_0 < \gamma$ **or** $\|r\|_2 > \epsilon$ **do**
4: Find the most correlated atom j with the residue r: $j \leftarrow \underset{j}{\operatorname{argmax}} \left| r^\top D_{:,j} \right|$
5: Update the weight solution: $w_j^{(i)} \leftarrow w_j^{(i)} + r^\top D_{:,j}$
6: Update the residual: $r \leftarrow r - w_j^{(i)} \cdot D_{:,j}$
7: **end while**
8: **return** $w^{(i)}$

In other words, the task is to minimize the norm of the *residue* $r = x^{(i)} - Dw^{(i)}$ subject to the constraint that the ℓ_0 norm of $w^{(i)}$ is upper-bounded by γ. The residue r is defined as the distance between observation $x^{(i)}$ and its estimate $Dw^{(i)}$, and the upper bound on the ℓ_0 norm of $w^{(i)}$ enforces sparsity.

We can envision the solution set of Eq. (2.2.9) as being divided in two regions. The first one contains solutions $w^{(i)}$ where $\|w^{(i)}\|_0 \leq \gamma$, and those solutions have an objective value given by the residual $\|x^{(i)} - Dw^{(i)}\|_2^2$. The second region contains solutions $w^{(i)}$ where $\|w^{(i)}\|_0 > \gamma$, and those solutions have an objective value of infinity.

There is a one-to-one correspondence between the penalized least-square problem of Eq. (2.1.22) and the constrained least-square problem of Eq. (2.2.9). For a given λ value, there is a corresponding γ such that $w^{(i)}$ is both a solution to Eq. (2.1.22) and Eq. (2.2.9). The penalized or constrained formulation does not change the solution set of the optimization problem. We only use it to make explicit that γ has a direct control on the maximum number of active atoms.

An overview of the matching pursuit approach by Mallat and Zhang (1993) is shown in Algorithm 2.3. The approach starts with a weight solution $w^{(i)} = 0$ (line 1) and a residue $r = x^{(i)} - Dw^{(i)} = x^{(i)}$ (line 2). The goal is to iteratively reduce the residual $\|r\|_2^2$ by updating the weight solution $w^{(i)}$ one position at a time. The position j to update is the one that has the highest correlation between its atom $D_{:,j}$ and the residue r (line 4):

$$j \leftarrow \underset{j}{\operatorname{argmax}} \left| r^\top D_{:,j} \right| \qquad \text{where} \qquad j \in \{1, 2, \dots\ d\}\,. \tag{2.2.10}$$

The weight solution $w^{(i)}$ is updated at position j by adding the value of the correlation

Algorithm 2.4 Orthogonal Matching Pursuit (Pati et al., 1993)

Require: Dictionary D, observation $x^{(i)}$, upper bound γ

1: Initialize the set of active atoms: $A = \emptyset$
2: Initialize residue: $r = x^{(i}$
3: **for** $t = 1$ **to** γ **do**
4: Find the most correlated atom j with the residue: $j \leftarrow \text{argmax}_j \left| r^\top D_{:,j} \right|$
5: Add j to the active set: $A \leftarrow A \cup \{j\}$
6: Let $\bar{D}$ be the dictionary of active atoms.
7: Find least square solution using $\bar{D}$: $\bar{w}^{(i)} = \underset{\bar{w}^{(i)}}{\text{argmin}} \left\| x^{(i)} - \bar{D}\bar{w}^{(i)} \right\|_2^2$
8: Compute new orthogonal residue: $r \leftarrow x^{(i)} - \bar{D}\bar{w}^{(i)}$
9: **end for**
10: Compute solution: $\begin{cases} w_j^{(i)} = \bar{w}_j^{(i)} & \text{when } j \in A \\ w_j^{(i)} = 0 & \text{when } j \notin A \end{cases}$
11: **return** $w^{(i)}$

(line 5):

$$w_j^{(i)} \leftarrow w_j^{(i)} + r^\top D_{:,j}\,, \tag{2.2.11}$$

as well as the residue, using the new value of the weight at j (line 6):

$$r \leftarrow r - w_j^{(i)} \cdot D_{:,j} \tag{2.2.12}$$

The algorithm stops when either $\|w^{(i)}\|_0$ as reached the upper bound γ or the residual $\|r\|_2^2$ has reached the lower bound ϵ.

One issue with Algorithm 2.3 is that a weight $w_j^{(i)}$ at a given position j can be updated more than one time. This is because all atoms are considered when searching for the position j to update in Eq. (2.2.10) (line 4). For instance, matching pursuit can select position j at the first iteration, then select position $j + 1$ at the second iteration, and select again position j at the third iteration. Thus, Algorithm 2.3 can iterate several times only to update the same position over and over again. This can lead to slow convergence, especially when using large dictionaries.

To overcome this limitation, Pati et al. (1993) proposed the Orthogonal Matching Pursuit (OMP) algorithm. An overview of the approach is presented in Algorithm 2.4. The approach starts with a weight solution $w^{(i)} = 0$ (line 1) and a residue $r = x^{(i)} - Dw^{(i)} = x^{(i)}$ (line 2). The goal is to iteratively reduce the residual $\|r\|_2^2$ by updating the weight solution $w^{(i)}$ one position at a time. The position j to update is the one that has the highest correlation between its atom $D_{:,j}$ and the residue r (line 4). The position j is

also added to the active set A (line 5), which is used to determine the dictionary $\bar{D}$ of active atoms (line 6). Then, the weight solution $\bar{w}^{(i)}$ of active atoms is computed using $\bar{D}$ (line 7) and is used to compute the new residue r (line 8). Finally, the approach iterates for γ iterations, which gives the final solution (line 10).

In contrast to the previous matching pursuit approach of Algorithm 2.3, OMP guarantees that $\|w^{(i)}\|_0 = \gamma$ after γ iterations. This is due to the way OMP updates the weight of the selected atom at position j. Instead of subtracting the correlation between the residue r and atom $D_{:,j}$ as in Eq. (2.2.11), the weight of the candidate atom is chosen such that the residue r becomes orthogonal to the basis defined by the active atoms $\bar{D}$ (line 7). This has the consequence that the correlation between the residue r and each active atom $\bar{D}_{:,j}$ becomes zero. Consequently, only inactive atoms can be selected as candidates when computing the correlation with Eq. (2.2.11) (line 4). Therefore, the ℓ_0 norm of the solution always increases by one after each iteration, which guarantees that $\|w^{(i)}\|_0 = \gamma$ after γ iterations.

Other approaches have been proposed to further improve matching pursuit and OMP. For instance, optimized OMP (OOMP) proposed another criterion to select the candidate atom j. Instead of using correlation that can lead to a sub-optimal choice when the atoms $D_{:,j}$ are not pair-wise orthogonal, OOMP rather proposed solving a *linear least square* optimization problem. Other approaches such as gradient pursuit (Blumensath and Davies, 2008), bag of pursuits (Labusch et al., 2011), stagewise OMP (Donoho et al., 2012), regularized OMP (Needell and Vershynin, 2009) and simultaneous OMP (Tropp et al., 2005) have also been proposed to improve the selection of the candidate.

2.2.3 Least Angle Regression

Least Angle Regression (LARS) is a greedy inference approach similar to matching pursuit (Efron et al., 2004). LARS is used to solve the penalized least-square problem of Eq. (2.1.22) under a ℓ_1 penalty. We present here the basic steps of the algorithm and refer the reader to (Efron et al., 2004) for more details. Starting with an initial weight solution $w^{(i)} = 0$, the first step is to find the most correlated atom j_1 with the residue $r = x^{(i)}$. We use notation j_t to make explicit the iteration t at which the atom was selected. This step is identical to the first step of matching pursuit in Eq. (2.2.10). However, the update of the weight solution $w^{(i)}$ is different. The value of the weight $w^{(i)}_{j_1}$ is increased in the direction of the sign of its correlation $r^\top D_{:,j_1}$, until another atom j_2 becomes as much correlated with the current residue as j_1. Then, LARS updates

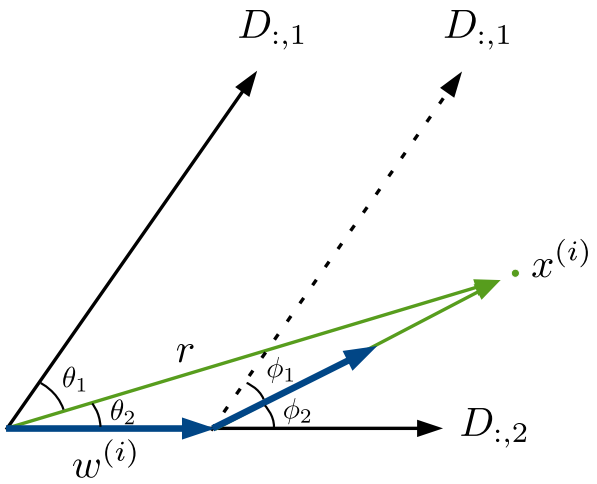


Figure 2.3: Illustration of LARS on a two-dimensional feature coding problem. The weight solution $w^{(i)}$ is colored in blue, the residue r in green and the dictionary atoms in black. At first, $D_{:,2}$ is more correlated with r than $D_{:,1}$ (since $\theta_1 > \theta_2$). The weight value $w_2^{(i)}$ is increased until both $D_{:,1}$ and $D_{:,2}$ becomes equally correlated with r (until $\phi_1 = \phi_2$). The weight solution $w^{(i)}$ is then updated by moving in the direction equiangular to $D_{:,1}$ and $D_{:,2}$ to preserve $\phi_1 = \phi_2$. The algorithm stops when $\|w^{(i)}\|_1 = \lambda$.

the weights $w_{j_1}^{(i)}$ and $w_{j_2}^{(i)}$ in the direction that preserves both the angle between j_1 and the residue, and the angle between j_2 and the residue. We call this direction the *joint least square direction*. The update continues until a third atom j_3 becomes as much correlated with the current residue as j_1 and j_2. These equiangular updates are repeated until $\|w^{(i)}\|_1 = \lambda$.

An illustration of LARS on a two-dimensional feature coding problem is shown in Fig 2.3. The weight solution $w^{(i)}$ is colored in blue, the residue r in green and the dictionary atoms in black. Since the dictionary atoms have a norm of one, the correlation between the residue r and an atom $D_{:,j}$ is given by the cosine of the angle between the two. At the start, $w^{(i)} = 0$ and $r = x^{(i)}$. We also have $\theta_1 > \theta_2$, which indicates that $D_{:,2}$ is more correlated with r than $D_{:,1}$. LARS then updates $w^{(i)}$ by increasing $w_2^{(i)}$ in direction $D_{:,2}$, as shown by the first blue arrow. The weight continues to increase until both $D_{:,1}$ and $D_{:,2}$ becomes equally correlated with the residue. This is shown with the angles ϕ_1 and ϕ_2 that have the same value. LARS then updates the weight solution $w^{(i)}$ by preserving $\phi_1 = \phi_2$. This is done by moving in the direction equiangular to $D_{:,1}$ and $D_{:,2}$, as shown by the second blue arrow.

LARS can be seen as a less greedy approach than matching pursuit. In matching pursuit, when a candidate atom j is selected, its associated weight value $w_j^{(i)}$ is updated so that all the explanatory power of atom $D_{:,j}$ is included in the weight solution $w^{(i)}$. We saw

this with OMP in step 7 of Algorithm 2.4, where $w^{(i)}$ was optimized so that the residue r became orthogonal with the basis of active atoms $\bar{D}$. This kind of weight update is fast and simple, but can also be too greedy (Efron et al., 2004). The selected atom j may have a smaller contribution in the global minimum solution than what would be computed by including its full explanatory power. The forward stage-wise regression of LARS tries to address this greediness by using partial updates. It finds the atom with the most explanatory power and slowly increases the weight solution $w^{(i)}$ by ϵ in the direction that preserves equiangularity. As a result, the selected active atoms contribute more equally.

2.2.4 Marginal Regression

Marginal Regression (also known as Soft-Thresholding) is a computationally simpler approach that performs *one-shot* optimization under a ℓ_1 penalty (Donoho and Johnstone, 1995). In contrast to previous approaches that iteratively update a candidate solution, marginal regression outputs a candidate solution in a single iteration.

The main difference lies in the formulation of the penalized least-square problem of Eq. (2.1.8). Instead of optimizing the ℓ_1-penalized residual value $\|x^{(i)} - Dw^{(i)}\| + \lambda\|w^{(i)}\|_1$ between an observation $x^{(i)}$ and its estimate $Dw^{(i)}$, it optimizes the following problem:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \frac{1}{2}\|D^\top x^{(i)} - w^{(i)}\|_2^2 + \lambda\|w^{(i)}\|_1 \,. \tag{2.2.13}$$

In other words, marginal regression tries to find the sparsest weight vector $w^{(i)}$ that approximates the vector $D^\top x^{(i)}$ of correlation between observation $x^{(i)}$ and each atom $D_{:,j}$.

One advantage of the formulation of Eq. (2.2.13) is that each weight $w_j^{(i)}$ appear in only one equation at a time. This allows to optimize Eq. (2.2.13) over each weight $w_j^{(i)}$ independently. We can then rewrite the minimization problem of Eq. (2.2.13) as d identical minimization problems:

$$w_j^{(i)} = \underset{w_j^{(i)}}{\operatorname{argmin}} \frac{1}{2}(D_{:,j}^\top x^{(i)} - w_j^{(i)})^2 + \lambda|w_j^{(i)}| \qquad \forall j \in \{1 \dots d\} \,. \tag{2.2.14}$$

We can solve Eq. (2.2.14) by equalling to zero the gradient with respect to $w_j^{(i)}$ and solving for $w_j^{(i)}$, but we have to take into consideration the non-differentiability of the absolute function at 0. We have to consider three cases: $w_j^{(i)} < 0$, $w_j^{(i)} > 0$ and $w_j^{(i)} = 0$.

When $w_j^{(i)} < 0$:

$$\frac{\partial}{\partial w_j^{(i)}} \frac{1}{2}(D_{:,j}^\top x^{(i)} - w_j^{(i)})^2 + \lambda |w_j^{(i)}| = -D_{:,j}^\top x^{(i)} + w_j^{(i)} - \lambda = 0 \tag{2.2.15}$$

$$w_j^{(i)} = D_{:,j}^\top x^{(i)} + \lambda \tag{2.2.16}$$

And since $w_j^{(i)} < 0$, we also have:

$$w_j^{(i)} = D_{:,j}^\top x^{(i)} + \lambda < 0 \tag{2.2.17}$$

$$D_{:,j}^\top x^{(i)} < -\lambda \tag{2.2.18}$$

When $w_j^{(i)} > 0$:

$$\frac{\partial}{\partial w_j^{(i)}} \frac{1}{2}(D_{:,j}^\top x^{(i)} - w_j^{(i)})^2 + \lambda |w_j^{(i)}| = -D_{:,j}^\top x^{(i)} + w_j^{(i)} + \lambda = 0 \tag{2.2.19}$$

$$w_j^{(i)} = D_{:,j}^\top x^{(i)} - \lambda \tag{2.2.20}$$

And since $w_j^{(i)} > 0$, we also have:

$$w_j^{(i)} = D_{:,j}^\top x^{(i)} - \lambda > 0 \tag{2.2.21}$$

$$D_{:,j}^\top x^{(i)} > \lambda \tag{2.2.22}$$

In summary, the optimal solution of Eq. (2.2.14) is defined as:

$$w_j^{(i)} = \begin{cases} D_{:,j}^\top x^{(i)} - \lambda & \text{if} \quad D_{:,j}^\top x^{(i)} > \lambda \\ D_{:,j}^\top x^{(i)} + \lambda & \text{if} \quad D_{:,j}^\top x^{(i)} < -\lambda \qquad \forall j \in \{1 \dots d\} \\ 0 \qquad\qquad \text{otherwise} \end{cases} \tag{2.2.23}$$

$$= \text{sign}(D_{:,j}^\top x^{(i)}) \max\left\{0, |D_{:,j}^\top x^{(i)}| - \lambda\right\} \tag{2.2.24}$$

$$= \mathcal{T}_\lambda \left(D_{:,j}^\top x^{(i)}\right) , \tag{2.2.25}$$

where $\mathcal{T}$ is the soft-thresholding function (Donoho and Johnstone, 1995). In vectorized form, the weight solution $w^{(i)}$ can be expressed as:

$$w^{(i)} = \mathcal{T}_\lambda \left(D^\top x^{(i)}\right) , \tag{2.2.26}$$

where $\mathcal{T}$ is applied element-wise.

Marginal regression has a computational complexity of $O(nd)$, where d is the number of dictionary atoms and n is the dimensionality of the observation $x^{(i)}$. Due to its one-shot

Table 2.1: Summary of the presented feature coding approaches. As a reminder, d is the number of dictionary atoms $D_{:,j}$ and n is the dimensionality of the input observation $x^{(i)}$.

Name		Complexity	Sparsity	Reference
Feature Sign Search	FSS	$\Omega(d^3 + nd^2)$	ℓ_1	Algorithm 2.2
Matching Pursuit	MP	$\Omega(nd)$	ℓ_0	Algorithm 2.3
Orthogonal Matching Pursuit	OMP	$\Theta(\gamma nd + \gamma^4 + n\gamma^3)$	ℓ_0	Algorithm 2.4
Least Angle Regression	LARS	$O(d^3 + nd^2)$	ℓ_1	Figure 2.3
Marginal Regression (Soft-Thresholding)	ST	$\Theta(nd)$	ℓ_1	Eq. (2.2.26)

optimization nature, it is not computationally demanding and can be used with large dictionary. Marginal regression can obtain similar performance as the other approaches even though it does not solve the feature coding problem as defined in the standard sparse dictionary learning framework (see (Genovese et al., 2012) for a recent experimental evaluation). For this reason, it has been applied in many contexts (Donoho, 2006; Fan and Lv, 2008; Balasubramanian et al., 2013; Hung et al., 2014)

2.2.5 Summary

A summary of the presented feature coding approaches is shown in Table 2.1. The first column contains the name of each approach, the second column contains the worst-case computational complexity, the third column contains the sparsity penalty and the last column contains a reference to the approach in the text. For the computational complexity, O refers to the standard Big O notation (for bounded above), Θ refers to the Big Theta notation (bounded below and above) and Ω refers to the standard Big Omega notation (for bounded below).

As we can see in Table 2.1, the worst-case computational complexity of both FSS and MP is defined using the Big Omega notation. This is due to a limitation of both approaches that have a problem-dependent stopping criterion. Indeed, FSS stops only when the conditions of Eq. (2.2.7) and Eq. (2.2.8) are met, while MP stops only when either $\|w^{(i)}\|_0$ has reached the upper bound γ or the residual $\|r\|_2^2$ has reached the lower bound ϵ. As a result, we can only describe a lower bound on the complexity. For FSS, the bound is determined by the inversion in Eq. (2.2.6), while for MP, the bound is determined by the search for the most correlated atom in Eq. (2.2.10).

In the case of OMP and ST, their computational complexity is defined using the Big Theta notation. This is due to their greedy nature that ensures a fixed number of iterations. For OMP, the algorithm guarantees that $\|w^{(i)}\|_0 = \gamma$ after γ iterations, while for ST, the algorithm is *one-shot*, meaning that it is executed only once. The bound of OMP is determined by the least-square optimization problem of Line 7 of Algorithm 2.1 (see Sturm and Christensen (2012) for the details on how to compute the bound), while the bound of ST is determined by the matrix operation of Eq. (2.2.26).

As for LARS, its computational complexity is defined using the Big O notation. This is because the sum of computations for a complete sequence of *joint least square directions* is the same as a least-square optimization problem, even though the algorithm is iterative (ref. Section 7 of Efron et al. (2004)). As a result, LARS has a worst-case computational complexity of $O(d^3 + nd^2)$. Moreover, in practice, the λ penalty will stop the computation before reaching the end of the sequence. The computational complexity can therefore be lowered down by opting for a smaller λ (to the detriment of a sub-optimal solution). This makes LARS an appealing feature coding approach for the ℓ_1 penalty because LARS has a good trade off between computational complexity and solution optimality.

2.3 A Review of Important Alternating Minimization Approaches for Dictionary Learning

The goal of dictionary learning is to learn a dictionary D such that feature coding can find weight solutions $w^{(i)}$ where both the residual and the penalty term are small, for all inputs $x^{(i)}$. We have seen in Eq. (2.1.16) that this problem can be defined as a nested optimization over both D and $w^{(i)}$, and can be tackled with alternating minimization as illustrated in Algorithm 2.1. In this section, we review important works on dictionary learning based on this scheme. We begin with Method of Optimal Directions, then Gain Shape Vector Quantization, Gradient Descent and finally Online Dictionary Learning. For a more detailed review on other dictionary learning approaches, see Tosic and Frossard (2011) and Mairal et al. (2014).

Algorithm 2.5 Solving Eq. (2.1.16) with Method of Optimal Direction.

Require: Observations $X = \{x^{(i)}\}_{i=1}^{N}$, penalty λ and number of iterations T.

1: Initialize dictionary D: $D_{:,j} \sim Uniform(0,1)$, $\forall j \in \{1 \dots d\}$

2: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

3: **for** $t = 1 \dots T$ **do**

4: Perform feature coding:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda \|w^{(i)}\|_1 \qquad \forall i \in \{1 \dots N\}$$

5: Update the dictionary: $D = XW^\top \left(WW^\top\right)^{-1}$

6: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

7: **end for**

8: **return** D

2.3.1 Method of Optimal Directions

Method of Optimal Directions (MOD) (Engan et al., 1999) solves the least-square optimization of Eq. (2.1.18) directly. Let $X = [x^{(1)} \dots x^{(N)}]$ be the observation matrix containing in its columns the observations $x^{(i)}$ and $W = [w^{(1)} \dots w^{(N)}]$ be the weight matrix containing in its columns the weights $w^{(i)}$. By expressing Eq. (2.1.18) in a matrix form, we have the following optimization:

$$D = \underset{D}{\operatorname{argmin}} \|X - DW\|_F^2 , \tag{2.3.1}$$

where $\|\cdot\|_F^2$ is the squared Frobenius norm. Due to the convexity of the Frobenius norm, the global minimum of Eq. (2.1.18) can be found by finding the zeros of the gradient of $\|X - DW\|_F^2$ with respect to D:

$$\frac{\partial}{\partial D}\|X - DW\|_F^2 = 0 \tag{2.3.2}$$

$$-2\left(X - DW\right)W^\top = 0 \tag{2.3.3}$$

$$D = XW^\top \left(WW^\top\right)^{-1} . \tag{2.3.4}$$

An overview of the approach is shown in Algorithm 2.5. The algorithm starts by initializing the dictionary D (line 1) and by constraining the dictionary atoms to the unit hyper-sphere (line 2). Then, the following three steps are performed for T iterations. First, perform feature coding on all observations $x^{(i)}$ (line 4). Second, update the dictionary D to its least-square solution (line 5). Third, constrain the dictionary atoms to

the unit hyper-sphere (line 6). The algorithm ends at line 8 by returning the learned dictionary D.

The main inconvenient of MOD is the computational complexity of the inverse operation. As seen in Eq. (2.3.4), computing the global solution requires inverting the $d \times d$ matrix $\left(WW^\top\right)^{-1}$. A standard Gauss-Jordan elimination is $O(d^3)$, which means that MOD can only be applied when the number of dictionary atoms d is not too large. For this reason, MOD is rarely used in practice.

2.3.2 Gain Shape Vector Quantization

Gain Shape Vector Quantization (GSVQ) (Coates and Ng, 2012; Varma and Zisserman, 2009; Dhillon and Modha, 2001) performs dictionary learning using damped updates. At each iteration, GSVQ updates the dictionary by solving the least-square problem of Eq. (2.1.18) with an additional penalty term that encourages the new dictionary $\bar{D}$ to be similar to the current dictionary D. The task can be defined as follows:

$$D \leftarrow \operatorname*{argmin}_{\bar{D}} \|X - \bar{D}W\|_F^2 + \|\bar{D} - D\|_F^2 \tag{2.3.5}$$

$$= \left(WW^\top + I\right)^{-1} \left(XW^\top + D\right) \tag{2.3.6}$$

$$\propto D + XW^\top, \tag{2.3.7}$$

$$D_{:,j} \leftarrow \frac{D_{:,j}}{\|D_{:,j}\|_2} \quad \forall j \in \{1 \dots d\}. \tag{2.3.8}$$

Therefore, GSVQ updates dictionary D by adding the sum of outer products between each $x^{(i)}$ and $w^{(i)}$. Removing the term $\left(WW^\top + I\right)^{-1}$ is a form of damping that helps the less active atoms to be more frequently updated. This helps prevent atoms from *dying*, which happens when the atoms are always inactive. It also lowers the computational burden of inverting the matrix $WW^\top + I$. The dictionary update is finally followed by a projection on the unit hyper-sphere (Eq. (2.3.8)), as done previously.

One particularity of GSVQ is that it usually defines the feature coding approach as Orthogonal Matching Pursuit (see Algorithm 2.4) with a sparsity term of $\gamma = 1$. In other words, the weight solution $w^{(i)}$ is defined as a vector of all zeros except at position j, where j is the atom $D_{:,j}$ that is the most correlated with observation $x^{(i)}$. In this setting, performing feature coding is straightforward and can be expressed as follows:

$$w_j^{(i)} = \begin{cases} D^{(j)\top} x^{(i)} & \text{if } j = \operatorname*{argmax}_j \left| D^{(j)\top} x^{(i)} \right| \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \{1 \dots d\}. \tag{2.3.9}$$

Algorithm 2.6 Solving Eq. (2.1.16) with Gain Shape Vector Quantization

Require: Observations $X = \{x^{(i)}\}_{i=1}^{N}$ and number of iterations T.

1: Initialize dictionary D: $D_{:,j} \sim Uniform(0,1)$, $\forall j \in \{1 \dots d\}$

2: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

3: **for** $t = 1 \dots T$ **do**

4: Perform feature coding on all $x^{(i)}$:

$$w_j^{(i)} = \begin{cases} D^{(j)\top} x^{(i)} & \text{if } j = \underset{j}{\operatorname{argmax}} \left| D^{(j)\top} x^{(i)} \right| \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \{1 \dots d\}, \forall i \in \{1 \dots N\}$$

5: Update dictionary: $D \leftarrow D + \sum_{i=1}^{N} x^{(i)} \cdot w^{(i)\top}$

6: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

7: **end for**

8: **return** D

A description of the approach is presented in Algorithm 2.6. The algorithm starts by initializing the dictionary D (line 1) and by constraining the dictionary atoms to the unit hyper-sphere (line 2). Then, the following three steps are performed for T iterations. First, perform feature coding on all observations $x^{(i)}$ (line 4). If Orthogonal Matching Pursuit with a sparsity term $\gamma = 1$ is chosen, this step amounts to performing Eq. (2.3.9). Second, update the dictionary D by applying the updates of Eq. (2.3.7) (line 5). Third, constrain the dictionary atoms to the unit hyper-sphere (line 6). The algorithm ends at line 8 by returning the learned dictionary D.

2.3.3 Gradient Descent

Gradient descent is one of the most common iterative methods in optimization. Even though it has been proposed decades ago, Olshausen et al. (1996) were the first to apply it in the context of dictionary learning. The idea underlying gradient descent is straightforward. First, compute the gradient of the objective function with respect to its parameters, then update the parameters in the direction of the negative gradient by moving a small step. The length of the step is usually governed by a hyper-parameter called the learning rate $\alpha > 0$. The approach alternates these two phases until convergence.

In the context of dictionary learning, the loss function is the least-square optimization

Algorithm 2.7 Solving Eq. (2.1.16) with Gradient Descent

Require: Observations $X = \{x^{(i)}\}_{i=1}^{N}$, penalty λ, learning rate α and number of iterations T.

1: Initialize dictionary D: $D_{:,j} \sim Uniform(0, 1)$, $\forall j \in \{1 \dots d\}$

2: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

3: **for** $t = 1 \dots T$ **do**

4: Perform feature coding:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda \|w^{(i)}\|_1 \qquad \forall i \in \{1 \dots N\}$$

5: Update dictionary: $D \leftarrow D + \dfrac{\alpha}{N} \sum_{i=1}^{N} \left(x^{(i)} - Dw^{(i)}\right) {w^{(i)}}^{\top}$

6: Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$

7: **end for**

8: **return** D

of Eq. (2.1.18). Thus, the gradient is as follows:

$$\frac{\partial}{\partial D} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2 = -2 \sum_{i=1}^{N} \left(x^{(i)} - Dw^{(i)}\right) {w^{(i)}}^{\top}. \tag{2.3.10}$$

In other words, the gradient of the objective with respect to D is the sum of the gradient of the residual $\|x^{(i)} - Dw^{(i)}\|_2^2$ evaluated at each observation $x^{(i)}$. It is common to divide the sum by the number of observations N in order to avoid numerical instabilities, and also to absorb the constant 2 in the learning rate α for simplicity. Therefore, the update of the dictionary D can be defined as follows:

$$D \leftarrow D + \frac{\alpha}{N} \sum_{i=1}^{N} \left(x^{(i)} - Dw^{(i)}\right) {w^{(i)}}^{\top}, \tag{2.3.11}$$

where the dictionary D is updated by subtracting the gradient of Eq. (2.3.10) scaled by $\dfrac{\alpha}{N}$. To keep the constraint on the norm of the dictionary atoms $\|D_{:,j}\|_2 = 1$, the update is followed by a projection on the unit hyper-sphere. The projection operation is simply the normalization of each dictionary atom:

$$D_{:,j} \leftarrow \frac{D_{:,j}}{\|D_{:,j}\|_2} \qquad \forall j \in \{1 \dots d\}. \tag{2.3.12}$$

A description of the approach is presented in Algorithm 2.7. The algorithm starts by initializing the dictionary D (line 1) and by constraining the dictionary atoms to

the unit hyper-sphere (line 2). Then, the following three steps are performed for T iterations. First, perform feature coding on all observations $x^{(i)}$ (line 4). Second, update the dictionary D in the direction of the negative gradient (line 5). Third, constrain the dictionary atoms to the unit hyper-sphere (line 6). The algorithm ends at line 8 by returning the learned dictionary D.

One advantage of gradient descent is that it does not suffer from the computational burden of matrix inversion that MOD has. Indeed, the gradient is computed in $O(d^2)$ by a simple matrix product, which is an order of magnitude smaller than matrix inversion.

One drawback of gradient descent is the presence of the hyper-parameter α. As shown in Algorithm 2.7, α controls the size of the step in the direction of the negative gradient. It has a great importance on the speed of convergence. One could be tempted to use a large α in order to quickly converge to the global minimum D^*. However, when the candidate solution D approaches the global minimum D^*, the updates will repetitively overshoot and cause random oscillations around D^*. The candidate D never truly converges to D^*, but rather converges in a neighborhood of D^* with a size proportional to α. When α is unreasonably too large, the updates can even increase the distance between D and D^* and make the candidate D diverge to infinity.

The hyper-parameter α must then be constrained to ensure convergence. A necessary condition is to ensure α complies with the Robbins-Monro conditions (Robbins and Siegmund, 1985). The conditions are expressed as follows:

$$\sum_t \alpha_t^2 < \infty \qquad \text{and} \qquad \sum_t \alpha_t = \infty \,, \tag{2.3.13}$$

where we use subscript α_t to make explicit the value of α at iteration t. One particular sequence of α_t suggested by Robbins-Monro has the following form:

$$\alpha_t = \frac{\alpha_0}{\beta \cdot t} \,, \tag{2.3.14}$$

where $\alpha_0 > 0$ is the initial learning rate and $\beta \in [0, 1]$ is the learning rate decay (Murata, 1999). In other words, it suggests that the learning rate α should start at a large initial value α_0 and decrease at each iteration t proportional to the learning rate decay β. The learning rate decay β must be chosen such that the learning rate α_t does not decrease too rapidly, otherwise thee candidate D will never reach the global minimum D^*. The proper value of β that ensures this condition is however undetermined and problem-dependent.

2.3.4 Online Dictionary Learning

Online Dictionary Learning (ODL) (Mairal et al., 2010) takes the alternating minimization scheme of Algorithm 2.1 a step further by defining a *block coordinate descent* scheme. Unlike standard alternating minimization that defines two blocks (in our case, the dictionary D is one block and the weights $\{w^{(i)}\}_{i=1}^N$ is the second one), ODL defines $d+1$ blocks, one block for the weights and one block for each dictionary atom $D_{:,j}$. The advantage of this scheme is that it removes the need to define a learning rate sequence α_t. Indeed, the authors Mairal et al. (2010) has shown that they can remove the need to define a learning rate sequence by employing an *online* gradient descent approach. We now explain how this can be achieved.

ODL can be derived by computing the zeros of the gradient of Eq. (2.1.18) with respect to only one dictionary atom $D_{:,j}$. First, we compute the gradient:

$$0 = \frac{\partial}{\partial D_{j,:}} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2 \tag{2.3.15}$$

$$0 = \sum_{i=1}^{N} \left(x^{(i)} - Dw^{(i)}\right) w_j^{(i)} \tag{2.3.16}$$

Then, we can extract weight $w_j^{(i)}$ and dictionary atom $D_{:,j}$ from the weighted sum $Dw^{(i)}$ in order to isolate $D_{:,j}$:

$$0 = \sum_{i=1}^{N} \left(x^{(i)} - D_{:,j} w_j^{(i)} - \sum_{k \neq j} D_{:,k} w_k^{(i)}\right) w_j^{(i)} \tag{2.3.17}$$

$$\sum_{i=1}^{N} D_{:,j} w_j^{(i)} w_j^{(i)} = \sum_{i=1}^{N} \left(x^{(i)} - \sum_{k \neq j} D_{:,k} w_k^{(i)}\right) w_j^{(i)} \tag{2.3.18}$$

$$D_{:,j} = \frac{1}{\sum_{i=1}^{N} w_j^{(i)} w_j^{(i)}} \sum_{i=1}^{N} \left(x^{(i)} - \sum_{k \neq j} D_{:,k} w_k^{(i)}\right) w_j^{(i)} \tag{2.3.19}$$

We can then distribute the weight $w_j^{(i)}$ inside the parenthesis:

$$D_{:,j} = \frac{1}{\sum_{i=1}^{N} w_j^{(i)} w_j^{(i)}} \sum_{i=1}^{N} \left(x^{(i)} w_j^{(i)} - \sum_{k \neq j} D_{:,k} w_k^{(i)} w_j^{(i)}\right) \tag{2.3.20}$$

and also distribute the sum over i:

$$D_{:,j} = \frac{1}{\sum_{i=1}^{N} w_j^{(i)} w_j^{(i)}} \left(\sum_{i=1}^{N} \left(x^{(i)} w_j^{(i)}\right) - \sum_{k \neq j} D_{:,k} \sum_{i=1}^{N} \left(w_k^{(i)} w_j^{(i)}\right)\right) . \tag{2.3.21}$$

We now see that there are terms that do not depend on $D_{:,j}$. By defining the accumulator matrices:

$$A = \sum_{i=1}^{N} w^{(i)} w^{(i)^\top} , \tag{2.3.22}$$

$$B = \sum_{i=1}^{N} x^{(i)} w^{(i)^\top} , \tag{2.3.23}$$

we can rewrite Eq. (2.3.21) as follows:

$$D_{:,j} = \frac{1}{A_{j,j}} \left(B_{:,j} - DA_{:,j} + D_{:,j} A_{j,j} \right) . \tag{2.3.24}$$

We find the following equality by distributing $\frac{1}{A_{j,j}}$:

$$D_{:,j} = D_{:,j} + \frac{1}{A_{j,j}} \left(B_{:,j} - DA_{:,j} \right) , \tag{2.3.25}$$

from which we can define the update rule of ODL:

$$D_{:,j} \leftarrow D_{:,j} + \frac{1}{A_{j,j}} \left(B_{:,j} - DA_{:,j} \right) . \tag{2.3.26}$$

A description of the approach is presented in Algorithm 2.8. The algorithm starts by initializing the dictionary D (line 1) and by constraining the dictionary atoms to the unit hyper-sphere (line 2). It also initializes two accumulators A and B to zero. Then, the following steps are performed for T iterations and for each observation $x^{(i)}$. First, perform feature coding on observation $x^{(i)}$ (line 6). Second, update the accumulators following Eq. (2.3.22) for A and Eq. (2.3.23) for B (line 7). Third, update each dictionary atom sequentially by applying the update of Eq. (2.3.26) and by constraining it to the unit hyper-sphere (line 10). The algorithm ends at line 14 by returning the learned dictionary D.

The authors Mairal et al. (2010) provided a sketch of the proof that showed that their ODL algorithm converges almost surely. We refer the reader to their paper for the details.

As seen in Algorithm 2.8, ODL does not define a learning rate sequence α_t. In fact, ODL uses a learning rate of $\frac{1}{A_{j,j}}$ for each dictionary atom j. The learning rate sequence is not defined by hand, but rather given by the exponential moving average updates on accumulator A. A closer comparison between the update rule of ODL and standard

Algorithm 2.8 Solving Eq. (2.1.16) with Online Dictionary Learning

Require: Observations $X = \{x^{(i)}\}_{i=1}^{N}$, penalty λ, decay β and number of iterations T.

Initialize dictionary D: $D_{:,j} \sim Uniform(0,1)$, $\forall j \in \{1 \dots d\}$
Constrain the norm: $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$, $\forall j \in \{1 \dots d\}$
Initialize accumulators: $A = 0$, $B = 0$.
for $t = 1 \dots T$ **do**
  for each $x^{(i)} \in X$ **do**
    Perform feature coding: $w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda \|w^{(i)}\|_1$
    Update accumulators:
    $A \leftarrow \beta A + (1-\beta) w^{(i)} w^{(i)^\top}$, and $B \leftarrow \beta B + (1-\beta) x^{(i)} w^{(i)^\top}$
    for $j = 1 \dots d$ **do**
      $D_{:,j} \leftarrow D_{:,j} + \dfrac{1}{A_{jj}}(B_{:,j} - DA_{:,j})$
      $D_{:,j} \leftarrow \dfrac{D_{:,j}}{\|D_{:,j}\|_2}$
    end for
  end for
end for
return D

gradient descent reveals why this is the case. As we have seen in the previous section, the update rule of gradient descent was defined in Eq.(2.3.11) as:

$$D \leftarrow D + \frac{\alpha}{N} \sum_{i=1}^{N} \left(x^{(i)} - Dw^{(i)}\right) w^{(i)^\top}. \tag{2.3.27}$$

After distributing $w^{(i)^\top}$ and the sum in the parenthesis, we can write:

$$D \leftarrow D + \frac{\alpha}{N} \left(\sum_{i=1}^{N} \left(x^{(i)} w^{(i)^\top}\right) - D \sum_{i=1}^{N} \left(w^{(i)} w^{(i)^\top}\right) \right). \tag{2.3.28}$$

Using the definition of accumulator A in Eq. (2.3.22) and B in Eq. (2.3.23), we can write:

$$D \leftarrow D + \frac{\alpha}{N} \left(B - DA\right). \tag{2.3.29}$$

As we can see, the update rule of standard gradient descent in Eq. (2.3.29) is closely related to the update rule of ODL in Eq. (2.3.26). In both cases, the atoms are updated using $B - DA$, but each approach uses a different step size. For standard gradient descent, the step size is given by the learning rate α divided by the number of observations

Table 2.2: Summary of the presented dictionary learning approaches. As a reminder, d is the number of dictionary atoms $D_{:,j}$, n is the dimensionality of the observations $x^{(i)}$, N is the number of observations and T is the number of iterations.

Name		Complexity	Reference
Method of Optimal Direction	MOD	$O(Nd^2 + d^3 + TN\delta)$	Algorithm 2.5
Gain Shape Vector Quantization	GSVQ	$\Theta(TNnd)$	Algorithm 2.6
Gradient Descent	GD	$O(TNnd + TN\delta)$	Algorithm 2.7
Online Dictionary Learning	ODL	$O(TNnd^2 + TN\delta)$	Algorithm 2.8

N, while for ODL, the step size is given by the inverse of $A_{j,j}$. The main advantage of ODL over standard gradient descent is that A is computed automatically from the data. There is no need to define a learning rate sequence. For this reason, ODL is often chosen instead of standard gradient descent.

2.3.5 Summary

A summary of the presented feature coding approaches is shown in Table 2.2. The first column contains the name of each approach, the second column contains the worst-case computational complexity while the last column contains a reference to the approach in the text. For the computational complexity, O refers to the standard Big O notation (for bounded above) and Θ refers to the Big Theta notation (bounded below and above). Moreover, we use δ to refer to the computational complexity of feature coding when describing the computational complexity of each dictionary learning approach.

As we can see in Table 2.2, the computational complexity of each approach contains the number of observations N. The term N appears because of the need to perform feature coding over all N observations during each iteration. For this reason, these approaches are rarely used *as they are* when the number of observations N is large, since it would be too computationally demanding. Indeed, GSVQ, GD and ODL are usually adapted to only using a subset of the available observations during each iteration rather than the full N observations. In this case, the dictionary is updated using only the observations from a smaller *mini-batch* of size B, where the B observations selected during each iteration are chosen at random. For GSVQ, this corresponds to performing the summation in line 5 of Algorithm 2.6 over only B observations. For GD, this corresponds to performing the summation in line 5 of Algorithm 2.7 over only B observations. For ODL, this corresponds to performing the for-loop in line 5 of Algorithm 2.8 over only B observations.

The use of only B observations improves training in two ways. First, it reduces the need to perform N feature coding steps during each T iteration. Instead, we only need to perform B feature coding steps. Second, it adds noise to the dictionary updates. This could be considered as harmful during minimization, but it is rather helpful. The reason explaining why it is helpful is beyond the scope of this thesis, so we refer to these works for more detail (Collet and Rennard, 2007; Bottou, 2010; Goodfellow et al., 2016).

Regarding convergence, we have seen in Section 2.1.2 that the alternating minimization scheme, from which Algorithm 2.1 is based, always converges to a stationary point (Xu and Yin, 2013). This means that if the number of iterations T is sufficiently large, MOD, GSVQ and GD will also converge to a stationary point, since they comply to the form of Algorithm 2.1. As for ODL, the authors Mairal et al. (2010) provided a sketch of the proof that showed that their algorithm converges almost surely.

However, optimization under sparsity constraints is susceptible to the problem of *dead atoms*. This problem can appear with every dictionary learning technique, thus including the approaches in Table 2.2. Dead atoms happen when some of the dictionary atoms $D_{:,j}$ are no more used during feature coding over all N observations. An inactive atom has a weight $w_j^{(i)}$ of zero, which means that the atom $D_{:,j}$ is no more updated during dictionary learning. This problem usually happens early during the optimization process and is dependent on initialization (Mairal et al., 2010).

Recovering from dead atoms is difficult for MOD, GSVQ and GD. This is due to their iterative nature that makes the dictionary update at iteration t dependent on the dictionary at iteration t only. As soon as a dictionary atom $D_{:,j}$ dies, it will stop being updated because the other atoms will always be selected during feature coding. This problem is far less present in ODL. The dictionary updates in ODL use two accumulators A and B that *accumulate* the information from the past iterations. This means that the dictionary update at iteration t not only uses the information at iteration t, but also at iterations $t-1$, $t-2$ up to iteration 1. Thus, as long as an atom becomes active at least once during the minimization process, it is ensured to never die. As a result, dead atoms are almost impossible with ODL. For this reason, ODL is usually the dictionary learning approach of choice when considering MOD, GSVQ and GD, because ODL lowers the likelihood of dead atoms and the need to deal with them.

2.4 Conclusion

In this chapter, we reviewed Sparse Dictionary Learning. We started with a definition of the framework from the formulation of the generative model and elaborated on the notion of sparsity. We then reviewed important work for both feature coding and dictionary learning. For feature coding, we focused on approaches using the ℓ_0 and ℓ_1 penalties, since they are the most often used in practice. We detailed Feature Sign Search, Matching Pursuit, Orthogonal Matching Pursuit, Least Angle Regression and Marginal Regression. For dictionary learning, we focused on approaches using alternating minimization to solve the nested optimization problem. We detailed Method of Optimal Directions, Gain Shape Vector Quantization, Gradient Descent and Online Dictionary Learning.

In the following chapter, we will introduce another framework for representation learning that focuses on training deep neural networks. We will introduce the framework of *Deep Learning*.

Chapter 3

Background Material on Deep Learning

In this chapter, we present background material on Deep Learning, which implements the *distributed*, the *simplicity of factor dependencies* and the *hierarchical* prior beliefs that we introduced in Section 1.2.2, 1.2.3 and 1.2.5. We start in Section 3.1 with a definition of the framework. In particular, we detail how the design of feedforward deep neural networks is motivated by the idea of parametric function approximation. Then, we present in Section 3.2 the importance of the activation function. We review important aspects to consider when choosing an activation function, such as vanishing gradients, bias shift and dead units. Then, we review the family of parametric activation functions in Section 3.3. We detail important works that learn the shape of the activation function. Finally, we present a review of key deep network architectures in Section 3.4, such as AlexNet, ResNet and DenseNet. We conclude the chapter in Section 3.5.

3.1 Definitions

Deep Learning is a powerful framework for machine learning, with a long-standing history dating back at least to the 1950s (Rosenblatt, 1958). It is based on the central concept of *parametric function approximation* that is behind nearly all modern practical applications of deep learning. The goal of parametric function approximation is to learn a parametric function $f(x;\theta)$ to approximate a target unknown function f^*. The target function f^* is unknown in the sense that it is extremely difficult to directly describe f^* in analytic form. This is because f^* usually represents a complex mapping problem that we would like to solve, but we do not have the tools nor enough understanding of

the problem to directly implement f^* as a working solution.

Take as an example the problem of image captioning (Xu et al., 2015). The goal of image captioning is to provide a textual description of the content of an image. The input observation x is defined as the image, the output value $f^*(x)$ is defined as the textual description of x and the target function f^* is defined as the mapping that transforms an input image into a text. We can already see that providing an analytic description of f^* would be extremely difficulty. We would need to dig deep into natural language processing and computer vision frameworks to provide a description on how groups of pixels can be interpreted as groups of alphanumeric characters that make up a logical description of the content of the image. This would be a hard task.

The fact that f^* cannot be described in analytic form is what makes parametric function approximation necessary, but it is also what makes it challenging. We cannot use an analytic description of f^* to implement the function, because no such description exists or is available. We must instead use a parametric function $f(x;\theta)$ and adjust its parameters θ such that the behavior of $f(x;\theta)$ matches the behavior of f^* as much as possible, i.e. $f(x;\theta) \approx f^*(x)$. This way, the output value of the parametric function $f(x;\theta)$ evaluated at all x is as similar as possible to the output value of the target function $f^*(x)$.

One of the best way to obtain information about the structure of f^* without an analytic description is to use input-output pairs (x, y), where $y \approx f^*(x)$. Pairs like this provide a noisy and approximate view of the true function f^* around each input observation x. Considering the problem of image captioning, a pair (x, y) would be defined as an image x with a textual description y that would be provided by a person. The textual description y is considered a noisy view of the true function $f^*(x)$ because the person can make mistakes. Nonetheless, the mistakes can be averaged out by using many input-output pairs and still be able to get a good approximation of the true function $f^*(x)$.

The ability of the parametric function f to approximate the target function f^* depends to a large extent on the architecture of f. In current established deep learning practices, f is defined as a *feedforward deep neural network*. We now detail what is meant by *feedforward*, what is meant by *deep*, and what is meant by *neural network*.

Feedforward

The architecture is qualified as *feedforward* because information flows in a single direction: from the input observation x to the output value $f(x;\theta)$. There are no feedback connections that feed output values back as input values.

Deep

The architecture is also called *deep* because it is typically presented as a composition of more than one function. The overall network can be decomposed as a directed acyclic graph that describes how the functions are composed together. For instance, f could be defined as a chained architecture $f_L(f_{L-1}(\ldots))$ of L *layers*. An example of a deep network with a chained architecture of three functions can be defined as follows:

$$f(x;\theta) = f_3(f_2(f_1(x;\theta_1);\theta_2);\theta_3)\,, \tag{3.1.1}$$

where each function f_l has different parameters θ_l, which can be adjusted separately, and $\theta = \{\theta_1, \theta_2, \ldots, \theta_L\}$ is the set of all parameters.

The chained architecture of Eq. (3.1.1) is what gives deep networks their representational complexity. Adding more functions f_l or parameters θ_l has been shown to increase the complexity of functions that a deep network can approximate (Hornik et al., 1989; Cybenko, 1989). It is simple and can be used effectively to scale up deep networks to more complex problems. As a result, it is often seen when designing a deep neural network.

Moreover, function composition is compatible with the *hierarchical* prior belief. As we saw in Section 1.2, the hierarchical prior belief suggests taking advantage of the fact that hierarchical organizations are often seen in nature. The use of function composition creates such a hierarchical organization of the layers that is in line with this prior belief. This encourages each individual function f_l to adjust their parameterization θ_l to find the underlying hierarchical organization of the target function f^* that is the most natural. As a result, deep neural networks often learn to approximate the target function f^* as a hierarchy of increasingly abstract concepts (Zeiler and Fergus, 2014).

Neural Network

Finally, the architecture is qualified as a *neural network* because it is inspired by neuroscience. In particular, it is inspired by the way *biological* neurons receive, process, and transmit information between them. A neuron receives electrical and chemical signals

from other neurons that alter its cross-membrane voltage. A large increase of voltage can cause the neuron to emit an all-or-none electrochemical pulse called an action potential that will be sent to other adjacent neurons. The action potential transits to other neurons via specialized connections called synapses. Together, neurons and synapses form a neural circuit, which is the primary component of the central nervous system.

Neurons in a deep neural network are however very different from biological ones. Modern neural network research is nowadays guided mainly by mathematical principles and engineering techniques rather than neuroscience. This is because the initial goal of trying to imitate how information is processed in the brain has now changed to trying to design novel function approximation machines to attain better statistical generalization. On occasions, neural network designs will draw some insights of our knowledge of the brain, but the focus will be to provide intuitions of how they work rather than to imitate the brain.

As an example, the *fully-connected* neural network is often used to draw a parallel between deep learning and neuroscience. The fully-connected neural network is designed as a composition of many functions $f_l(h_{l-1}; \theta_l)$ (as mentioned earlier), where each function is defined as a linear transformation followed by a nonlinear transformation as follows:

$$h_l = f_l(h_{l-1}; \theta_l) \tag{3.1.2}$$

$$= f_{act}\left(W_l h_{l-1} + b_l\right) , \tag{3.1.3}$$

where h_{l-1} is the output vector at previous layer $l-1$, h_l is the output vector at layer l, W_l is the weight matrix at layer l, b_l is the bias vector at layer l and f_{act} is the nonlinear transformation, also known as the *activation function*. In this case, the vectors h_{l-1} and h_l represent the artificial neurons, while the weight matrix W_l represents the artificial synapses.

An illustration of a layer in a fully-connected network is shown in Figure 3.1. The neurons h_{l-1} from the previous layer are processed from left to right into new neurons h_l. The process first apply a linear transformation parameterized by the weight matrix W_l and the bias vector b_l, then apply the activation function f_{act} element-wise on the result of the linear transformation. The resulting neurons h_l are then *sent* to the next layer to continue the computation. The fully-connected network of Figure 3.1 is often used to draw a parallel between deep learning and neuroscience, since it has a relatively similar structure to the biological neural circuit.

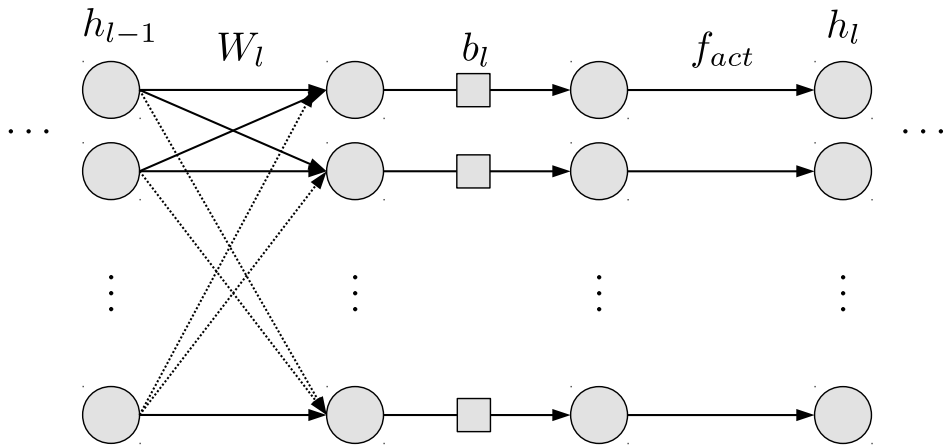


Figure 3.1: Illustration of a fully-connected network. The neurons h_{l-1} of the previous layer are processed into new neurons h_l from left to right. The process applies a linear transformation parameterized by a weight matrix W_l and by a bias vector b_l, followed by an element-wise nonlinear transformation f_{act}. This model is often used to draw a parallel between deep learning and neuroscience.

3.2 The Importance of the Activation Function

The activation function f_{act} plays an important role in deep learning because it is the main way by which we create neural networks that are nonlinear functions of the input. Without the activation function, a network with many layers would still be just a linear function. This is because a linear transformation of a linear transformation can be written as a single linear transformation. The role of the activation function is thus to break the chain of linear dependency between each pair of subsequent linear layers, such that the network becomes a sequence of nonlinear transformations.

The activation function must therefore be nonlinear, but it must also have other needed characteristics. This is because it influences other aspects of the network apart from breaking linear dependency. In general, we are concerned with its influence on *vanishing / exploding gradients* (Hochreiter, 1998), *dead units* (Standford, 2015) and *bias shift* (Clevert et al., 2016). In the following sections, we will provide more details about these three problems, and we will introduce an activation function that addresses each of them.

A summary of the activation functions that we will present in the following sections is shown in Table 3.1. We start our review by introducing the problem of vanishing and exploding gradients in Section 3.2.1 and present the Sigmoid, the Tanh and the ReLU activation functions in Section 3.2.2. We then present the problem of dead units in

Table 3.1: Summary of the presented activation functions.

Name		Function	Eq.	Parameters
Fixed Activation Functions				
Sigmoid	Sig	$f_{sigmoid}(x) = \dfrac{1}{1+\exp(-x)}$	(3.2.5)	-
Hyperbolic Tangent	Tanh	$f_{tanh}(x) = 2f_{sigmoid}(2x) - 1$	(3.2.13)	-
Rectified Linear Unit	ReLU	$f_{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$	(3.2.7)	-
Leaky ReLU	LReLU	$f_{LReLU}(x) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases}$	(3.2.10)	$a = 0.01$
Exponential Linear Unit	ELU	$f_{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ a(\exp(x) - 1) & x \leq 0 \end{cases}$	(3.2.14)	$a = 1$
Parametric Activation Functions				
Parametric ReLU	PReLU	$f_{LReLU}(x) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases}$	(3.3.1)	a
Adaptive Piecewise Linear Unit	APL	$f_{APL}(x) = \max\{0, x\} + \sum_{s=1}^{S} a_s \odot \max\{0, b_s - x\}$	(3.3.3)	$\{a_s, b_s\}_{s=1}^{S}$
S-Shaped ReLU	SReLU	$f_{SReLU}(x) = \begin{cases} t_l + a_l(x - t_l) & x \leq t_l \\ x & t_l < x < t_r \\ t_r + a_r(x - t_r) & x \geq t_r \end{cases}$	(3.3.8)	t_l, a_l, t_r, a_r
Maxout	-	$f_{Maxout}(x) = \max\{W_1 x + b_1, \ldots, W_K x + b_K\}$	(3.3.10)	$\{W_k, b_k\}_{k=1}^{K}$

Section 3.2.3 and present the LReLU in Section 3.2.4. We further present the problem of bias shift in Section 3.2.5 and present the ELU in Section 3.2.6. Finally, we present the parametric activation functions PReLU, APL, SReLU and Maxout, as well as an associated problem with piecewise linear function, in Section 3.3.

3.2.1 The Problem of Vanishing / Exploding Gradients

The presence of vanishing or exploding gradients is one of the main concerns when choosing an activation function. In order to better understand this problem and its relation with the activation function, we will investigate it on the following simple network:

$$\begin{cases} h_0 &= x \\ z_l &= w_l h_{l-1} + b_l \\ h_l &= f(z_l) \\ E &= \ell(h_L, y) \end{cases} \qquad (1 \leq l \leq L)\,, \tag{3.2.1}$$

where x is the input, y is the target output, f is the activation function, h_L is the network output, ℓ is the loss function and E is the value of the loss. We only used one neuron in each linear transformation $w_l h_{l-1} + b_l$ without loss of generality to simplify the equations.

We are interested in the quantity $\dfrac{\partial E}{\partial w_k}$ as a function of k, which is the derivative of loss E with respect to weight w_k at layer k. During training, the value $\dfrac{\partial E}{\partial w_k}$ will be used to update weight w_k by performing a gradient step in the opposite direction (in order to minimize E):

$$w_k \leftarrow w_k - \alpha \frac{\partial E}{\partial w_k}\,, \tag{3.2.2}$$

where $\alpha > 0$ is the learning rate. The analytic expression for computing the value $\dfrac{\partial E}{\partial w_k}$ will help us understand the difficulty of training a network with many layers.

Using the chain rule of derivation, we can compute the expression of $\dfrac{\partial E}{\partial w_k}$ as follows:

$$\frac{\partial E}{\partial w_k} = \left(\frac{\partial E}{\partial h_L}\right)\left(\prod_{l=k+1}^{L} \frac{\partial h_l}{\partial z_l}\frac{\partial z_l}{\partial h_{l-1}}\right)\left(\frac{\partial h_k}{\partial z_k}\frac{\partial z_k}{\partial w_k}\right)\,, \tag{3.2.3}$$

$$= \left(\frac{\partial E}{\partial h_L}\right)\left(\prod_{l=k+1}^{L} f'(z_l) w_l\right)\left(f'(z_k) h_{k-1}\right)\,. \tag{3.2.4}$$

As we can see in Eq. (3.2.4), this expression contains three parts. The first one is $\dfrac{\partial E}{\partial h_L}$, the derivative of the loss E with respect to the output of the network h_L. The second one is a product of $L - k$ terms, where each term contains the derivative with respect to z_l

of the activation function $f'(z_l)$, multiplied by weight w_l. The third one is the derivative of the activation function $f'(z_k)$ with respect to z_k multiplied by the activation neurons h_{k-1}.

As the number of layer L increases, the value of the derivative $\frac{\partial E}{\partial w_k}$ will be dominated by the product of $L-k$ terms in the second part of Eq. (3.2.4). When many terms $f'(z_l)w_l$ have magnitudes $|f'(z_l)w_l|$ lower than one, the product will tend to zero and the overall value $\frac{\partial E}{\partial w_k}$ will *vanish* to zero, as we go deeper. This is further accentuated as $k \to 0$, i.e. as we consider layers closer to the input. On the contrary, when many terms $f'(z_l)w_l$ have magnitudes $|f'(z_l)w_l|$ higher than one, the product will tend to a large magnitude, and the overall value $\frac{\partial E}{\partial w_k}$ will *explode*. The likelihood that any of these two events happens increases as $L-k$ increases. In other words, as the number of layers L increases and as we consider weights w_k at lower layers k. Thus, the weights w_k at lower layers are more likely to experiment vanishing or exploding gradients, as depth increases.

The presence of vanishing or exploding gradients has a large influence on the rate of convergence of the weights w_k. As seen in Eq. (3.2.2), the weights are updated based on the magnitude of the derivative $\frac{\partial E}{\partial w_k}$. When vanishing gradients happen, $|\frac{\partial E}{\partial w_k}| \approx 0$ and the weights w_k are barely modified, since $w_k \approx w_k - \alpha\frac{\partial E}{\partial w_k}$. On the contrary, when exploding gradients happen, $|\frac{\partial E}{\partial w_k}| \to \infty$ and the weight update of w_k becomes too large. This causes problems in both cases. For vanishing gradients, the network will be forced to converge to a local minimum located near the initial values of the weights. This local minimum will most likely not generalize well. For exploding gradients, the weights will diverge away from any local minimum and training will not be possible. In both cases, training will fail to learn a network that generalize well.

The sigmoid activation function (also known as logistic) is an example of a nonlinear function that can increase the likelihood of vanishing gradients. Even though it breaks linearity, it is not always suitable as activation function. The function is defined as follows:

$$f_{sigmoid}(x) = \frac{1}{1+\exp(-x)}\,, \tag{3.2.5}$$

and its derivative with respect to its input is:

$$f'_{sigmoid}(x) = f_{sigmoid}(x)\cdot(1-f_{sigmoid}(x))\,. \tag{3.2.6}$$

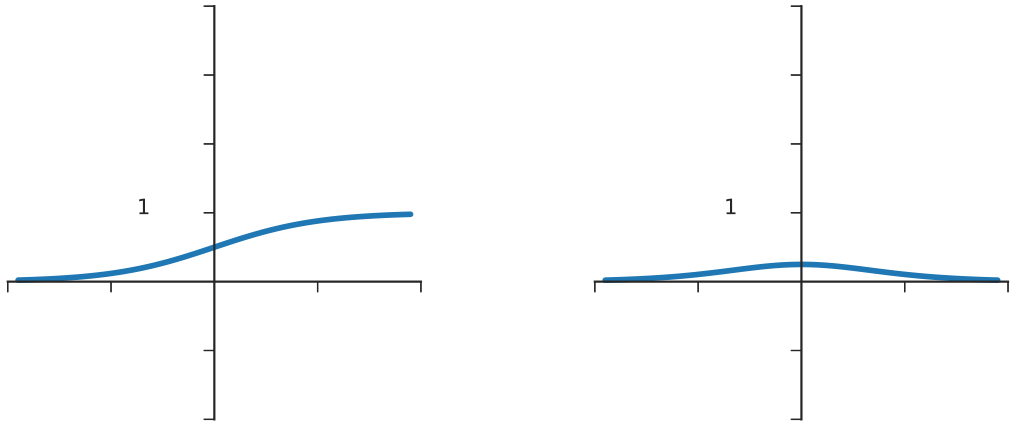


Figure 3.2: The sigmoid activation function (left) and its derivative (right). This activation function increases the likelihood of vanishing gradients because its derivative is close to zero of most of its domain.

As we can see from Figure 3.2, it has a derivative close to zero on most of its domain. This makes the product term in Eq. (3.2.4) more likely to vanish, since the interval of input values with a large derivative is small. For this reason, the sigmoid activation function is rarely used within the hidden layers of a deep neural network (it is usually kept for gating mechanisms (Hu et al., 2018) or to provide probability values as output).

3.2.2 Addressing Vanishing Gradients with the Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) (Nair and Hinton, 2010) has become the *de facto* activation function in deep learning. It is defined as identity for positive arguments and zero elsewhere:

$$f_{ReLU}(x) = \max\{x, 0\}\,. \tag{3.2.7}$$

The shape of the function, as well as its derivative, is shown in Figure 3.3. We say that the function is *non-saturated* because it has a derivative of one for all positive arguments.

The ReLU was first proposed in the context of unsupervised pre-training for Restricted Boltzmann Machine (RBM) (Nair and Hinton, 2010; Jarrett et al., 2009), and then successfully used for training deep neural networks (Glorot et al., 2011; Krizhevsky et al., 2012). For the first time, we could train deep networks solely based on labelled data without performing unsupervised pre-training, which is a tedious network ini-

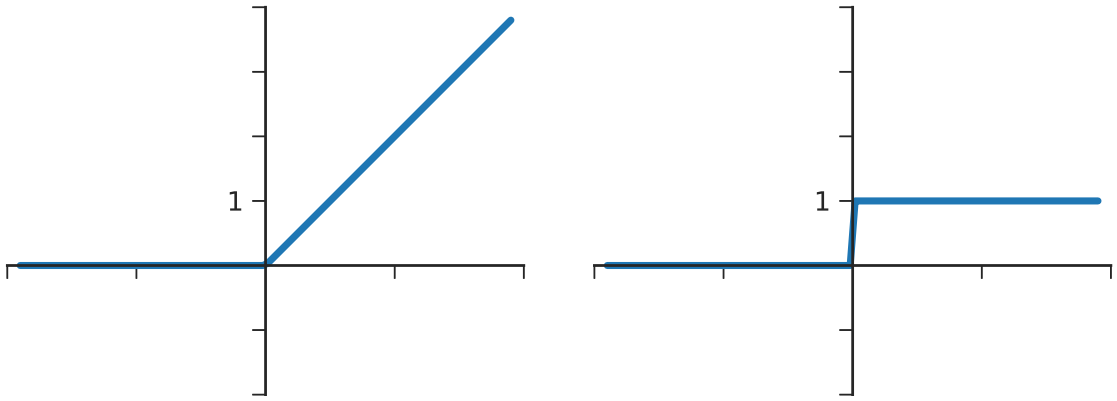


Figure 3.3: The Rectified Linear Unit (left) and its derivative (right).

tialization that substantially increases training time (Glorot et al., 2011). This was due to its characteristic of being *non-saturated* that lowers the likelihood of vanishing gradients (Nair and Hinton, 2010). It was then later used in the seminal *AlexNet* architecture (Krizhevsky et al., 2012), which won the popular ImageNet 2012 competition. As a result, the ReLU replaced the sigmoid function that was the most common activation function at the time.

3.2.3 The Problem of Dead Units

Beyond lowering the risk of vanishing gradients, the ReLU has the advantage of inducing sparsity within the vector (or feature map) of activated neurons. We can see how ReLU induces this sparsity by analyzing the statistics of the neuron activation after weight initialization. Using the standard *Kaiming normal* initialization (He et al., 2015), the weights of the network are initialized using a normal distribution with a mean of zero and a variance of $\sigma^2 = \frac{2}{q}$, where q is the number of input neurons:

$$W_{pq} \sim \mathcal{N}\left(0, \frac{2}{q}\right) . \tag{3.2.8}$$

As for the biases b_p, they are initialized to zero. Given that $h = Wz + b$, and using the fact that the weighted sum $s = \sum_i a_i v_i$ of independent normally distributed random variables $v_i \sim \mathcal{N}\left(\mu_i, \sigma_i^2\right)$ is normally distributed with a mean $\sum_i a_i \mu_i$ and variance $\sum_i a_i^2 \sigma_i^2$, we have at initialization that a single neuron preactivation h_p is distributed

according to the following distribution:

$$h_p = \sum_{i=1}^{q} W_{pi} z_i \sim \mathcal{N}\left(0, \sum_{i=1}^{q} \frac{2z_i^2}{q}\right), \ \forall p\,. \tag{3.2.9}$$

In other words, the pre-activation neurons h are normally distributed around a mean of zero and a variance of $\sum_{i=1}^{q} \frac{2z_i^2}{q}$. Due to the symmetry of the normal distribution around its mean, about half the entries of h are negative and about half the entries of h are positive. Thus, given that the vector h is forwarded to the ReLU activation f_{ReLU}, approximately half the entries of $f_{ReLU}(h)$ will be zero, and approximately half the entries of $f_{ReLU}(h)$ will be positive. As a result, the output vector $f_{ReLU}(h)$ is 50% sparse on average. As we saw in Section 1.2.4, sparsity is related to Occam's razor learning principle that is one of the prior beliefs to find better representations.

Sparsity can be beneficial, but can make training more difficult. The fact that ReLU outputs zero for negative values can cause some units to never be positively activated. As a result, the gradient of the loss function with respect to the weights of these units will always be zero. So long as the units are negative, the weights will not be updated during training. We refer to them as *dead units* (Standford, 2015).

The death of a unit can happen randomly during training. It usually occurs when gradient descent reaches a poor local minimum. Such a poor local minimum is hard to escape because the weights of the corresponding unit never change. In the following section, we will detail the Leaky ReLU (LReLU) that addresses this problem.

3.2.4 Addressing Dead Units with the Leaky ReLU (LReLU)

The Leaky ReLU (LReLU) (Maas et al., 2013) activation function was proposed as an attempt to overcome the issue of dead units when employing the ReLU. The idea of the LReLU is to modify the output value for negative arguments by using a linear function with a small positive slope a. The function can be defined as follows:

$$f_{LReLU}(x) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases}, \tag{3.2.10}$$

which can also be written as:

$$f_{LReLU}(x) = \max\{0, x\} + \min\{0, ax\}\,. \tag{3.2.11}$$

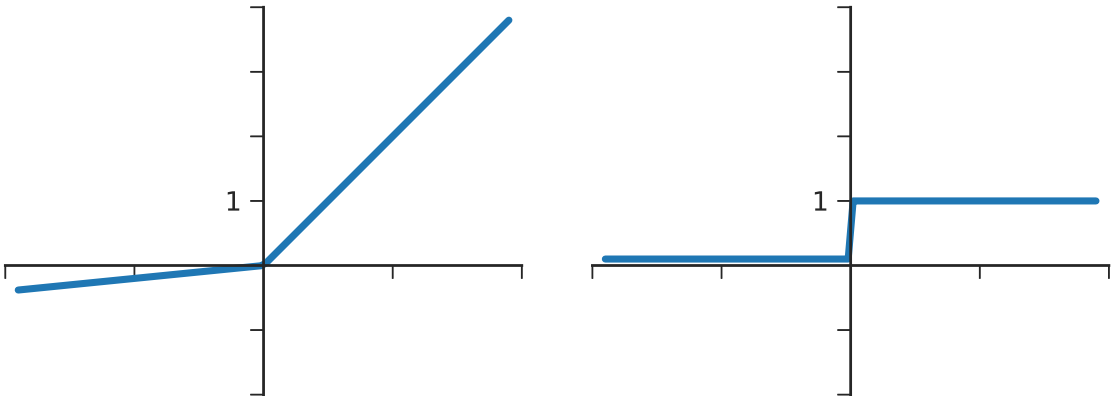


Figure 3.4: The Leaky ReLU (left) and its derivative (right).

The purpose of the slope a is to have a positive derivative value for negative arguments:

$$f'_{LReLU}(x) = \begin{cases} 1 & x > 0 \\ a & x \leq 0 \end{cases}. \tag{3.2.12}$$

The shape of the function, along with its derivative, is shown in Figure 3.4. As we can see, a unit x that is always negative when forwarded to LReLU will be activated to a small negative value ax, rather than zero. The gradient will be always positive, which means that the unit can never die. The corresponding weights will always have a non-null gradient, no matter how negative the unit is.

The chosen value of the slope hyper-parameter a has an impact on training. A large value will effectively lower the number of dead units, but will decrease the nonlinearity of the function. On the other hand, a small value will make the activation more nonlinear, but will bring back the problem of dead units. In general, the slope hyper-parameter a must be chosen such that there is a good compromise between nonlinearity and dead units. Maas et al. (2013) suggested a default value of $a = 0.01$, but opted for a hyper-parameter grid search to determine the proper value for their experiments.

3.2.5 The Problem of Bias Shift

Bias shift is a problem often seen when trying to alleviate vanishing gradients by using a non-saturated activation function, like the ReLU. This problem is defined as the change of the mean neuron activation as depth increases and as training progresses (i.e. as weights are updated). To better illustrate this problem, we performed the following experiment. We computed the mean neuron activation at each layer of two randomly

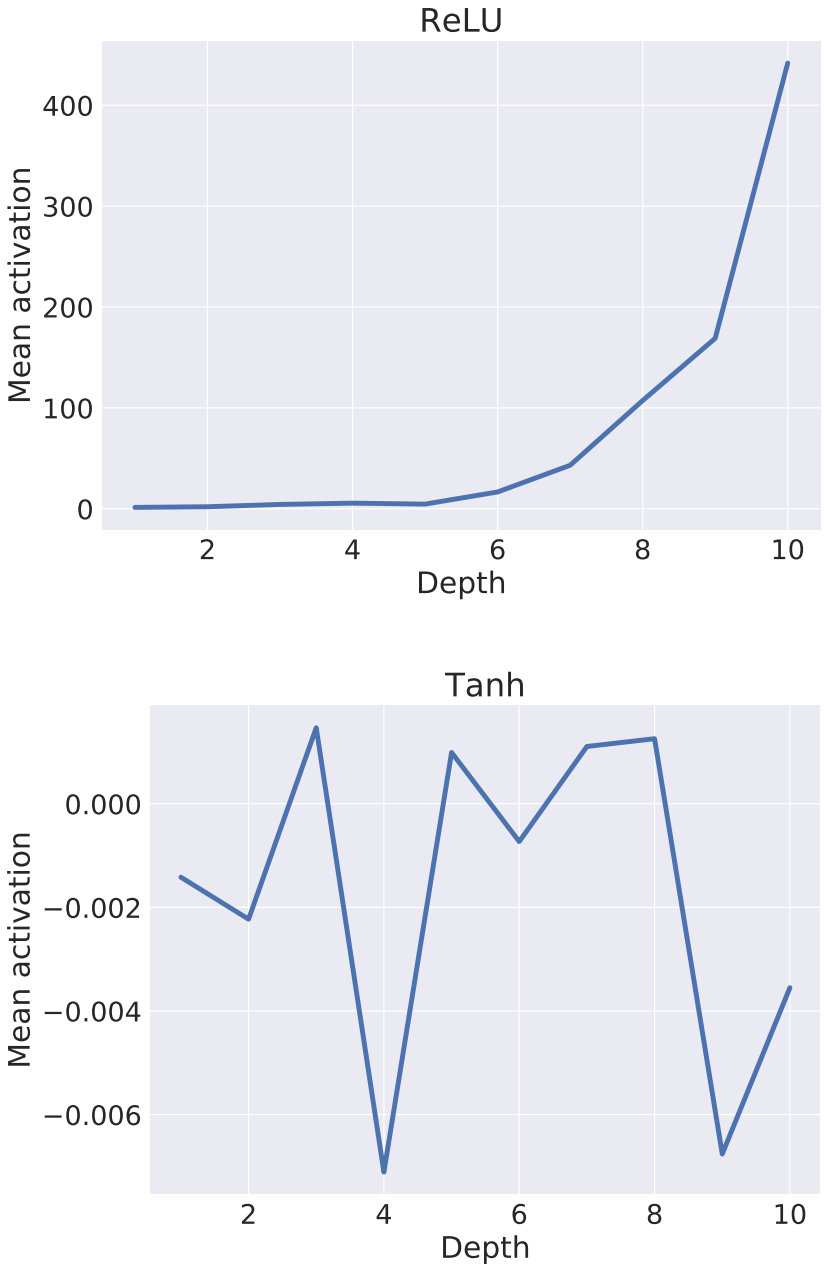


Figure 3.5: Illustration of bias shift on randomly initialized networks. The figures present the mean activation at each layer, where each layer has ten dimensions, and the input and the weights are normally distributed with a mean of zero and variance of 1 (the networks have no biases). The network at the top has ReLU activations, while the network at the bottom has Tanh activations. For the ReLU network, the mean activation across the layers increases because ReLU has a mean activation larger than one. For the Tanh network, the mean activation across the layers stays around zero, because Tanh has a mean activation of zero. The ReLU network is more likely to encounter bias shift than the Tanh network.

initialized networks. Both networks have the same identical architecture, but each uses a different activation function: one uses ReLU, the other uses Tanh. The input and the weights are normally distributed with a mean of zero and a variance of one. The network has no trainable biases b_l to better illustrate the problem of bias shift.

The results of our experiment are shown in Figure 3.5. At the top of Figure 3.5, we show the mean activation for a ReLU network, while at the bottom, we show the mean activation for a Tanh network, as a function of depth. Recall that the Tanh activation function is a rescaled version of the Sigmoid activation that outputs values in the interval $(-1, 1)$:

$$f_{tanh}(x) = 2f_{sigmoid}(2x) - 1\,. \tag{3.2.13}$$

The important difference between ReLU and Tanh is that Tanh has a mean activation of zero while ReLU has a positive mean activation. This is because Tanh is antisymmetric around zero whereas ReLU is not. As a result, ReLU will tend to increase the mean neuron activation as more transformations are applied, while Tanh will tend to keep it around zero. This is what we observe in Figure 3.5. As depth increases, the mean activation of the ReLU network increases, while the mean activation of the Tanh network stays around zero. This is what we refer to as bias shift.

We can see the problem of bias shift as forcing each transformation to apply a positive bias to its output. It is as if we designed the linear transformations followed by ReLU to apply a positive bias *a priori*, even though no trainable bias b_l is used (as we showed in Figure 3.5). As a result, the ReLU network will tend to have a larger mean activation at the higher layers, and a smaller mean activation at the lower layers.

Bias shift can have a significant impact on training. A weight update in the lower layers that generates a small bias increase will increase the mean activation in the higher layers. Likewise, a bias decrease will decrease the mean activation in the higher levels also by a great amount. This is due to the intensification effect on the mean activation as it propagates through the network. Thus, the mean activation at the higher levels will tend to oscillate more than the mean activation at the lower layers. These oscillations can contribute in slowing down convergence, and thus make training deep networks with many layers difficult.

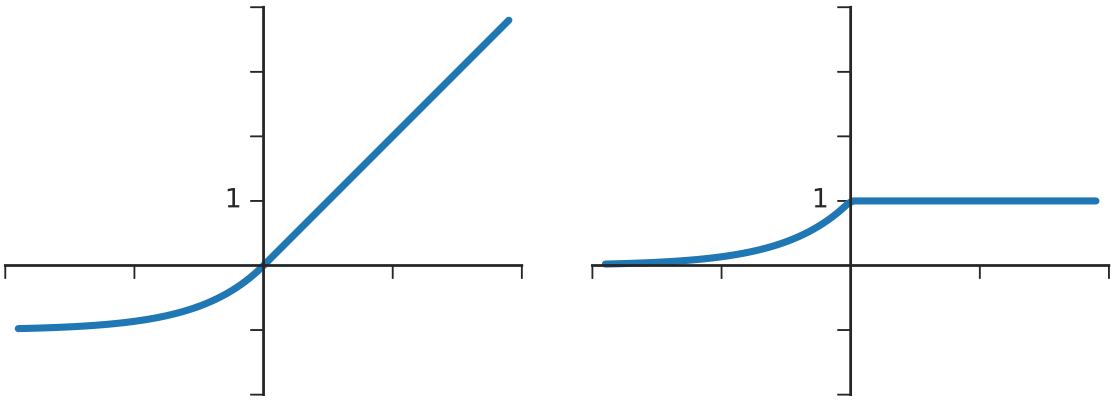


Figure 3.6: The Exponential Linear Unit (left) and its derivative (right).

3.2.6 Addressing Bias Shift with the Exponential Linear Unit (ELU)

The Exponential Linear Unit (ELU) (Clevert et al., 2016) was designed precisely to address this problem of *bias shift*. The underlying idea is to output negative values, instead of zero, for negative arguments in order to shift the mean activation closer to zero. To do so, the ELU uses an exponential decay for negative arguments that saturates at a value of -1. The function is defined as follows:

$$f_{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(\exp(x) - 1) & \text{otherwise} \end{cases}, \tag{3.2.14}$$

where the authors proposed $a = 1$ as default. In other words, ELU is defined as identity for positive arguments and $\exp(x)-1$ for negative arguments. The shape of the function, along with its derivative, are shown in Figure 3.6. Note that the exponential decay on the negative side ensures that the function is differentiable everywhere.

The use of identity for positive arguments makes ELU a non-saturated activation function. The derivative for the positive arguments is one everywhere, which lowers the likelihood of vanishing gradients just as ReLU. In addition, ELU helps to manage bias shift due to its negative saturation. The presence of negative values lowers the mean neuron activation and thereby, reduces oscillations. Note that bias shift can still happen in deep networks with ELU. The mean neuron activation can still increase as it passes through the network. This is because the largest negative value is bounded to -1 while the largest positive values is unbounded. Nonetheless, ELU is still well-suited to reduce bias shift (Clevert et al., 2016).

3.3 Parametric Activation Functions

We have seen previously when we introduced the Leaky ReLU (LReLU) that it employs a slope hyper-parameter a that must be selected by hand. The authors Maas et al. (2013) suggested a default value of 0.01, but relied on grid search in their experiments to find the best value. A grid search can be a good strategy to determine the value of hyper-parameters, but it will struggle on large datasets and big networks. Each additional hyper-parameter with a set of T candidate values multiplies the number of training runs by T. In the case of LReLU, the number of total hyper-parameters to search for is the number of slopes a, which corresponds to the number of LReLU layers (assuming each LReLU layer defines a single slope a, as it is done in practice). This means that a network with L LReLU layers would need to perform T^L training runs for grid search. Considering that a single training run can take up to several hours, grid search quickly becomes unsuitable as the number of layers increases.

The Parametric ReLU (PReLU) (He et al., 2015) is a simple yet elegant solution to this problem of choosing the slope a in LReLU. The idea is to consider a as part of the overall parameter space of the network and learn it during training. Gradient descent can then use $\frac{\partial E}{\partial a}$, the derivative of the loss with respect to a, in order to update a iteratively, just as it is done for the weights and biases. The gradient of f_{PReLU} with respect to a is given by:

$$\frac{\partial f_{PReLU}(x)}{\partial a} = \begin{cases} 0 & \text{if } x > 0 \\ x & \text{if } x \leq 0 \end{cases} \tag{3.3.1}$$

which is used to compute $\frac{\partial E}{\partial a}$ and update parameter a:

$$a \leftarrow a - \alpha \frac{\partial E}{\partial a} \,. \tag{3.3.2}$$

The PReLU is an activation function in the family of *parametric activation functions*. A parametric activation function can be seen as an *evolutive* function that changes its shape during training. It defines parameters that control different aspects of its shape and uses the gradient of the loss function to guide the updates of its shape during training. In the case of PReLU, it defines a single parameter a that controls the slope of the linear function for negative inputs.

Parameterizing an activation function opens up the opportunity to find a better activation. More precisely, it allows the data to guide training towards a shape more

suitable than a predefined one, which would otherwise be difficult to chose by hand. In the case of PReLU for instance, learning the slope a removes the need to perform hyper-parameter grid search and allows training to find a good compromise between dead units and nonlinearity (He et al., 2015). In general, parameterizing an activation function adds a new layer of flexibility that has been shown to be beneficial (He et al., 2015).

In the following sections, we review other works on parametric activation functions. We review the Adaptive Piecewise Linear Unit (APL), the S-Shaped ReLU (SReLU) and Maxout. Each function takes a step further in the direction of parameterizing the ReLU and tries to learn a piecewise linear function. We also explain an issue with piecewise linear functions and gradient descent that can impede training.

3.3.1 Adaptive Piecewise Linear Unit

The Adaptive Piecewise Linear (APL) (Agostinelli et al., 2014) unit is defined as ReLU plus a sum of S parametric Hinge-shaped functions, as follows:

$$f_{APL}(x) = \max\{0, x\} + \sum_{s=1}^{S} a_s \odot \max\{0, b_s - x\}\,, \tag{3.3.3}$$

where $\odot$ denotes the element-wise multiplication between two vectors and the max operator is also performed element-wise. The number of parametric Hinge functions S is a hyperparameter fixed in advance, while parameters a_s and b_s are learning during training. Parameters a_s control the slopes of the linear segments, while parameters b_s control the locations of the intersection points of the linear segments.

Examples of APL shapes are shown in Figure 3.7. The left column shows the shape of the functions, while the right column shows their corresponding derivatives. As we can see, APL can learn piecewise linear functions that can be both convex and non-convex (second row of the figure). Moreover, APL has the particularity that, as $x \to \infty$, it tends to the identity function $f_{APL}(x) = x$, and as $x \to -\infty$, it tends to a linear function $f_{APL}(x) = \alpha x + \beta$, for some α and β.

APL is a generalization of PReLU and we can see how by rewriting PReLU as a sum of Hinged-shaped functions in the form presented in Eq. (3.3.3). Using the fact that the slope in PReLU is constrained with $a \in (0, 1)$, we can redefine PReLU as follows:

$$f_{PReLU}(x) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases} \tag{3.3.4}$$

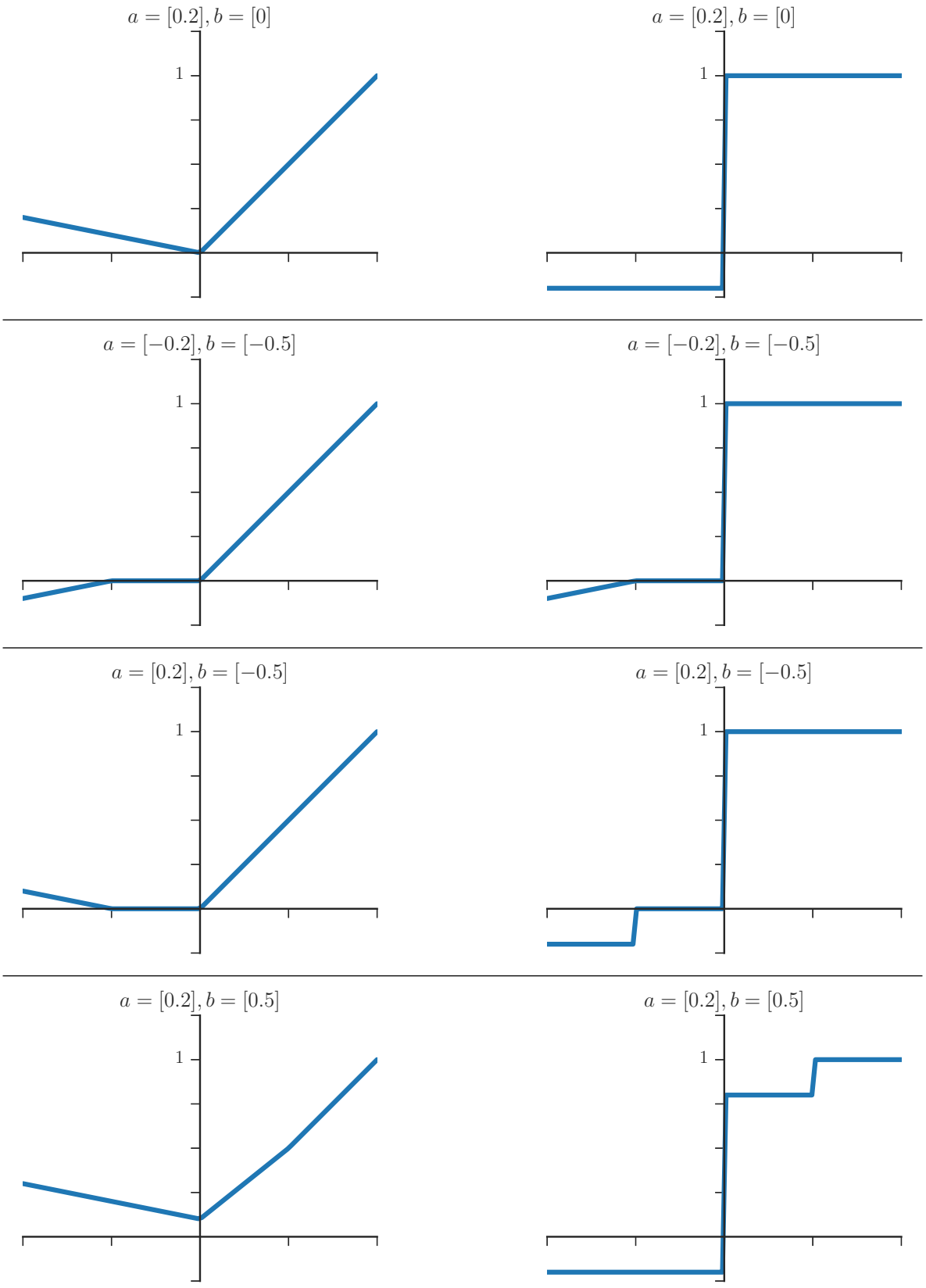


Figure 3.7: Examples of Adaptive Piecewise Linear Unit (left) and their corresponding derivatives (right).

$$= \max\{0, x\} + \min\{0, ax\}\,, \tag{3.3.5}$$

$$= \max\{0, x\} + a\min\{0, x\}\,, \tag{3.3.6}$$

$$= \max\{0, x\} - a\max\{0, -x\}\,. \tag{3.3.7}$$

In other words, PReLU is a specific case of APL where $S = 1$, $b_1 = 0$, $a_1 \in (-1, 0)$ and both parameters b_1 and a_1 are scalars instead of vectors. As a result, APL can also alleviate dead units by having a non-null slope a_s for the leftmost linear segment.

3.3.2 S-Shaped ReLU

The S-Shaped ReLU (SReLU) (Jin et al., 2016) is a parametric activation function that tries to imitate the Webner-Fechner law and the Stevens law. It is defined as follows:

$$f_{SReLU}(x) = \begin{cases} t_l + a_l(x - t_l) & x \leq t_l \\ x & t_l < x < t_r \\ t_r + a_r(x - t_r) & x \geq t_r \end{cases}, \tag{3.3.8}$$

where a_l is the slope of the leftmost segment, a_r is the slope of the rightmost segment, t_l is the point of intersection between the leftmost segment and the center segment, and t_r is the point of intersection between the center segment and the rightmost segment. Examples of SReLU shapes are shown in Figure 3.3.8. As we can see, the function is defined as identity in the range $[t_l, t_r]$, as linear with a slope of a_l and an intercept of $t_l(1 - a_l)$ in the range $(-\infty, t_l]$, and as linear with a slope of a_r and an intercept of $t_r(1 - a_r)$ in the range $[t_r, \infty)$.

We can see how SReLU is also a generalization of PReLU. Indeed, by using $t_l = 0$, $a_l = 0$, $t_r = \infty$ and any values for a_r and t_r, we have:

$$f_{SReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & 0 < x < \infty \\ t_r + a_r(x - t_r) & x \geq \infty \end{cases}, \tag{3.3.9}$$

which is the ReLU. Thus, SReLU is a generalization of PReLU for two reasons. The first one is the use of a second intersection point on the positive part. The second one is the fact that SReLU can learn the position of both intersection points. Note that unlike APL, SReLU is forced to have $f_{SReLU}(0) = 0$ for any parameterization. See Figure 3.7 for an example of APL where $f_{APL}(0) \neq 0$.

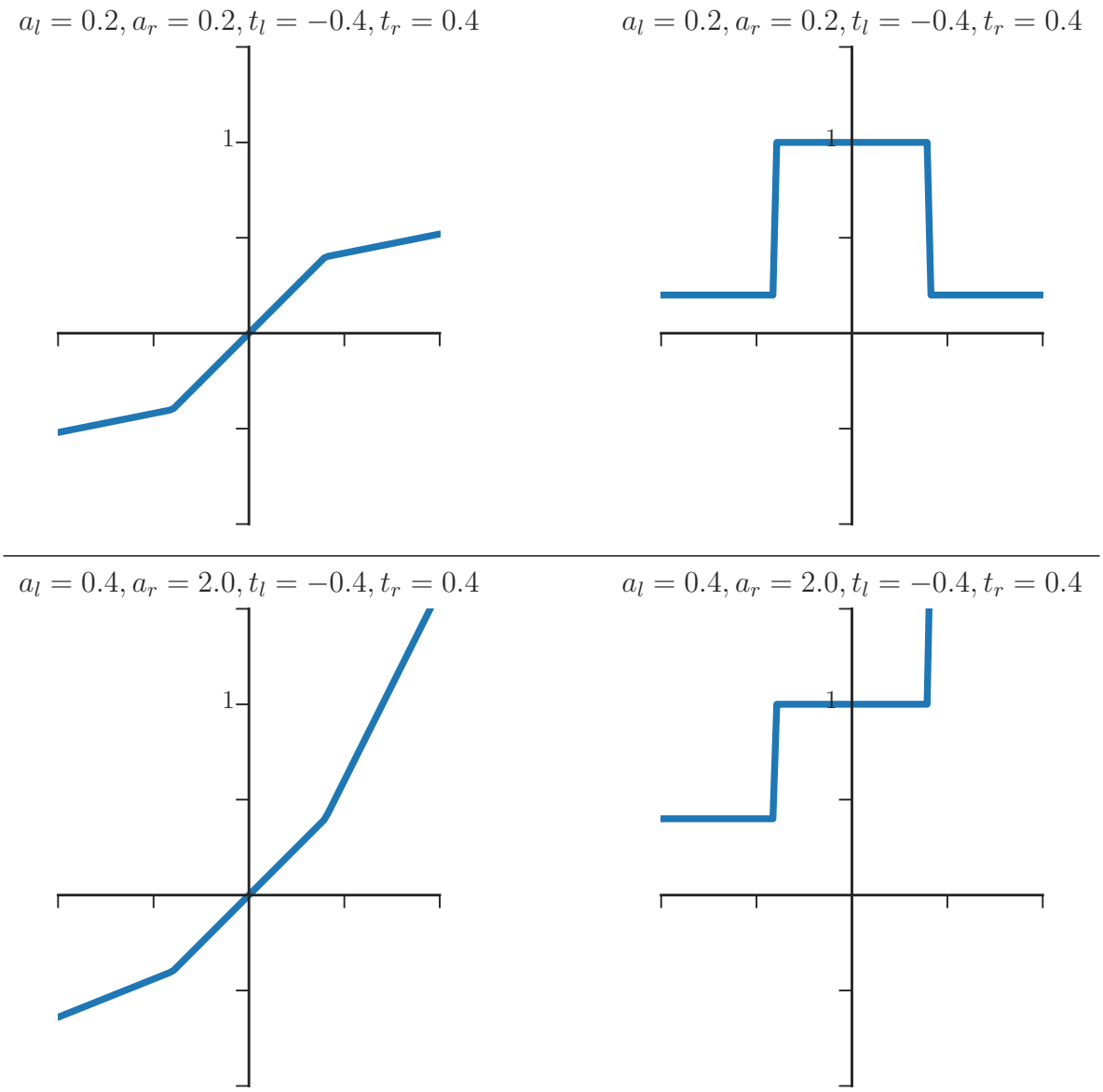


Figure 3.8: Examples of S-Shaped ReLU (left) and their corresponding derivatives (right).

3.3.3 Maxout Unit

The Maxout Unit (Maxout) (Goodfellow et al., 2013) is another activation function in the family of parametric activation functions. Unlike the other activations, Maxout computes the maximum over K linear transformations rather than an element-wise nonlinear function over the entries of the input tensor. In other words, it does not follow the standard form $f(z) = f(Wh + b)$, but is instead defined as follows:

$$f_{Maxout}(x) = \max\{z_1, \ldots, z_K\} \tag{3.3.10}$$

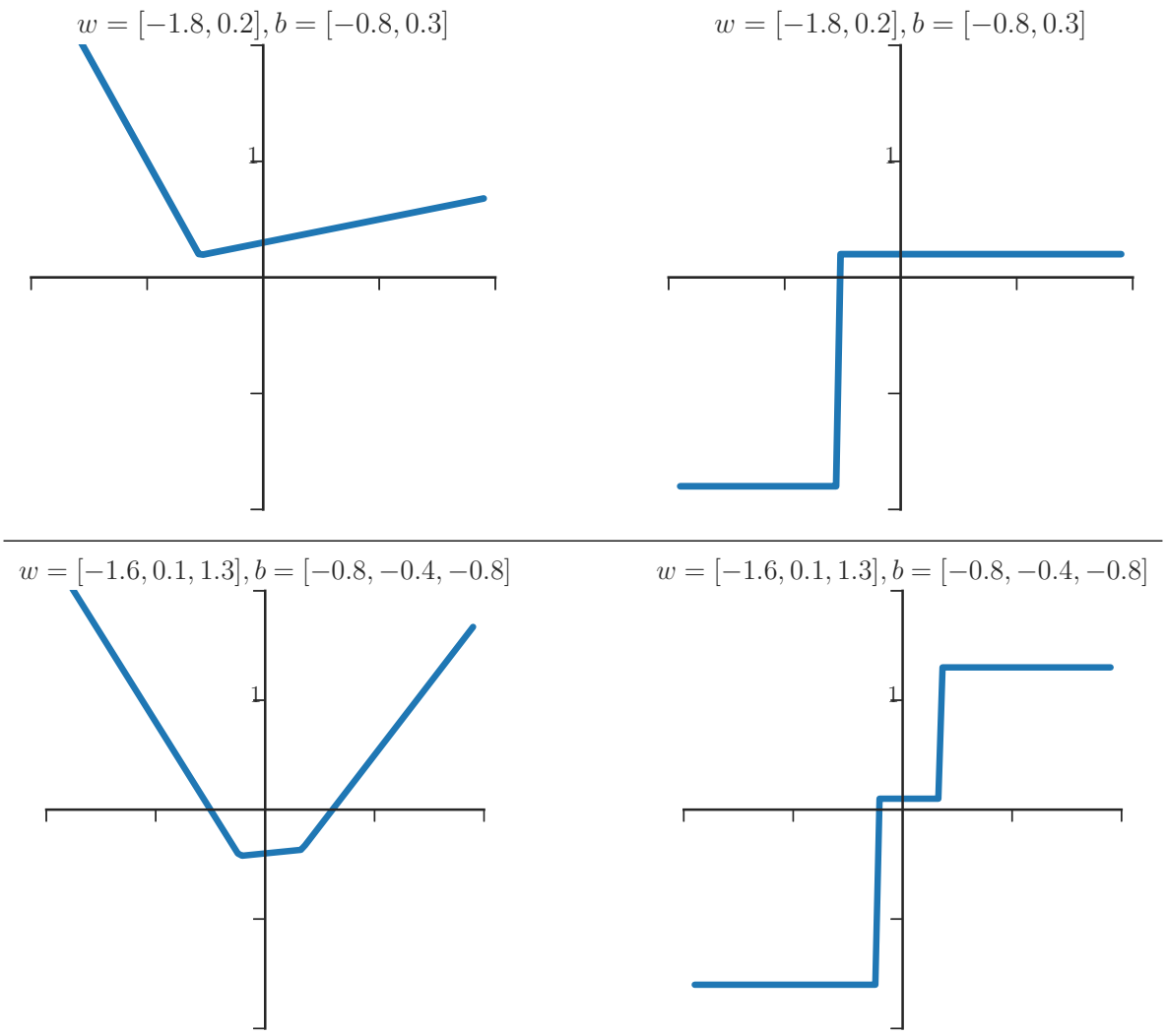


Figure 3.9: Examples of Maxout activation functions (left) and their corresponding derivatives (right).

$$= \max\left\{W_1 x + b_1, \ldots, W_K x + b_K\right\} , \tag{3.3.11}$$

where W_k and b_k are the weights and biases of the K linear transformations, and the max operator is performed element-wise over all K outputs $z_k = W_k x + b_k$. Examples of Maxout shapes are shown in Figure 3.9.

We can see how Maxout is a generalization of PReLU as follows. We can obtain PReLU by using $K = 2$ linear transformations and by setting $W_1 = 1$, $b_1 = 1$, $W_2 = a$ and $b_2 = 0$, where a is a positive scalar between 0 and 1. In that case, we obtain $f_{Maxout}(x) = \max\{W_1 x + b_1, W_2 x + b_2\} = \max\{x, ax\}$, which is PReLU.

Maxout has, however, two limitations that the previous parametric activation functions

do not have. The first one is that Maxout can only learn convex shapes. This is due to the max operator applied on linear transformations that creates a convex piece-wise linear function (Ovchinnikov, 2002). The second is that Maxout multiplies the number of parameters to be learned by K. This is due to its use of K linear transformations rather than a single linear transformation. For these reasons, the other existing parametric activation functions are usually favored over Maxout.

3.3.4 A Problem with Piecewise Linear Function

A piece-wise linear function, such as PReLU, APL, SReLU and Maxout, can add a lot of flexibility when its number S of linear segment is large, but can also cause problems during training. This is because the number of points at which the function is non-differentiable increases linearly with S. In that case, computing the gradient with back-propagation requires the use of sub-derivatives. A sub-derivative provides a predetermined value for the derivative at a non-differentiable point that is selected based on the slopes of the two segments that intersect at the point. For instance, in the case of ReLU, the function is non-differentiable at $x = 0$. The segment on the left of $x = 0$ has a derivative of zero and the segment on the right of $x = 0$ has a derivative of one. The sub-derivative for ReLU at $x = 0$ is therefore a value in the interval $[0, 1]$. It is standard to use 0, but it could be any value inside $[0, 1]$, since there are no one definite choice.

For this reason, the selected value for the derivative can be relatively different from the derivative of the two segments. Considering ReLU again, the derivative jumps from one to zero as the input changes from positive to zero to negative. This jump can be relatively large in comparison to the change of the input, since a infinitesimal change of ϵ can be sufficient to cause that jump. As a result, a variation in the input around the non-differentiable points can increase the variance of the weight update during gradient descent and cause oscillations. An activation function with few non-differentiable points should usually be favored rather than an activation function with many of them (LeCun et al., 2015). In our contribution in Chapter 6, we will see how to parameterize the ELU by ensuring that the shape remains differentiable at all points during training, thus addressing this problem.

3.4 A Review of Important Convolutional Neural Network Architectures

In this section, we review important neural network architectures. We focus our review on the family of convolutional architectures, since convolutional neural networks (CNNs) will be the main model of our contributions in Chapter 6 and Chapter 7. We review AlexNet, NiN, Overfeat, VGG, GoogLeNet, All-CNN, ResNet and DenseNet by comparing their architecture, characteristics and performance on the well-known ImageNet task. We end this section with a summary of their performance, size and number of operations.

3.4.1 AlexNet

AlexNet (Krizhevsky et al., 2012) has been an influential network over the past decade. It was proposed in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton for the annual ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), which is a coveted competition in computer vision. ImageNet started as an object recognition task and quickly evolved into a mix of object recognition, detection and segmentation tasks over the years. For the 2012 competition, the main challenge was object recognition, with a training set containing 1.2 million large-scale images, a validation set and a test set containing 150,000 ones. It was considered a challenging task at the time for both its unprecedented large scale and difficulty (there are one thousand classes). Many still consider it challenging even today with our current high-end GPUs.

We can get an idea of the difficulty of the task by looking at the results from the first two years. The lowest top-5 error rate of 2011 was 25.8% [1], while the lowest top-5 error rate of 2010 was 28.2% [2]. The top-5 error rate is the rate at which a model does not output the correct label in its top 5 predictions. In other words, the model can guess the correct answer five times before we rule the classification as a failure. At that time, it was considered standard to have about 30% top-5 error rate when experimenting novel approaches on the dataset. We also expected the top-5 error rate to improve at a rate of about 2% per year, since we only saw an improvement of 2.4% between the year 2010 and the year 2011. ImageNet competition was pushing the limits of the models of the time.

[1] http://image-net.org/challenges/LSVRC/2011/results
[2] http://image-net.org/challenges/LSVRC/2010/results

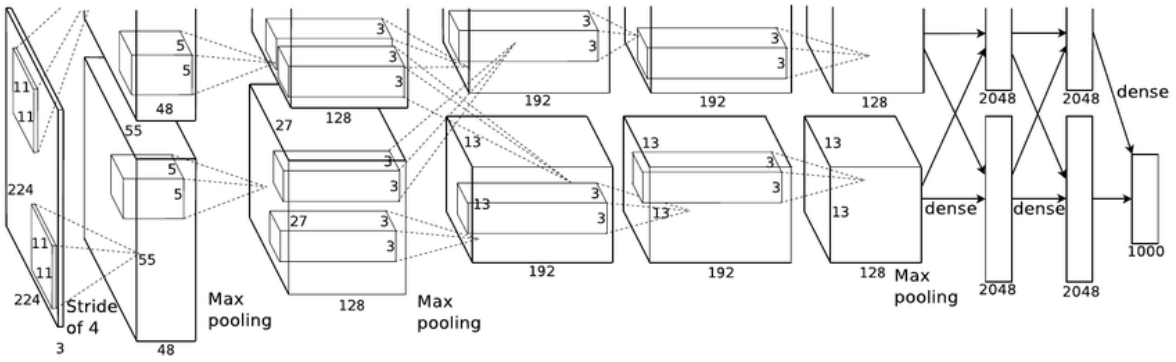


Figure 3.10: AlexNet architecture. Image taken from Krizhevsky et al. (2012).

The introduction of AlexNet in 2012 changed the game entirely. AlexNet obtained the lowest top-5 error rate with 16.4%, while the next best entry obtained 26.2% [3]. AlexNet improved the top-5 error rate by 9.8% over the next best entry, and by 9.4% over the best entry of the previous year. The next best entry did not even improve the accuracy of the best entry of the previous year, which both used shallow approaches. What was surprising is that AlexNet was a convolutional neural network, a model already introduced two decades earlier (LeCun et al., 1998), but with several modifications that made it more efficient. This set the tone for the competitions of the following years, which were quickly overwhelmed by neural network approaches. To this day, many considered the events of the 2012 ImageNet competition as the most contributing factor for the resurgence of neural networks.

AlexNet has a relatively simple structure in comparison to nowadays architectures. Its architecture is shown in Figure 3.10. At the time however, it was quite different from most previous neural network architectures. First, AlexNet uses the ReLU activation function instead of the Sigmoid activation function. ReLU was not as popular as the other saturated activation functions back in the days, but it was growing in popularity due to the works of Jarrett et al. (2009), Nair and Hinton (2010) and Glorot et al. (2011). After AlexNet was proposed, ReLU quickly became the mainstream activation function.

Second, the fully-connected part at the topmost layers of the architecture employs Dropout (Srivastava et al., 2014), which is a regularization tool based on random perturbations. The idea of Dropout is to prevent co-adaptation by randomly dropping hidden units from the network during training. Co-adaptation is a problem where multiple hidden units activate similarly on most of the inputs, making them redundant.

[3] http://image-net.org/challenges/LSVRC/2012/results.html

Dropout forces each unit to operate as much as an independent feature detector as possible without relying on the other units in the same layer. As a result, network trained in such a way can be seen as a collection of many smaller networks that form an ensemble of exponentially large networks. At the time, Dropout significantly reduced overfitting and gave major improvements over other regularization methods (Hinton et al., 2012; Deng et al., 2013; Simonyan and Zisserman, 2014; Kiros et al., 2014).

Third, the authors used heavy data augmentation techniques during training. The goal of data augmentation is to artificially increase the number of training data, in order to further reduce overfitting. It can be done by applying class-preserving stochastic transformations on the available images from the dataset to randomly generate new images that look similar, but that are different. They employed random translations, horizontal reflections and patch extraction, as well as a novel transformation that randomly alters the pixel intensities, which they proposed for the competition.

And finally, they trained their network on two GTX 580 GPUs, which gave them a significant speed advantage. A single GTX 580 has only 3GB of memory, so they had to split their architecture on two GPUs to be able to train. The parallelization scheme that they employed put half the neurons on one GPU, and the other half on the other GPU. This allowed them to train a network about twice bigger. They also indicated that training their bigger network on two GPUs simultaneously took less time than training their smaller network on one GPU.

In summary, the work of Krizhevsky et al. (2012) showed that convolutional neural networks could be trained efficiently and obtain better performances than the previous standard vision models. This simultaneously shattered two preconceived ideas that first, neural networks were too bulky to be trained efficiently and second, that they were too complex to attain statistical generalization. Nowadays, these ideas are accepted by most machine learning practitioners, even though AlexNet is no longer the mainstream architecture.

3.4.2 Network in Network (NiN)

The Network in Network (NiN) (Lin et al., 2013) architecture soon followed AlexNet. This paper was influential for its two contributions: the MLPConv layer and Global Average Pooling. Both contributions are detailed below.

NiN Contribution #1: MLPConv Layer

NiN was influential because it had a new take on the design on convolutional layers. They proposed a novel convolutional operation, called the MLPconv (MLP stands for Multi-Layer Perceptron), which can be seen as a nonlinear extension of the standard *linear* convolution. The idea was to replace the dot product between each kernel and the units of the receptive field by a fully-connected neural network. For instance, a standard convolutional layer with $K \times 3 \times 3$-dimensional kernels (where K is the depth of the feature map) would be replaced by a MLPconv layer with a fully-connected network that has $K \cdot 3 \cdot 3$ input neurons. The fully-connected network was itself convolved on the feature map and had a depth of two or three layers.

NiN Contribution #2: Global Average Pooling

The second reason why NiN was influential is that the authors proposed Global Average Pooling (GAP) near the top of the network. GAP was designed to be used as a bridge between the convolutional part and the fully-connected part of the network. Traditional convolutional networks like AlexNet, and even previous ones like LeNet (LeCun et al., 1998), relied on matrix flattening followed by several fully-connected layers to convert the three-dimensional feature maps as a two-dimensional probability vector for classification or as a single value for regression. The main issue with this approach is that it adds a considerable number of parameters to be learned. For instance, the number of parameters of the fully-connected part in AlexNet represented more than 95% of the total number (2.5M for the convolutional part and 58.6M for the fully-connected part). This is in fact the reason why AlexNet relied on Dropout layers in its fully-connected part, in order to regularize this large portion of the network.

The idea of GAP is to reduce the dimensionality of the feature maps, before sending them to the fully-connected layers, by computing the global spatial average. In other words, GAP replaces the flattening operation by an average pooling layer, where the receptive field of the pooling layer is identical to the spatial dimensions of the feature maps. For instance, a $K \times W \times H$-dimensional feature map will be transformed into a $K \times 1 \times 1$-dimensional feature map, where each K value represents the average value over the $W \times H$ dimensions. GAP therefore reduces the number of parameters of the subsequent fully-connected layers by a factor of $W \cdot H$.

We can also see GAP as a structural regularizer that enforces the feature maps to act as confidence maps for the categories of the task. Indeed, average pooling is normally

used to sum out spatial information within each receptive field to improve translation invariance. In the case of GAP, the receptive field is identical to the spatial dimension of the feature maps. The feature map with the most positive neurons and with the highest positive values will be selected as the predicted category, while the other feature maps will be discarded.

Enforcing this correspondence between the feature maps of the convolutional part and the categories of the task has the important advantage of lowering the likelihood of overfitting. The number of neurons sent to the fully-connected part is divided by $W \cdot H$, which substantially reduces the number of weights of the fully-connected layers. The fully-connected part can even be removed entirely by enforcing the last convolutional layer to have the same number of feature maps than the number of categories. This is what Lin et al. (2013) originally proposed in their approach, but nowadays GAP is followed by a single fully-connected layer most of the time. As a result, GAP renders the use of Dropout on the fully-connected layers unnecessary, since the number of weights is considerable smaller. Dropout has been shown to make training longer due to the use of random perturbations, so removing it is beneficial to accelerate training.

Summary of NiN Contributions

In summary, the work of Lin et al. (2013) was influential mostly for GAP rather than MLPconv. GAP is used in most state-of-the-art architectures (He et al., 2016a,b; Huang et al., 2017), but the convolutional operation are still used as they were originally proposed (LeCun et al., 1998).

3.4.3 Overfeat

The 2013 edition of the ILSVRC competition introduced the detection track. Its training set contained about 400,000 images, while its validation set and test set contained about 20,000 and 40,000 images. For the localization track, the dataset was the same as the recognition track, except for the additional labels indicating the location of the objects. Overfeat (Sermanet et al., 2014) was the only approach that entered all tracks (recognition, detection and localization). It won the localization track of the ILSVRC 2013 competition with 29.9% error rate, secured the fourth position on the classification track with 14.2% error rate and the third position of the detection track with 19.4% mean average precision [4].

[4] http://www.image-net.org/challenges/LSVRC/2013/results.php

The convolutional neural networks developed for the ILSVRC 2013 competition were not very different from AlexNet. Overall, the teams were all trying to fine-tune AlexNet introduced the previous year to get better results. Overfeat used the same architecture as AlexNet, with the only difference that they removed the local response normalization layer and increased the number of convolutional kernels. On a side note, we also started to see more teams using ensemble learning to further improve the accuracy of AlexNet. Indeed, the Overfeat submission on the classification track used an ensemble of 7 models with majority voting.

3.4.4 Visual Geometry Group (VGG)

The Visual Geometry Group (VGG) (Simonyan and Zisserman, 2014) architecture secured the first and second place of the localization and classification tracks of the ILSVRC 2014 competition. For the localization track, they obtained the first place with 25.32% error, just under GoogLeNet (Szegedy et al., 2015) with 26.44% error [5]. For the classification track, they obtained the second place with 7.33% top-5 error rate, while GoogLeNet obtained the first place with 6.66% top-5 error rate.

The 2014 edition of ILSVRC marked the advent of truly deep networks. Previous networks like AlexNet and Overfeat had no more than 10 layers, while VGG had 19 layers and GoogLeNet had 22 layers. As a consequence, VGG and GoogLeNet obtained large performance improvements in comparison to the other architectures. For instance, VGG obtained 7.33% error rate on the classification track while Overfeat obtained 14.2%, yet both had the same number of parameters. GoogLeNet even had a lower number of parameters than both Overfeat and AlexNet (with only 6.8M), but still obtained better performances. These results were indicative that increasing depth was beneficial for improving performance.

VGG was influential for another reason. The receptive field of its convolutional layers was much smaller than the receptive field of the previous architectures. For instance, both AlexNet and Overfeat used receptive fields as large as 11×11, while GoogLeNet used receptive fields as large as 5×5. As for VGG, it only used a receptive field of 3×3. What was particular is that its architecture was identical to its predecessors. It was still based on a series of convolutional layers followed by ReLU, interleaved with max pooling. It still even used flattening rather than global average pooling to connect the convolutional part with the fully-connected part.

[5] http://image-net.org/challenges/LSVRC/2014/results

However, this subtle modification of the receptive field had two important advantages. A convolutional layer with a large receptive field can be approximated by several subsequent convolutional layers with a smaller receptive field. For instance, a stack of two 3×3 convolutional layers has an effective receptive field of 5×5, while a stack of three 3×3 convolutional layers has an effective receptive field of 7×7. There is no loss of effective receptive field when a convolutional layer with a large receptive field is replaced with an adequate number of convolutional layers with a smaller receptive field. However, the use of many convolutional layers introduces additional nonlinearities. This makes the network more flexible, which in turn allows it to learn a more discriminative decision functions (Simonyan and Zisserman, 2014). This is the first advantage of using a smaller receptive field.

The second advantage is that replacing a convolutional layer with a large receptive field by a series of convolutional layers with a smaller receptive field reduces the number of parameters. For instance, suppose that both the input and the output of the convolutional layer have C feature maps. A convolutional layer with a receptive field of 7×7 would require learning $7^2 \cdot C^2 = 49C^2$ parameters. In comparison, a stack of three convolutional layers with a receptive field of 3×3 would require learning $3(3^2C^2) = 27C^2$ parameters. The stack of three 3×3 convolutional layers has the same effective receptive field than a single 7×7 convolutional layer, but the stack has fewer parameters to be learned.

The use of convolutional layers with a small 3×3 receptive field can therefore be seen as a form of regularization. It imposes that a convolutional layer with a large receptive field can be approximated by a stack of convolutional layers with smaller receptive fields, fewer parameters and more nonlinear rectification layers. This turned out to be of great importance for the design of neural network architectures, which can be seen nowadays in the majority of architectures that are made of convolutional layers with a receptive field of 3×3.

3.4.5 GoogLeNet

The GoogLeNet (Szegedy et al., 2015) architecture tried to tackle a design question that many deep learning practitioners were overlooking: can the receptive field be learned. At the time, the receptive field was always fixed. AlexNet drew from 3×3 to 11×11, while VGG used 3×3 everywhere. There were not actively trying to optimize the receptive field size. The difficulty, however, in optimizing the receptive field size is that

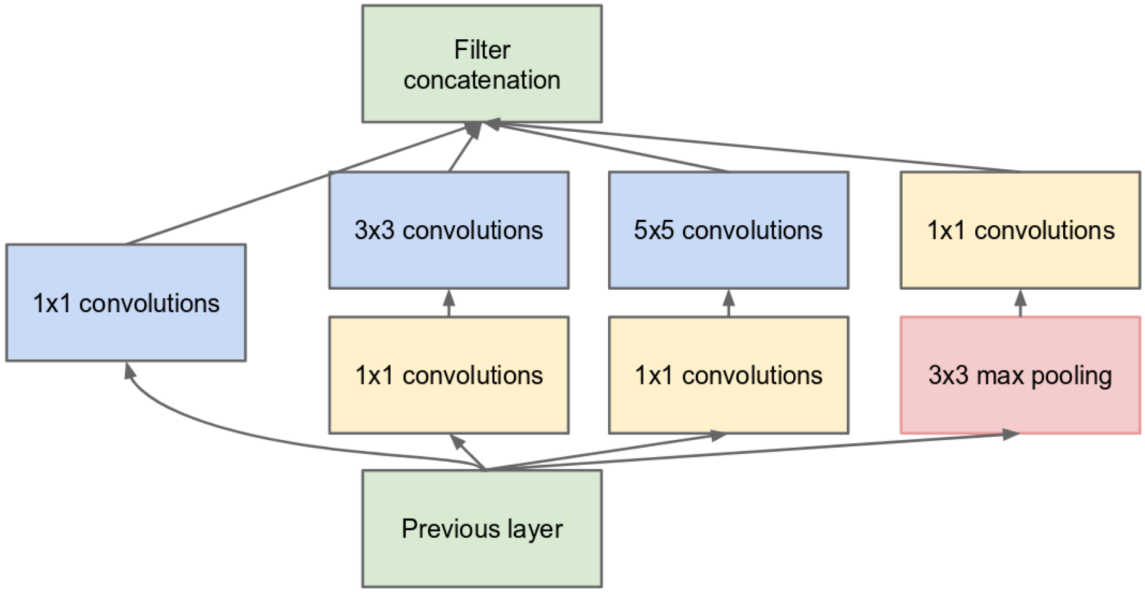


Figure 3.11: The Inception module of the GoogLeNet architecture (Szegedy et al., 2015). The module defines parallel computation, where each convolutional layer has a different receptive field size. It also uses max pooling that does not change the dimensionality of the feature maps in order to add a different information processing layer. The use of 1×1 convolutional layers in yellow reduces the number of feature maps to lower the dimensionality of the output feature maps after depth-wise concatenation. Image taken from (Szegedy et al., 2015).

it cannot be done directly with gradient descent. This is because the size is a discrete value, so the loss E is non-differentiable with respect to it.

To get around this difficulty, Szegedy et al. (2015) introduced the Inception module that propelled the idea of parallel computation within the network. An illustration of the Inception module is shown in Figure 3.11. Up until then, it was usual to view a neural network as a machine of serial computations. The Inception module shattered this view by using many parallel paths, each composed of a series of convolutional layer followed by a rectification layer. They took the same feature maps as input, but computed different output feature maps and sent them a depth-wise concatenation layer that merged them together.

Using their Inception module, Szegedy et al. (2015) were able to optimize for the receptive field size *indirectly* by using a different one for each path, simultaneously. As seen in Figure 3.11, one has a size of 1×1, another one has a size of 3×3 and the last one as a size of 5×5. Combined with the concatenation layer at the end, the module allowed training to guide the network toward the most suitable receptive field size, since it was

able to choose between many of them. This thus acted as a workaround solution to the non-differentiability difficulty of the loss.

As a result, the Inception module proposed to answer the question of how to choose the receptive field size by simply using many of them in parallel. By doing so, Szegedy et al. (2015) created one of the first network architecture that strayed from the standard design principle that was used at the time. This important idea turned out to be important for network design. It has been re-used in many network architectures since then, especially in architectures with skip connections like ResNet and DenseNet. Both the ResNet and DenseNet are based on this notion of parallel computation, and they both turned out to be two of the most important architectures of the past decade. We will discuss ResNet and DenseNet in the following section, but first, we introduce All-CNN that made us reconsider an important principle of network design.

3.4.6 All-CNN

With their All Convolutional Neural Network (All-CNN) architecture, Springenberg et al. (2015) questioned the established belief that max pooling was always required to reduce the spatial dimensions of the feature map. In their work, they proposed to replace each max pooling layer with a convolutional layer with a matching stride. For instance, a max pooling layer with a receptive field of 2×2 and a stride of 2 would transform a $W \times H \times C$-dimensional input feature map into a $\lfloor W/2 \rfloor \times \lfloor H/2 \rfloor \times C$-dimensional output feature map. We can create a convolutional layer that modifies the input feature map in an identical way, by using C convolutional kernels with a receptive field of 2×2 and a stride of 2. In this case, instead of computing the maximum value over all $2 \cdot 2$ values of the receptive field for each C feature map independently, the pooling layer will compute C dot products between each kernel and $2 \cdot 2 \cdot C$ values of the receptive field of each feature map. Springenberg et al. (2015) conducted several experiments with these type of pooling layers and showed that they could get similar and sometimes better performances with them.

The use of convolutional pooling layers can be seen as using a parametric version of the traditional max pooling layers. Recall that max pooling defines no parameters, which means that its pooling behavior is fixed throughout training. In contrast, convolutional pooling layers are parametric, which means that they can adjust their pooling behavior during training. They can thus learn different kinds of pooling, which adds another layer of parameterization to the network.

3.4.7 Residual Network (ResNet)

The Residual Network (ResNet) (He et al., 2016a) secured the first position of the classification track of the ILSVRC 2015 competition with a staggering 3.57% top-5 error rate. This corresponded to a relative improvement of 46% in comparison to GoogLeNet of the previous year, which obtained the first place with 6.66%. Recall that in 2012, AlexNet obtained a top-5 error rate of 16.42%, while the best entry of the previous year obtained 25.77%. This corresponded to a relative improvement of 36% in 2012, which is of the same order of magnitude as of ResNet that we observed in 2015.

The introduction of ResNet marked the advent of immensely deep networks. As we saw previously, VGG pushed the depth to 19 layers, while GoogLeNet pushed it further to 22 layers. This was already a large increase in comparison to AlexNet with 8 layers. He et al. (2016a) completely changed the picture by submitting a ResNet with a depth of 152 layers for the classification track of ILSVRC 2015. Their ResNet was 8x deeper than VGG and 7x deeper than GoogLeNet. In subsequent experiments on the smaller dataset CIFAR-10, they were able to successfully train a ResNet with 1,000 layers (He et al., 2016a). These results came up as a surprise for many in the deep learning community, as no one was able to train networks much deeper than VGG and GoogLeNet prior to that.

The reason why it was difficult to train deeper networks was due to the *degradation* problem. This problem was observed by He et al. (2016a) during their experiments on varying the depth of their network. They observed that the train accuracy would stop to improve once the network reached a certain depth threshold. They further observed that the train accuracy would start to degrade as they continued to increase depth pass that threshold point. In other words, they observed a plateau where adding more layers would no longer improve train accuracy, but rather degrade it. This was rather unexpected because a network becomes more complex and more able to overfit the training data when its depth increases. This means that the train accuracy should have decreased as depth increased, but the experimental results suggested otherwise.

The degradation problem of the train accuracy indicates that not all systems are easily optimized. To better illustrate this problem, consider both a shallow and a deep architecture, where the deep architecture is created by adding more layers to the shallow architecture. There exists *by construction* a deep architecture that is equivalent to the shallow architecture. This architecture is created by simply adding *identity* layers to the shallow architecture, where an identity layer outputs its input feature maps without

changing them. It is easy to see that this deep architecture has the same accuracy as its shallow counterpart, even though it is deeper. This suggests that a deep network should never produce a worse train accuracy than its shallower counterpart. The fact that we instead observe a degradation of the train accuracy when adding more layers indicates that the solvers are unable to learn at least the identity function for the added layers. This point towards the fact that learning the identity function is rather difficult in the context of standard feedforward architectures.

The idea of He et al. (2016a) was thus to create a module that can easily learn this identity function. To do so, they introduced the key principle of *residual learning*. To understand the difference between standard learning and residual learning, consider a standard feedforward module defined as a parametric transformation $\mathcal{H}(x)$. For instance, $\mathcal{H}$ could be a series of nonlinear layers like multiple convolutional layers, followed by a nonlinear rectification layer. In this case, learning the identity function with $\mathcal{H}$ amounts to learning the proper weights and biases of the nonlinear mapping, such that $\mathcal{H}(x) = x$. This is a hard problem. The idea of residual learning is to rather consider the residual mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$, and recast the original problem of learning the mapping $\mathcal{H}$ as learning the mapping $\mathcal{F}(x) + x$ instead. With this new formulation, it becomes easy to learn the identity function with a stack of nonlinear transformations. The solver simply needs to push the parameters of $\mathcal{F}$ to zero, such that $\mathcal{F}(x) = 0$ and the mapping $\mathcal{F}(x) + x = x$ becomes an identity mapping.

A module that implements residual learning has the following general formulation:

$$y_l = \mathcal{S}(x_l) + \mathcal{F}(x_l) \tag{3.4.1}$$

$$x_{l+1} = g(y_l)\,, \tag{3.4.2}$$

where $\mathcal{S}$ are the *skip connections*, $\mathcal{F}$ is a standard feedforward module and g is an optional function. In the original formulation of the residual module by He et al. (2016a), g was defined as the ReLU activation function, while $\mathcal{S}$ was defined differently depending on $\mathcal{F}$. Indeed, $\mathcal{S}$ was defined as the identity function when $\mathcal{F}$ did not change the dimensionality of x, but $\mathcal{S}$ was defined as a 1×1 convolutional layer with a stride larger than one when $\mathcal{F}$ changed the dimensionality of x.

In a subsequent experimental study, He et al. (2016b) showed that defining g as the identity function gave better performances than defining g as ReLU. Within this new formulation, a module that implements residual learning can be defined simply as fol-

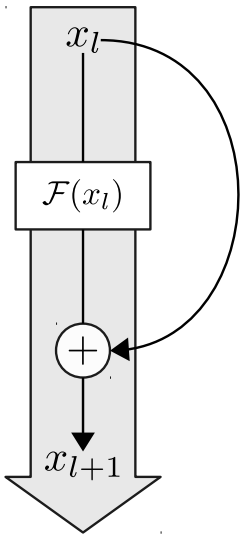


Figure 3.12: Illustration of a residual module that implements residual learning with $x_{l+1} = \mathcal{F}(x_l) + x_l$. The module can be seen as a standard feedforward module $\mathcal{F}$ with an additive skip connection that implements the identity mapping.

lows:

$$x_{l+1} = \mathcal{F}(x_l) + x_l \,, \tag{3.4.3}$$

as shown in Figure 3.12. In other words, it can be seen as a standard feedforward module $\mathcal{F}$ with an additive skip connection that implements the *identity* mapping. Note that with the formulation of Eq. (3.4.3), $\mathcal{F}$ can be easily deactivated by simply pushing its parameters to zero, which would result in the identity function $x_{l+1} = 0 + x_l$.

The use of additive skip connections has many advantages, but we still do not fully understand their impact. In the work of He et al. (2016b), it was shown that the use of additive skip connections, like the one of $x_{l+1} = \mathcal{F}(x_l) + x_l$, leads to interesting backward propagation properties. In particular, they showed that the gradient of the loss function with respect to any output feature map x_l has the following formulation:

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L} + \frac{\partial E}{\partial x_L} \left(\frac{\partial}{\partial x_l} \sum_{i=l}^{L-1} \mathcal{F}(x_i) \right) . \tag{3.4.4}$$

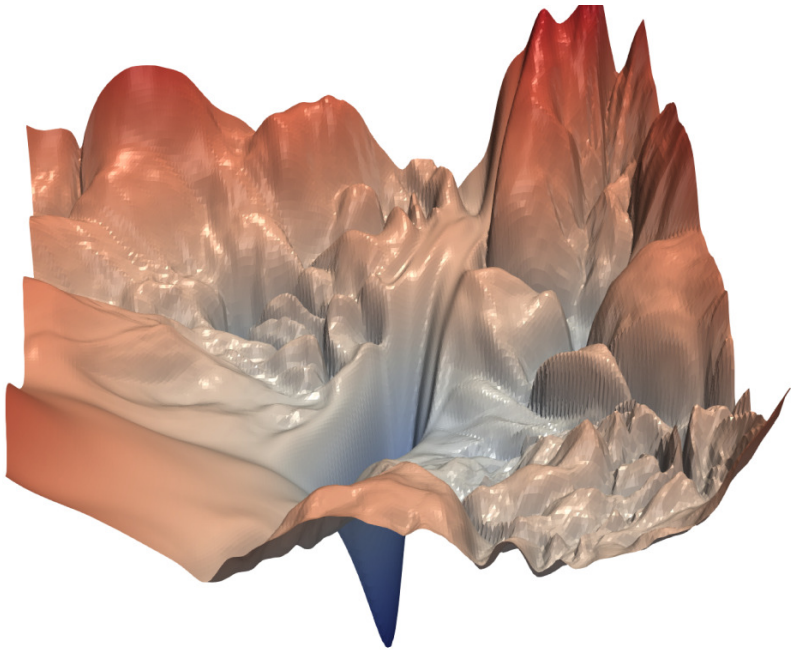
Eq. (3.4.4) indicates that the gradient $\dfrac{\partial E}{\partial x_l}$ can be decomposed as a sum of two terms. The first one is $\dfrac{\partial E}{\partial x_L}$, which propagates the error signal of the loss function to each layer of the hierarchy, and the second one is $\dfrac{\partial E}{\partial x_L} \left(\dfrac{\partial}{\partial x_l} \sum_{i=l}^{L-1} \mathcal{F}(x_i) \right)$, which propagates

through the layers. The fact that $\frac{\partial E}{\partial x_L}$ is added to the other term suggests that it is unlikely that $\frac{\partial E}{\partial x_L}$ is cancelled out by $\frac{\partial E}{\partial x_L}\left(\frac{\partial}{\partial x_l}\sum_{i=l}^{L-1}\mathcal{F}(x_i)\right)$, because both terms are rarely the opposite of each other. This implies that the gradient $\frac{\partial E}{\partial x_l}$ at layer l rarely vanishes, even when the weights or the activations at the layers above l are arbitrarily small, which would make the term $\frac{\partial E}{\partial x_L}\left(\frac{\partial}{\partial x_l}\sum_{i=l}^{L-1}\mathcal{F}(x_i)\right)$ close to zero. As a result, the information of the error signal is always directly propagated backward to every shallower layer, which lowers the likelihood of vanishing gradients.

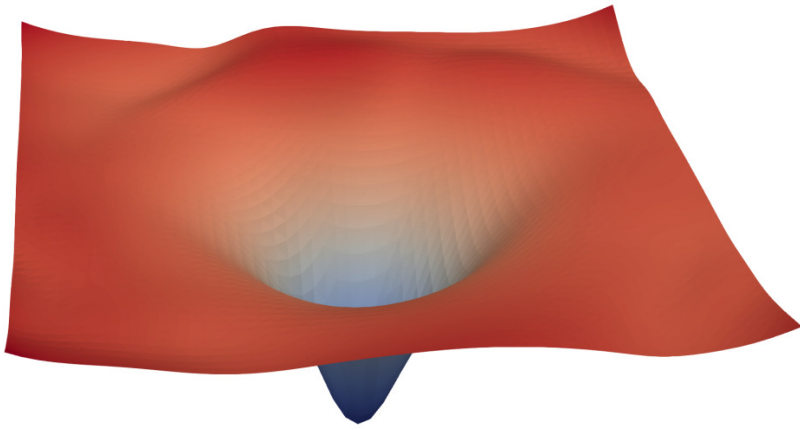
The work of Veit et al. (2016) shed light on another interesting advantages of additive skip connections. They showed that a residual network can be interpreted as a collection of many paths of different length. What is interesting is that most paths are much shallower than the effective depth of the network. They further performed an ablation study that revealed that these paths have an ensemble-like behavior. They do not strongly depend on each other, with performance smoothly correlating with the number of valid paths. Removing the residual mapping $\mathcal{F}$ (using the identity skip connection instead) from a single residual module has little to no effect on accuracy, even though all residual modules are learned jointly. In comparison, the standard feedforward VGG network predicted no better than random guesses when layers are removed, even when only one is removed.

These results suggested that extremely deep residual networks with hundreds of layers do not produce better results due to their depth alone. They in fact suggested that residual networks produce better performances in part due to behaving like ensemble methods, which are known for improving statistical generalization. This key insight revealed that depth is still an open research question, even though the effective depth of a deep residual network can be many times larger than standard feedforward networks like VGG and GoogLeNet.

Another work has studied the shape of the loss function to better understand why the use of skip connections improves training and performance (Li et al., 2018). In their work, Li et al. (2018) analyzed the difference between the loss function landscape of a network without skip connections and a network with skip connections. To take into account the fact that the dimensionality of the loss function is dependent on the number of weights, and that a network can have millions of weights, they proposed a method called *filter normalization* to create a low-dimensional approximation of the



(a) without skip connections



(b) with skip connections

Figure 3.13: The difference between the loss function landscape of a network without skip connections (a) and a network with skip connections (b). The use of skip connections makes the loss function smoother, which makes the search for a good minimizer easier. Figure taken from Li et al. (2018).

loss function. With their approach, they plotted the shape of the loss function in two dimensions. Their results are shown in Figure 3.13. One can see that with a standard feedforward network, the loss function is non-convex and has many local minima, while with skip connections the loss function is significantly smoother. As a result, skip connections facilitate the search for a good minimizer because the loss function is easier to minimize.

Overall, we still do not have a complete understanding of the behavior induced when using skip connections. But we know from the experimental evaluations of He et al. (2016b) and He et al. (2016a), as well as from the theorical studies of He et al. (2016b), Veit et al. (2016) and Li et al. (2018), that skip connections improves training, accuracy, lowers the likelihood of vanishing gradients and are easy to incorporate in any existing network architecture. They have many advantages and should always be considered first when designing novel network architectures.

3.4.8 Densely Connected Network (DenseNet)

Densely Connected Network (DenseNet) (Huang et al., 2017) is a network architecture similar to ResNet, in the sense that DenseNet is also based on skip connections. The main difference between ResNet and DenseNet is that ResNet uses additive skip connections, while DenseNet uses concatenative skip connections. The base module that implements concatenative skip connections in DenseNet is defined as follows:

$$x_{l+1} = [x_l, \mathcal{F}(x_l)] \tag{3.4.5}$$

where $[\cdot]$ stands for depth-wise concatenation.

To better illustrates the difference between DenseNet and ResNet, we can redesign the ResNet module of Eq. (3.4.3) and the DenseNet module of Eq. (3.4.5) using another formulation with a new mapping $\mathcal{H}$. For ResNet, Eq. (3.4.3) can be redesigned as follows:

$$x_{l+1} = \mathcal{H}(x_1 + x_2 + \ldots + x_l)\,, \tag{3.4.6}$$

while for DenseNet, Eq. (3.4.5) can be redesigned as follows:

$$x_{l+1} = \mathcal{H}([x_1, x_2, \ldots, x_l])\,. \tag{3.4.7}$$

This new formulation reveals the main difference between ResNet and DenseNet. With ResNet, the information about the feature maps of the previous layers $l-1$, $l-2$, ...,

up to 1 is blended together into the feature maps $x_1 + x_2 + \ldots + x_l$. With DenseNet, that information is fully accessible, since each feature map x_l is kept separated. Thus, the use of each x_l in the input of mapping $\mathcal{H}$ can therefore be seen as adding one skip connection to each previous feature map x_l. Hence the name *densely* connected.

This implies that concatenative skip connections enjoy the same advantage of lowering the likelihood of vanishing gradients as additive skip connections. The difference is that concatenative skip connections have the additional advantage of maintaining the integrity of the information computed at the lower layers. That low-level information can be used *as is* in the higher layers of the hierarchy, which has been shown to improve performance (Huang et al., 2017).

3.4.9 Comparing Performance, Size and Operations

A recent analysis by Canziani et al. (2016) compared the presented CNNs on their accuracy on the ImageNet recognition task, their size in terms of number of parameters and their number operations required for a single forward pass. These three characteristics are important to consider when choosing an architecture, especially in a context of limited computational resources like mobile computing or embedded systems. The performance on ImageNet ranks the architectures on *accuracy*, the number of parameters ranks the architectures on *memory requirement* and the number of operations ranks the architectures on *energy consumption*.

The results of Canziani et al. (2016)'s comparative study are presented in Figure 3.14. Each circle corresponds to a network architecture. The size of the circle illustrates the number of parameters, where the reference scale that associates the size of the circle to the number of parameters is shown at the bottom of the figure. The abscissa value of each circle corresponds to the number of operations required for a single pass (in G-Ops), while the ordinate value of each circle corresponds to the Top-1 accuracy on the ImageNet object recognition task. The best architecture is the one with the smallest circle and the most top left.

The results in Figure 3.14 show that ResNet-50 is the best architecture overall, closely followed by Inception-v3. With approximately 35M parameters, ResNet-50 reaches about 76% Top-1 accuracy on ImageNet and requires about 7 G-Ops for a single forward pass. In comparison, AlexNet, which is the most popular architecture in the deep learning community, requires 3 G-Ops for a single forward pass, has about 65M parameters and only reaches about 55% Top-1 accuracy. These results corroborate with our

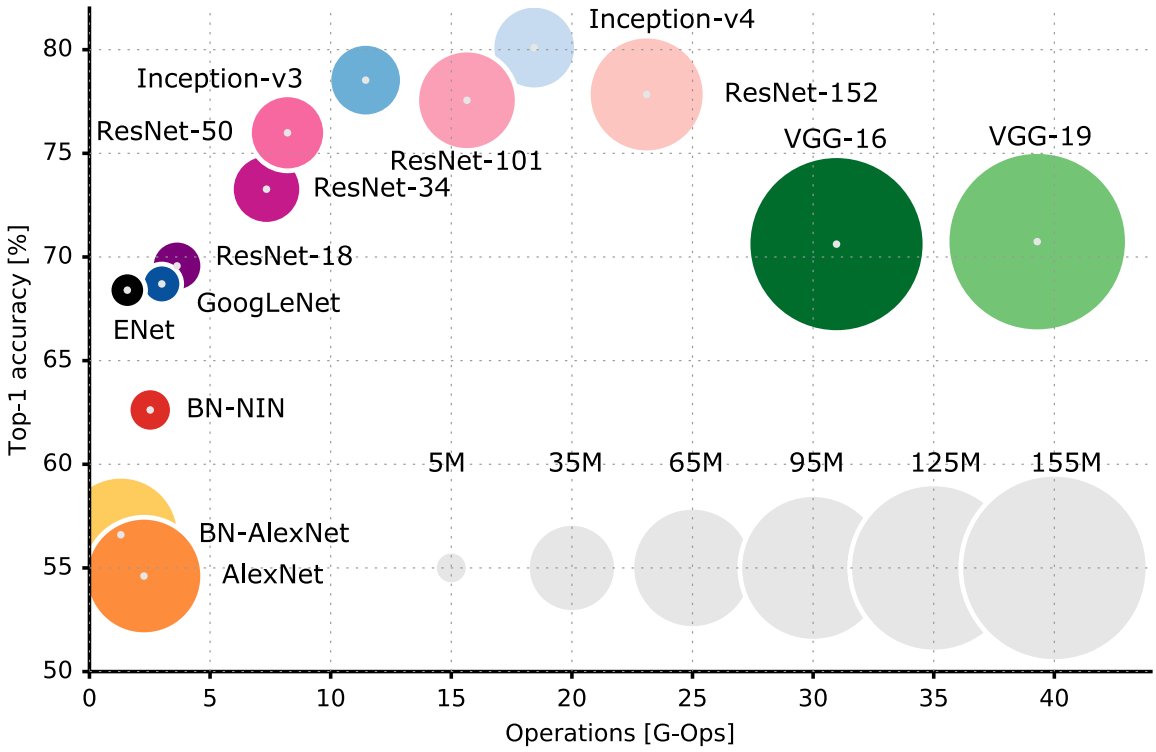


Figure 3.14: Comparing convolutional neural network architectures on their accuracy on the ImageNet recognition task (ordinate), their number of operations required for a single forward pass (abscissa) and the number of parameters (size of circle). The best overall architecture is ResNet-50, which corroborates with our understanding of how residual learning improves training and generalization performance. Image taken from Canziani et al. (2016).

theoretical understanding of how residual learning improves training and generalization performances (ref. Section 3.4.7 for more details). Thus, feedforward neural networks should preferably incorporate skip connections in their architectures in order to fully take advantage of residual learning.

3.5 Conclusion

In this chapter, we introduced the framework of Deep Learning (DL). The goal of DL is to train a deep neural network, which can be seen as learning the parameters of a parametric function in *parametric function approximation*. We further explained the importance of the activation function when designing the architecture of the network. Choosing an activation function requires taking into account many considerations, such as vanishing gradients, dead units and bias shift. To this end, we presented different

activation functions that address each of these considerations, such as the ReLU, the LReLU and the ELU. We further explained that they can define parameters that control different aspects of their shape, which can be learned during training with gradient descent. This lead us to consider the family of parametric activation functions. We review important works that tried to further improve the flexibility of the ReLU by designing different types of parameterization, such as the PReLU, the APL, the SReLU and the Maxout. Finally, we reviewed key convolutional network architectures, including the recent ResNet, that implements residual learning.

In the following chapter, we will introduce another framework for representation learning that focuses on learning many tasks in parallel. We will introduce the framework of *Multitask Learning*.

Chapter 4

Background Material on Multitask Learning for Deep Learning

In this chapter, we present background material on Multitask Learning (MTL), with an emphasis on deep learning approaches. MTL is a representation learning framework that implements the *shared factors across tasks* prior belief that we introduced in Section 1.2.6. We start with the definition of MTL in Section 4.1, which can be summarized as the problem of learning many tasks in parallel. We also explain the difference between soft parameter sharing and hard parameter sharing, which are two MTL implementations. Then, we review important deep learning works on soft parameter sharing in Section 4.2, such as regularization terms, the Cross-Stitch block, the Sluice block and the Cross-Residual block. We also point out to a limitation of these approaches that we will address in our contribution in Chapter 7. We finally conclude the chapter in Section 4.3.

4.1 Definition

Multitask Learning (MTL) is a field of machine learning that has been around for more than two decades (Caruana, 1997; Evgeniou and Pontil, 2004; Argyriou et al., 2007; Liu et al., 2009; Zhang et al., 2012, 2014). It was put forward by Caruana (1997) in his seminal work, who provided a definition that is nowadays still widely cited:

> Multitask Learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias. It does this by learning tasks in parallel

> while using a shared representation; what is learned for each task can help other tasks be learned better. (Caruana, 1997)

Since then, MTL has proven its value in several domains over the years. It has become a dominant field of machine learning with many influential works (see (Zhang and Zhou, 2014) for a review). Moreover, recent major advances in deep learning opened up opportunities for novel contributions. Works on grasping (Pinto and Gupta, 2017), pedestrian detection (Tian et al., 2015), natural language processing (Liu et al., 2015a), face recognition (Yim et al., 2015; Yin and Liu, 2017) and object detection (Misra et al., 2016) helped MTL make a resurgence in the deep learning community. They have shown the potential of MTL to improve the generalization ability of deep networks, fuelling its growth in popularity.

MTL falls into the category of *transfer learning* (Pan and Yang, 2010), which is a broad field of machine learning that focuses on knowledge transfer. The knowledge gained while solving task one is used to solve task two, and the new knowledge gained while solving task two is used to refine the knowledge gained while solving task one, and so forth. It draws on the long-standing literature of *transfer of learning* in psychology (Ellis, 1965) that studies the way people learn. The transfer of learning in psychology is interested in how the improvement of one mental function affects the improvement of related mental functions. It describes the processes with which past experiences affect learning, performance and adaptation in new situations. Transfer of learning in people has been used as a biological inspiration to put forward the notion of knowledge transfer in MTL.

The goal in MTL is to solve a main task while drawing knowledge from other related tasks *in parallel*. This particularity of learning multiple tasks in parallel is what differentiates MTL from other fields of transfer learning, like *domain adaptation* (Daume III and Marcu, 2006) or *learning to learn* (Thrun and Pratt, 2012). In MTL, all tasks are learned at the same time, even though the focus is to only perform well on the main one. As Caruana (1997) put into words: "Usually, we do not care how well extra tasks are learned; their sole purpose is to help the main task be learned better." (Caruana, 1997).

The definition of MTL by Caruana (1997) introduces the important principle of *inductive bias*. An inductive bias is a set of assumptions about the target function or mapping that is used by the learner to predict the result of inputs never seen before (Mitchell, 1980). It can be seen as a set of preferences for some hypotheses over others, when the

learner tries to explain inputs that fall outside the training data distribution.

We can illustrate an example of inductive bias with the linear model $f(x) = \theta^\top x$ introduced in Chapter . The inductive bias in a linear model is described as follows: the relation between the input features x and the target value y is linear. A linear model uses this inductive bias to compute the target value $f(x)$ of inputs x it has never seen by assuming that x and y are always linearly related. In other words, the predicted target value $f(x)$ associated with any inputs x will always be a weighted sum $\theta^\top x$, no matter if the input x has been observed during training or not.

The inductive bias is related to the notion of generalization. Indeed, approaches with an improper inductive bias usually do not generalize well, while approaches with a proper inductive bias can generalize well given the proper training framework. Consider for instance the problem of linear regression. Suppose that we try to capture the relation between the input and the output with a linear model by minimizing the mean squared error. In the case where the underlying relation is nonlinear, the linear model will fail to correctly capture the underlying relation no matter what optimization approach is used. The linear model will not correctly predict the nonlinear relation between the input and output outside (and even inside) the data distribution because the inductive bias is not the proper one.

On the other hand, in the case where the underlying relation is truly linear, the linear model can succeed to capture the underlying relation only if the mean squared error is correctly optimized. For instance, outliers can have a large impact when learning the weights θ. The presence of a single outlier can affect the minimization of the mean squared error in such a way that the linear model is shifted toward the statistically unrepresentative outlier. When this happens, the learned weights fail to correctly capture the underlying linear relation, even though the inductive bias is the proper one. But if the weights are correctly optimized, the linear model will be able to correctly predict the relation between the input and the output everywhere on the input domain.

In the case of MTL, the inductive bias is defined as follows: approaches are trained to have utility across many tasks. The key insight is that the tasks are learned simultaneously. This makes available a rich source of information that is not available when the tasks are learned in isolation. This source of information represents domain information that is shared among many problems, which helps promote generalization. A MTL approach can access this source of information through the training signals of the related tasks, and thus learn a shared feature representation with better generalization

qualities.

4.1.1 Multitask Learning Training Objective

The main difference between MTL and standard Singletask Learning (STL) is the definition of the training objective. The goal of MTL is to learn multiple tasks in parallel in order to improve the performance on the main task. To this end, MTL defines many training objectives, one for the main task and one for each related task. The objectives take the form of loss functions and are combined with a weighted sum, as follows:

$$E = E_0 + \sum_{t=1}^{T} \gamma_t E_t \,, \tag{4.1.1}$$

where E_0 is the error loss function of the main task, T is the number of related tasks, E_t are the error loss functions of the related tasks and γ_t are the task-specific strength hyper-parameters. The hyper-parameters γ_t control the importance of minimizing each function E_t. As we can see in Eq. (4.1.1), minimizing E will simultaneously minimize the error E_0 on the main task, as well as the errors E_t on the auxiliary tasks, as wanted.

The training objective defined in Eq. (4.1.1) is generic and can appear in many contexts. It can even appear in a STL context, especially when regularization is used. In this case, the related tasks are sometimes referred to as *auxiliary loss functions*. For instance, adding ℓ_1 or ℓ_2 regularization to a supervised learning problem can be seen as defining an MTL objective. The main task is the original training objective, while the related task is the regularization objective. As in MTL, the goal is still to obtain the best performance on the main task. But with regularization, we add an inductive bias favoring specific types of solutions, such as parcimonious solutions in the case of ℓ_1 or low magnitude solutions in the case of ℓ_2.

The formulation of Eq. (4.1.1) allows us to define a multitask training objective. One question that remains is how to implicitly share the domain information between the tasks. In the following section, we will describe two main approaches to do so.

4.1.2 Domain Information Sharing with Hard and Soft Parameter Sharing

In the context of neural networks and deep learning, MTL approaches generally fall into the following two categories: hard parameter sharing and soft parameter sharing (Ruder,

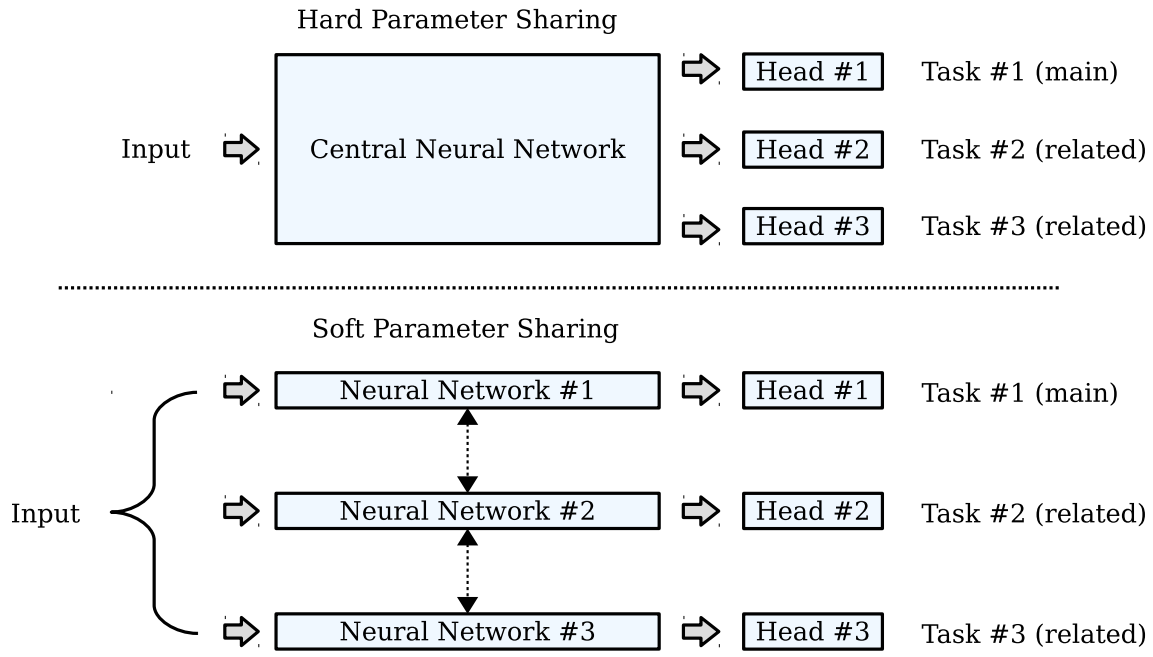


Figure 4.1: MTL approaches in deep learning generally fall into two categories: hard parameter sharing and soft paramter sharing. The top figure corresponds to hard parameter sharing, while the bottom figure corresponds to soft parameter sharing. Hard parameter sharing defines a shared central newtork, with one head per task. Soft parameter sharing defines one network per task, but uses a sharing mechanism (the dotted arrows) to make available each task-specific information to all networks. Soft parameter sharing is growing in popularity because the sharing mechanism is more flexible than the central network in hard parameter sharing.

2017). An illustration of the difference between the two is shown in Figure 4.1. The top part corresponds to hard parameter sharing, while the bottom part corresponds to soft parameter sharing. The figure shows an example of a MTL problem with one main task and two related tasks.

As presented in Figure 4.1, approaches in the hard parameter sharing category define a shared central network with one head for each task, where the heads are usually fully-connected networks with no more than one or two layers. The central network takes as input the observation and outputs a shared feature representation. Each head then takes the shared feature representation as input and predicts the output value for their corresponding task. Note that the separation between the central section and the heads is artificial. It is needed only to make explicit that the heads are influenced only by the training signal of their corresponding task, while the central network is influenced by the training signals of all tasks. The overall network is still trained end-to-end using backprop, to the difference that the gradient incorporates the error signals coming for

all heads.

Hard parameter sharing dates back to the original work of Caruana (1997) on MTL and is the most common of the two categories. It is still used nowadays to enable knowledge sharing between tasks (Ranjan et al., 2017; Zhang et al., 2014; Pinto and Gupta, 2017; Yin and Liu, 2017; Chen et al., 2018b), with a recent success on mastering the game of Go (Silver et al., 2017) that has attracted a lot of attention. In their AlphaGo Zero architecture, Silver et al. (2017) defined a hard parameter sharing CNN that outputs simultaneously the value and the policy of a given state in a reinforcement learning framework.

On the other hand, approaches in the soft parameter sharing category define one network for each task, but equip them with a sharing mechanism. The sharing mechanism can take the form of a regularization term that is used as an auxiliary loss in the training objective (Yang and Hospedales, 2016b), or can be defined as a component of the network with trainable parameters that influence both the forward pass and the backward pass (Misra et al., 2016). In each case, each task-specific network takes the same observation as input and directly predicts the output value for their corresponding task. Unlike hard parameter sharing, there is no central network that is influenced by the training signals of every task. It comes down to the sharing mechanism to allow each task-specific information to be available to all networks. The sharing mechanism is necessary to allow knowledge sharing between the tasks, otherwise the overall network would only be a set of independent task-specific networks.

Soft parameter sharing is more recent and is growing in popularity (Ruder, 2017). The first occurrence of the name *soft parameter sharing* can be traced back to the work of Duong et al. (2015) on natural language processing a few years ago. Since then, many works have been done in this field (Yang and Hospedales, 2016b,a; Misra et al., 2016; Ruder, 2017). Soft parameter sharing is gaining attention in the community and some see it as the next step for MTL (Ruder, 2017).

The reason explaining why soft parameter sharing is gaining in popularity is due to the sharing mechanism (Ruder, 2017). In hard parameter sharing, the central network outputs a single shared representation that is used by every heads. This creates a competition between the task-specific features that encourages the shared representation to minimize all objectives. This competition can be beneficial when the tasks are related, but can also be detrimental when the tasks are different. It can be hard for the network to learn features specific to only one task, since features of this kind have value only

for the corresponding task. They do not usually improve the performance of the other related tasks, so they are not favored during training. This problem is further accentuated when the shared representation has low dimensionality and when the number of tasks is large (Ruder, 2017).

The sharing mechanism was proposed as an alternative to deal with this problem. With the sharing mechanism, there is less competition because there is no single shared feature representation. The task-specific networks can more easily learn a feature representation specific to their task, even though there is still an incentive to simultaneously minimize all objectives. The sharing mechanism can also be designed with trainable parameters, so that the network can learn to control the sharing of information between the networks. As a result, there is less interference with the sharing mechanism than with the central network, even though there is still information sharing between each task.

Soft parameter sharing will be the main subject of our contribution in Chapter 7. For this reason, we will focus our review on soft parameter sharing rather than hard parameter sharing.

4.2 A review of Important Soft Parameter Sharing Mechanisms

In this section, we review important soft parameter sharing mechanisms. We first review sharing mechanisms defined as a regularization term then introduce sharing mechanisms defined as a trainable component of the network. We cover the Cross-Stitch block, the Sluice block and the Cross-Residual block. We will also describe a limitation with these approaches that we will address in our contribution in Chapter 7.

4.2.1 The Sharing Mechanism Defined as a Regularization Term

A soft parameter sharing mechanism defined as a regularization term applies a penalty function P on the parameters $\theta_1, \ldots, \theta_T$ of all task-specific networks. The goal is to link the parameters θ_t together in order to share the training signals of each task. The value of the penalty function is added to the multitask training objective and is minimized

jointly. In general, the training objective looks as follows:

$$E = E_0 + \sum_{t=1}^{T} \gamma_t E_t + \lambda P(\theta_1, \ldots, \theta_T)\,, \tag{4.2.1}$$

where E_0 and E_t are the main and task-specific error loss functions (as described in Section 4.1.1) and λ is a hyper-parameter that determines the importance of minimizing the penalty function P. Note that we define λ as a scalar without loss of generality.

Several approaches based on regularization terms for deep learning have been proposed over the past few years. In the following sections, we will review the squared Euclidean distance, the multilinear relationship network and the tensor trace norm. We then follow with a discussion of the inconvenience of defining the sharing mechanism as a regularization term.

Squared Euclidean Distance.

One regularization term has been proposed to perform cross-lingual parameter sharing for the problem of dependency parsing (Duong et al., 2015). In their approach, the neural parser of the target language is regularized such that its weights are similar to the weights of the neural parser of the source language (Duong et al., 2015). The regularization term is defined as the squared Euclidean distance between the weights of the target and source neural parser. The idea is to help the networks to more easily learn the grammatical structure of their language, since what is learned for one language can be used to learn the other language better.

Multilinear Relationship Networks.

Another approach proposed to model task relationships from a Bayesian point of view (Long and Wang, 2015). We have seen in Section 2.1 that both the feature coding and the dictionary learning optimization problems of sparse dictionary learning can be defined as computing the maximum *a posteriori* (MAP) estimate of the parameters of the generative model. The MAP estimate is a standard tool to associate the optimization problem of a learning algorithm to a probabilistic formulation of a generative model (Robert, 2014). Similarly, Long and Wang (2015) started from the probabilistic formulation of their multitask classification problem and added a prior distribution on all task-specific weights of the penultimate fully-connected layers. They then expressed the MAP estimate as an optimization problem and obtained a regularization term based on the selected prior. They chose as prior distribution the tensor normal distribution

with hyper-parameter Σ_l, which models the relationships between all tasks. The hyper-parameter Σ_l is initialized to exhibit task independence at first, but is updated at each step to find the proper task relationships.

Tensor Trace Norm

A recent approach proposed to have a specific sharing mechanism that encourages parameter reusability (Yang and Hospedales, 2016b). The idea is to define the regularization term from the point of view of the optimization problem, in order to encourage each task-specific network to re-use the parameters of the other networks. Yang and Hospedales (2016b) proposed penalizing the tensor trace norm of the tensor of concatenated task-specific weights at each layer. The tensor trace norm is defined as the standard trace norm applied over a certain tensor unfolding, where the trace norm is defined as the sum of singular values. Yang and Hospedales (2016b) proposed three different tensor trace norms, each encouraging different forms of parameter reusability. Some can even encourage within-task parameter reusability, that is, sharing across the filters of the convolutional layers.

4.2.2 An Inconvenience of Defining the Sharing Mechanism as a Regularization Term

Approaches based on a regularization term have the appealing property that the regularization term can be designed to have a well-chosen influence during training. However, its main drawback is the need to finetune its hyper-parameter λ in Eq. 4.2.1. Tuning λ is critical for the proper functioning of the method. When λ is too large, the regularization term overpowers the main objective and the network fails to perform well on the main task. On the contrary, when λ is too small, the regularization term fails to properly regularize the network and is rendered useless during training. A proper balance between all the elements in Eq. 4.2.1 is essential, but it is hard to attain. It requires careful tuning of λ that can necessitate many trials and errors, which can make the overall training process computationally demanding.

4.2.3 The Sharing Mechanism as a Trainable Component

To address this limitation, the sharing mechanism can be defied as a trainable component of the network. The main difference between the trainable component and the regularization term is that the former is used during both the forward and backward

pass, while the latter is only used during the backward pass. The regularization term is applied on the weights or on the activations *a posteriori*, while the component is applied *during* the forward computation. Thus, it takes part in the computation of the output value rather than only acting on it afterwards. As a result, the trainable component removes the penalty function $P(\theta_1, \ldots, \theta_T)$ and its hyper-parameter λ in Eq. 4.2.1. This is one of the main advantages of the component.

The component can also have parameters that can be learned with gradient descent during training, since we can compute the gradient of the loss function with respect to its parameters. In comparison, learning hyper-parameter λ with gradient descent would not be possible. This is because λ multiplies the penalty function $P(\theta_1, \ldots, \theta_T)$, which means that a straightforward solution to the minimization problem is to always have $\lambda = 0$. The fact that the sharing mechanism can have trainable parameters opens up the opportunity to learn a component with better domain information sharing abilities. We can design the sharing mechanism according to our specifications and let the data guide training towards to proper parameter solution. The approaches that we now present define the sharing mechanism in such a way.

Cross-Stitch Block

The idea of the Cross-Stitch block (Misra et al., 2016) is to define the sharing mechanism as trainable linear combinations of task-specific feature representations. To illustrate the approach, we define x_1 and x_2 to represent the feature representations computed by the networks for tasks 1 and 2. Without loss of generality, we dropped the layer index and only use two tasks. A cross-stitch block with parameters $\alpha_{i,j}$ is defined as follows:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} \\ \alpha_{2,1} & \alpha_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \tag{4.2.2}$$

where y_1 and y_2 are the output feature representations of the cross-stitch block. In other words, the output feature representation y_1 for task 1 is defined as a linear combination $\alpha_{1,1}x_1 + \alpha_{1,2}x_2$ of the input feature representation x_1 and x_2. Similarly for task 2, the output feature representation y_2 is defined as a linear combination $\alpha_{2,1}x_1 + \alpha_{2,2}x_2$ of the input feature representation x_1 and x_2. The multiplications by alpha is a simple scalar multiplication, such that there is no conceptual difference between a cross-stitch block applied on the vector representations of fully-connected layers and the feature map representations of convolutional layers.

The Sluice Block

The Sluice block (Ruder et al., 2017) is an extension of the cross-stitch block. Like the cross-stitch block, it defines the sharing mechanism as trainable linear combinations of task-specific feature representations. However, unlike the cross-stitch block, the sluice block defines another set of linear combinations on layer-specific feature representations. It learns a linear combination between the feature representations computed at every layer of each task-specific network. The resulting value of the linear combination is then used directly as input to the last layer, typically a softmax layer for classification problems.

Assuming that $x_{t,l}$ is the feature representation of task t computed at layer l, the linear combination on the layer-specific feature representations $x_{t,l}$ can be defined as follows:

$$y_t = \sum_{l=1}^{L} \beta_{t,l} x_{t,l} \,, \tag{4.2.3}$$

where $\beta_{t,l}$ are the weights of the linear combination. Note than the product $\beta_{t,l}x_{t,l}$ is a simple scalar multiplication, such that there is no conceptual difference between a sluice block applied on vectors and a sluice block applied on feature maps.

The goal of the linear combination of Eq. (4.2.3) is to learn hierarchical relations between each task. It is based on the observation that the tasks have a different depth at which they interact (Hashimoto et al., 2017; Søgaard and Goldberg, 2016). This depth is difficult to establish *a priori* when designing the network and an improper choice can lead to bad interactions. The linear combinations address this limitation by learning it with gradient descent during training.

Cross-Residual Block

The Cross-Residual Block (Jou and Chang, 2016) is another extension of the cross-stitch block based on the recent advances in residual learning (He et al., 2016a, 2015) (ref. Section 3.4.7). They defined their cross-residual block following the standard residual block introduced in Section 3.4.7. The cross-residual block has the following structure:

$$y_t = \mathcal{F}_t(x_t) + \sum_{t=1}^{T} x_t \,, \tag{4.2.4}$$

where $\mathcal{F}_t$ is the residual mapping for task t and T is the total number of tasks.

The cross-residual block defined in Eq. (4.2.4) implements the sharing mechanism similarly to the cross-stitch block. Indeed, the task-specific feature representations x_t are

combined together using linear combinations. In the case of the cross-stitch block, the linear combinations are parameterized and are learned during training. In the case of the cross-residual block, the linear combinations are not parameterized and only amount to summing the features maps together.

4.2.4 A Limitation of the Reviewed Trainable Components

The sharing mechanisms defined as a trainable component presented in the previous section all shared the same limitation: they were limited to *linear* relations between the tasks. The cross-stitch block, the sluice block and the cross-residual block all defined their lateral connectivity as a linear combination of the feature maps. Linear relations are less expressive than nonlinear ones. As a result, the amount of information sharing that is allowed between the networks is also limited. In our contribution in Chapter 7, we address precisely this limitation by designing a sharing mechanism based on nonlinear relations. We further take into account the fact that depth influences the relevance of each task towards another by designing our nonlinear transformations based on residual learning.

4.3 Conclusion

In this chapter, we reviewed important Multitask Learning works in the field of deep learning. We focused our review on approaches in the family of soft parameter sharing that define a mechanism for leveraging task-specific domain information. We started with approaches based on regularization terms and explained that they require a fine adjustment of their influence in the training objective in order to work. We then elaborated on approaches that define the sharing mechanism as a trainable component of the network, to do away with this fine adjustment. We started with the cross-stitch block that defines information sharing as trainable linear combinations. We then followed with extensions of the cross-stitch block with the sluice block, which focuses on the hierarchical relations between each task, and the cross-residual block, which implements residual learning. We finally explained that linear relations are more limited than nonlinear relations, which restrict the amount of information sharing between the tasks.

We now present, in the following chapters, our contributions for representation learning. We start in Chapter 5 with our experimental evaluation of sparse dictionary learning on the problem of grasp localization. We also detail our proposed residual network for grasp

detection and compare it to sparse dictionary learning. We then present our work on a novel parametric activation function in Chapter 6. We show that our parameterization of the Exponential Linear Unit improves performance. We finally present our work on a novel soft parameter sharing mechanism for MTL in Chapter 7. We show how residual learning can help to leverage the domain information of related tasks.

Chapter 5

Representation Learning Exploration on the Problem of Grasping

Representation learning techniques aim to automatically learn to extract features from the data. Their goal is to replace manual feature engineering, which has been the favored approach before representation learning, with an approach that automatically discovers the underlying factors of variation using machine learning principles. It has since become a well-established field in machine learning. In this chapter, we explore representation learning on the specific problem of grasp localization. To this effect, we present a comparative study of sparse dictionary learning approaches on the problems of grasp recognition and grasp detection, and propose a convolutional neural networks for grasp detection. The analysis of our empirical evaluation illustrates the advantages and disadvantages of both frameworks.

5.1 Introduction

The ability to grasp objects is a fundamental skill for a robotic system. It has become an important issue over the past few years (Kehoe et al., 2015; Levine et al., 2018). Companies like Amazon show great interests in developing novel advances in the domain. Indeed, Amazon holds its *picking challenge* (Correll et al., 2016) every year in order to fuel the development of its *smart* warehouses where grasping is the central concern. We also see an increase in the development of domestic robots that need to interact with their complex surrounding environment (Bogue, 2017). They need to perform difficult actions such as opening doors, manipulating objects, using tools or moving things around. At the basis of these interactions lies the ability to grasp objects. It is thus

a fundamental skill to integrate into a robotic system in order to allow the system to accomplish its tasks.

Visual perception plays an important role in object grasping (Saxena et al., 2008). Its role in the grasping context is to localize affordances, or graspable regions, by analyzing the content of the scene. It is one of the first steps to do before elaborating a grasp manoeuver. The system interprets 2D (image) and 3D (depth) information, and generates a list of candidate regions. The regions are usually ordered by graspability, which is a measure of grasp success probability.

Detecting graspable regions involves elaborated and complex visual tasks. For instance, the system needs to perform image segmentation, visual disambiguation and object detection in order to output valuable candidate regions. These tasks usually have a high complexity, which makes learning an efficient perception system difficult. In our case, we will focus strictly on the case where the scene is uncluttered and contains a single object.

Several previous works used 3D simulations to localize graspable regions (Bicchi and Kumar, 2000; Goldfeder et al., 2007; Miller and Allen, 2004; Detry et al., 2013; Pelossof et al., 2004). A strong limitation of these approaches is the need to obtain 3D physical models of the objects beforehand. This requirement reduces their applicability for general grasp localization because the objects to be grasped are rarely known *a priori*. We will instead focus our efforts into approaches that do not require building complex physical models prior to the execution.

The advent of the inexpensive Microsoft Kinect 3D camera has opened up the door for novel approaches to identify grasp candidates. Several works have used it to show the importance of depth information for representing images (Lai et al., 2011; Blum et al., 2012; Maitin-Shepard et al., 2010; Saxena et al., 2008). Its market accessibility and ease-of-use has provided a straightforward solution for incorporating depth and light information – known as RGBD images – into various industrial settings.

The main challenge when using Kinect RGBD images is the presence of two types of noise in the depth information. One is the axial and lateral noise model of the object distance to the camera, and the other is the mask noise model of missing 3D information. The latter can be particularly limitative when computing the depth information of shiny surfaces. Shiny surfaces often cause structured-light 3D cameras to fail and hence, return no depth values for certain pixels.

Training an efficient perception system therefore requires taking into account these additional challenges. Indeed, the system must not only consider the characteristics of the environment but also the Microsoft Kinect axial, lateral and mask sources of noise. This comes down to extracting the information related to the content of the image, while discarding the unimportant sources of noise. As shown in previous works (Socher et al., 2012; Gupta et al., 2014; Lenz et al., 2015; Redmon and Angelova, 2015; Rusu et al., 2010), the main way with which we can achieve this goal is by transforming the RGBD input image into a high-level feature representation.

In this chapter, we thus explore representation learning on the problem of grasp localization. We focus our efforts on two well-established representation learning frameworks: sparse dictionary learning and convolutional neural networks. We perform a comparative study of several previously proposed sparse dictionary learning approaches on the problem of grasp recognition and grasp detection, as well as propose a convolutional neural network for grasp detection.

5.2 Model

In this section, we introduce several models for grasp localization. We start by describing the grasp rectangle, which is a low-dimensional representation of a two-plates parallel gripper. We then elaborate on the sparse dictionary learning approaches that we selected. Finally, we detail the convolutional neural network architecture that we modified to meet our needs for grasp localization.

5.2.1 Grasp Rectangle Representation

A fundamental concept in grasping is its representation, i.e. how to parameterize a grasping point. It has undergone significant evolution over the past decades (Saxena et al., 2006; Le et al., 2010). The 5-dimensional *grasp rectangle* representation is one of the most useful representation for encapsulating the 7-dimensional two-plates parallel gripper configuration (Jiang et al., 2011). The grasp rectangle is defined as a 2D oriented rectangle that indicates the gripper's location, orientation and physical limitations. As shown in Figure 5.1, it encapsulates the gripper with the center coordinates (x, y), the width, the height and the orientation with respect to the image reference:

$$R = \{x, y, \theta, w, h\} .$$

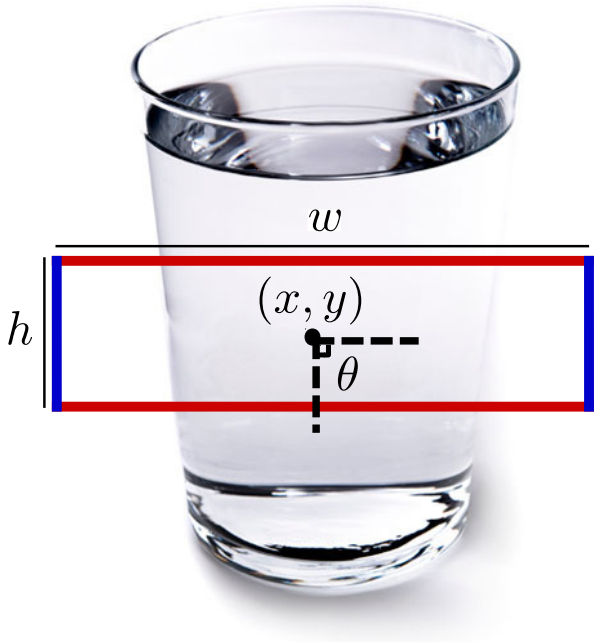


Figure 5.1: The grasp rectangle is a five-dimensional grasp representation. The 2D rectangle is fully determined by its center coordinates (x, y), width, height and its angle θ from the x-axis. The blue edges indicate the gripper plate location and the red edges show the gripper opening, prior to grasping.

The main advantage of the grasp rectangle is that it makes grasp recognition analogous to object recognition, and grasp detection analogous to object detection. Indeed, the goal of grasp recognition is to determine whether a grasp rectangle R is a good candidate or a bad one, which can be seen as a binary object recognition problem. Similarly, the goal of object detection is to predict the configuration of the best rectangle R^*, which can be seen as a standard object detection problem with a bounding box. As a result, this opens up the opportunity to leverage the large body of works on object recognition and detection from the computer vision literature. For this reason, we will use the grasp rectangle representation in all our experiments.

5.2.2 Sparse Dictionary Learning

Sparse Dictionary Learning (SDL) poses that an observation can be expressed as a sparse linear combination of few atoms from a dictionary. Recall from Section 2.1 that a standard SDL approach is divided into a *dictionary learning* phase and a *feature*

Table 5.1: All combinations of dictionary learning and feature coding approaches.

Feature Coding	
Least Angle Regression	(LARS)
Masked LARS	(mLARS)
Orthogonal Matching Pursuit	(OMP)
Masked OMP	(mOMP)
Soft-Thresholding	(ST)
Natural	(N)
Dictionary Learning	
Online Dictionary Learning with ℓ_0	(ODL-0)
Online Dictionary Learning with ℓ_1	(ODL-1)
Gain Shape Vector Quantization	(GSVQ)
Normalized K-Means	(NKM)
Random Patches	(RP)
Random	(R)

coding phase. For feature coding, the problem is formulated as:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\Omega(w^{(i)})\,, \tag{5.2.1}$$

where D is the dictionary, $x^{(i)}$ are the input observations, $w^{(i)}$ are the feature representations of the corresponding inputs, Ω is the sparsity measure and λ is the penalty weight. For dictionary learning, the problem is formulated as:

$$D = \underset{D}{\operatorname{argmin}} \min_{W} \sum_{i=1}^{N} \|x^{(i)} - Dw^{(i)}\|_2^2 + \lambda\Omega(w^{(i)}) \tag{5.2.2}$$
$$\text{subject to } \|D^{(j)}\|_2 = 1,\ \forall j \in \{1 \dots d\}\,,$$

where N is the number of observations and d is the dimensionality of the inputs $x^{(i)}$.

For our comparative study, we selected six feature coding approaches and six dictionary learning approaches, and compared each pairwise combination together. We selected approaches following a previous sparse dictionary learning experimental evaluation on object recognition (Coates and Ng, 2011) An overview of the selected approaches is shown in Table 5.1. We now briefly review each of them, and refer the reader to Section 2.2 and 2.3 for more details.

Dictionary Learning

The first approach that we selected in our comparative framework is Online Dictionary Learning (ODL) (Mairal et al., 2009). It is one of the most common approach for dictionary learning. It is a block-coordinate descent approach based on an iterative-alternative scheme to solve the nested optimization problem. ODL can be used with both the ℓ_0 and the ℓ_1 sparsity measures. In the case where $\Omega = \ell_0$, we use Orthogonal Matching Pursuit for feature coding, while with $\Omega = \ell_1$, we use Least Angle Regression. We refer to ODL with a ℓ_0 sparsity measure as ODL-0, and ODL with a ℓ_1 sparsity measure as ODL-1.

As second approach, we opted for Gain Shape Vector Quantization (GSVQ) (Coates and Ng, 2012). GSVQ solves the least-square optimization problem in an iterative fashion, using damped updates. It also defines feature coding as Orthogonal Matching Pursuit with a sparsity value of $\gamma = 1$. In other words, the weight solution $w^{(i)}$ is defined as a vector of all zeros except at position j, where j is the atom $D^{(j)}$ that is the most correlated with observation $x^{(i)}$.

We also tried learning a dictionary D without specifying a sparsity measure. One way is to run K-Means clustering on ZCA-whitened data and use the normalized centroids as dictionary atoms (Coates et al., 2011). As seen in Section 2.1, ZCA-whitening is a powerful tool to remove correlation while ensuring that the decorrelated observations are as close as possible to the non-decorrelated observations. By combining whitening with K-Means ability to find the relevant clusters of the data distribution, we can learn representative centroids that capture nonlinear dependencies. We refer to this method as Normalized K-Means (NKM) in our experiments.

As the final approach, we included two methods based on random dictionaries. The first one is to randomly sample d whitened observations and use their normalized version as dictionary atoms. This can be seen as a one-shot approximation of K-Means clustering. We refer to this approach as Random Patches (RP). The other one is to sample d times the uniform distribution $U([-1, 1]^n)$ and use the normalized vectors as dictionary atoms. Random dictionaries have been advertised in past works as displaying good performances even though no training is done (Saxe et al., 2011; Jarrett et al., 2009). However, we show in our experiments that they always give the worst performance. We refer to this method as Random (R).

Feature Coding

The first approach that we selected is Least Angle Regression (LARS) (Efron et al., 2004). As seen Section 2.2, LARS solves the penalized least-square problem of Eq. (5.2.1) under the $\Omega = \ell_1$ sparsity measure. It is a greedy inference approach that iteratively updates the weight solution $w^{(i)}$ in the direction that preserves the angles between each active atoms and the residue.

The second approach that we selected is Orthogonal Matching Pursuit (OMP) (Pati et al., 1993). As seen Section 2.2, OMP solves the penalized least-square problem of Eq. (5.2.1) under the $\Omega = \ell_0$ sparsity measure. Similarly to LARS, OMP is a greedy inference approach that iteratively updates the weight solution $w^{(i)}$ by activating one atom at a time. The approach guarantees that $\|w^{(i)}\|_0 = \gamma$ after γ iterations because the weight solution $w^{(i)}$ is updated such that the residue becomes orthogonal to the active atoms.

We also considered extending Eq. (5.2.1) to take into account a mask vector m representing masked-out entries in observation $x^{(i)}$. The purpose of mask m is to remove the contribution of certain entries during the computation of the residual. The reason we would consider using m is to integrate mask noise models during feature coding. This has been suggested in previous work to improve the sparsity of the weight solution (Mairal et al., 2009). The feature coding problem of Eq. (5.2.1) can be modified to include m as follows:

$$w^{(i)} = \underset{w^{(i)}}{\operatorname{argmin}} \|\operatorname{diag}(m)\left(x^{(i)} - Dw^{(i)}\right)\|_2^2 + \lambda\Omega(w^{(i)})\,. \tag{5.2.3}$$

In this case, both LARS and OMP can be extended to take into account mask m (Mairal et al., 2014). We refer to these approaches as mLARS and mOMP.

The next approach that we included is Marginal Regression (Donoho and Johnstone, 1995). In contrast to LARS and OMP, marginal regression outputs a candidate solution in a single iteration. It tries to find the sparsest weight vector $w^{(i)}$ that approximates the vector $D^\top x^{(i)}$ of correlation between observation $x^{(i)}$ and each atom $D_{:,j}$. The weight solution $w^{(i)}$ is straightforward to compute by simply applying the soft-thresholding (ST) function parameterized by λ on vector $D^\top x^{(i)}$. We refer to this approach as ST.

We also included Natural (N) feature coding approaches. Recall that the penalty weight λ for feature coding in Eq. (5.2.1) is different from the penalty weight for dictionary learning in Eq. (5.2.2). This means that we need to perform a two-dimensional grid search to define the penalty weight of feature coding and the penalty weight of dictio-

nary learning. The idea of the Natural feature coding is to reduce the computational burden of the two-dimensional grid search by using a single penalty weight. The penalty weight for feature coding becomes the same as the one used during dictionary learning. This can be seen as bounding the feature coding approach to the dictionary learning approach. Thus, for ODL-1, we used as natural feature coding approach LARS with the same sparsity parameter λ that was used during dictionary learning. Similarly for ODL-0, we used as natural feature coding approach OMP with the same sparsity parameter γ that was used during dictionary learning. For GSVQ, we used OMP with a fixed $\gamma = 1$. For R and RP, we used ST with $\tau = 0$, which corresponds to a random linear projection.

Finally, for NKM we chose as natural feature coding approach K-Means Triangular (KMT) Coates et al. (2011). KMT is a soft alternative to the usual *one-of-K* encoding that associates each observation to the closest centroid. It can be formulated as follows:

$$w_j^{(i)} = \max\{0, \mu(z^{(i)}) - z_j^{(i)}\}, \tag{5.2.4}$$

where $z_j^{(i)} = \|x^{(i)} - D^{(j)}\|_2$ is the euclidean distance between observation $x^{(i)}$ and centroid $D^{(j)}$, and $\mu(z^{(i)})$ is the mean of the elements of $z^{(i)}$. We also make an additional distinction when using KMT is that we do not normalize the K-Means centroids. In other words, we do not project the centroids on the unit hyper-sphere after training is done.

After computing the weight solution $w^{(i)}$, we apply Polarity Splitting (PS) (Coates and Ng, 2011). PS constructs a vector $z^{(i)}$ from $w^{(i)}$ by splitting the positive weights from the negative ones. This can be formulated as follows:

$$z_j^{(i)} = \max\{w_j^{(i)}, 0\} \qquad\qquad f_{j+d}^{(i)} = \max\{-w_j^{(i)}, 0\}$$

This technique improves the representative ability of linear classifiers as it forces them to model positive and negative weights differently. Starting from this point, we are now referencing to feature vector $z^{(i)}$ when talking about the output vector of any feature coding approach.

Feature Extraction Pipeline

The grasp rectangle introduces another key notion that the selected sparse dictionary learning approaches rely one, which is the *grasp rectangle image*. It is defined as the underlying image captured by the grasp rectangle. Extracting relevant features from the grasp rectangle image is essential for grasp localization. It is not sufficient to simply

perform feature coding once on the entire input image. We have to rely on a more complex feature extraction process known as *spatial pyramid matching* (Yang et al., 2009).

Our spatial pyramid matching approach is depicted in Figure 5.2. From the RGBD image, we first compute the gray channel (K) and the depth normal coordinates N_x, N_y and N_z. The depth normal coordinates correspond to the coordinates of the normal vector to the surface. In other words, considering that the depth image is the function $z(x, y)$, the normal vector is the direction $(-\frac{\partial z(x,y)}{\partial x}, -\frac{\partial z(x,y)}{\partial y}, 1)$, normalized to unit norm. Thus, we concatenate the four new channels K, N_x, N_y and N_z to each image, which results in a total of eight channels. We then rotate the grasp rectangle image to match the orientation of the global image and rescale it to $24 \times 24 \times 8$ with aspect-ratio preserved. Preserving aspect-ratio is important to avoid distortion. An example of distortion that can be avoided by preserving aspect-ratio is shown in Figure 5.3.

Then, from the $24 \times 24 \times 8$ grasp rectangle image, we further extract all $6 \times 6 \times 8$ sub-images in a sliding window manner. We convert these sub-images into a vector format by applying matrix vectorization, which results in a 288-dimensional vector that is used as observation $x^{(i)}$. We perform channel-wise standardization and ZCA whitening, then extract feature vectors $z^{(i)}$ using a combination of dictionary learning and feature coding approach. Afterwards, we divide the image into four quadrants and perform sum pooling over the feature vectors $z^{(i)}$ of each quadrant respectively. Finally, the pooled feature vectors from all quadrants are concatenated into a single vector that is used as input to the classifier.

5.2.3 Convolutional Neural Networks

As seen in Chapter 3, Convolutional Neural Networks (CNNs) have become the leading deep learning approach for visual recognition (Dong et al., 2016; Ren et al., 2015). They only recently became popular even though they were originally proposed in the 80s due to being difficult to train. Their hierarchical structure makes them vulnerable to many problems, which become more pervasive as the CNN gets deeper. Recent works in Residual Network (ResNet) (He et al., 2016b,a) made seminal contributions that significantly reduced the difficulty of training a CNN. The central idea in ResNet is to replace the standard feed-forward mappings with additive residual mappings as the main building blocks of the network, as done in Section 3.4.7. This increases the information flow during both the forward and backward passes, which facilitates train-

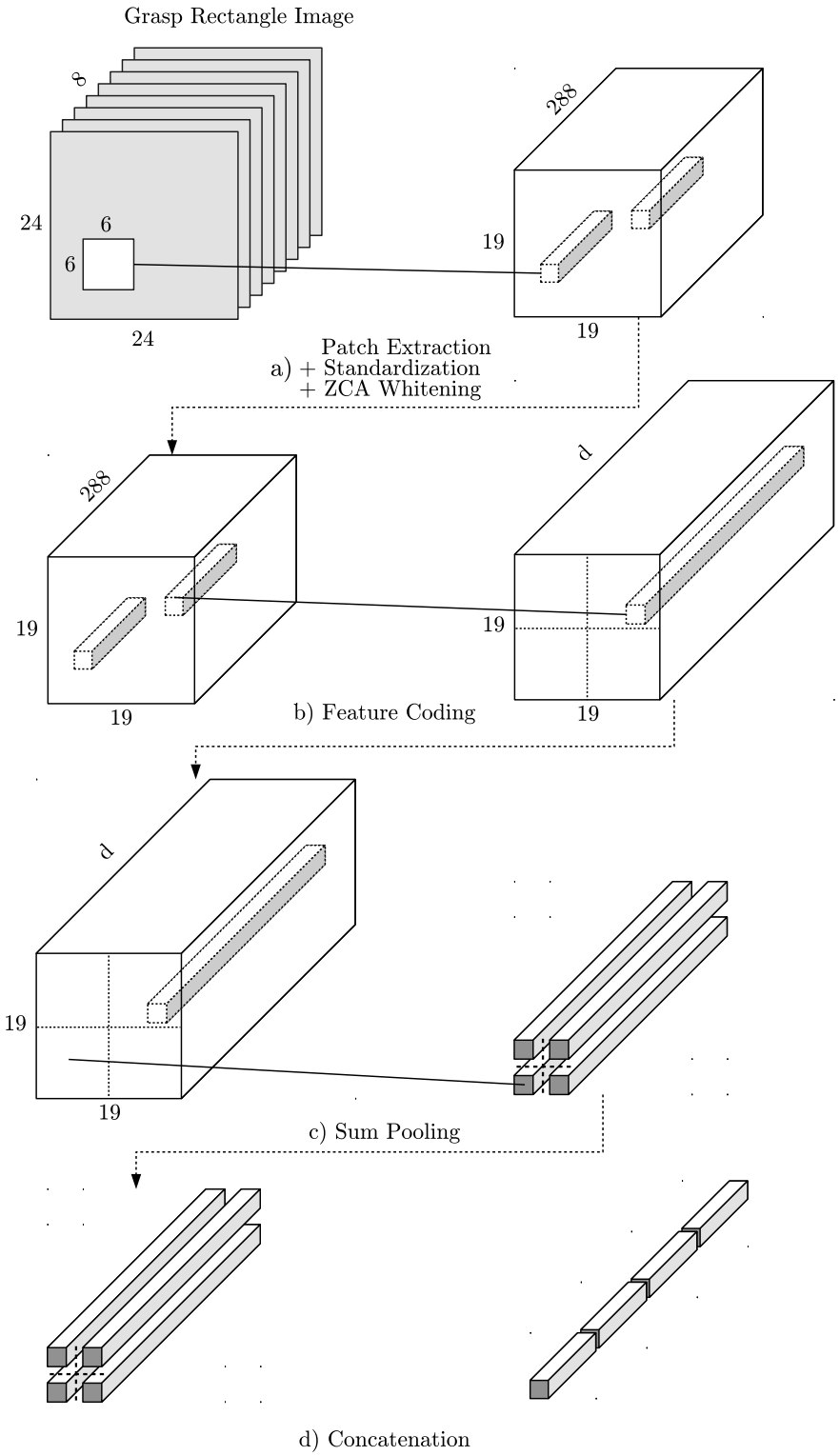


Figure 5.2: Feature extraction process for our sparse dictionary learning approaches. a) all $6 \times 6 \times 8$ patches $x^{(i)}$ are extracted in a sliding window manner from the grasp rectangle image. b) each patch $x^{(i)}$ is mapped to its feature representation $z^{(i)}$ given a dictionary D and a feature coding method. c) a four quadrants sum pooling is applied on the feature vectors $z^{(i)}$. d) all four quadrant pooled weights are concatenated into a single vector. The final vector on the right is used as input to the SVM.

a) *b)* *c)* *d)*

Figure 5.3: a) a pair of scissors with a candidate grasp rectangle image. b) image taken from the rectangle rotated to match the global image orientation. c) rescaled image with preserved aspect ratio. The black regions indicate masked-out padding. d) rescaled image without preserved aspect ratio. Preserving aspect ratio reduces distortion.

ing. For these reasons, we opted for residual blocks in our CNN. We now elaborate in more details on our CNN architecture.

Network Architecture

Contrary to sparse dictionary learning, we designed our CNN to take as input the original RGBD image instead of the grasp rectangle image. The CNN directly outputs candidate grasp rectangles from the original RGBD image, instead of labelling the grasp rectangle as positive or negative, i.e. it performs detection rather than classification. To take into account the fact that each image has a variable number of graspable regions, we divided the CNN predictions into a 7×7 grid (Redmon and Angelova, 2015). For each cell, the CNN predicts both a grasp rectangle and a confidence score, which amounts to predicting 49 grasp rectangles and 49 confidence scores. The cell with the highest confidence score is selected as the candidate grasp rectangle for the image.

An illustration of predictions by the CNN is shown in Figure 5.4. The figure on the left shows the grasp rectangles predictions, while the figure on the right show the confidence map containing the graspability scores for each rectangle. The final prediction is the grasp rectangle with the highest score.

We further take into account the rotational invariance of the gripper angle when predicting the grasp rectangle. Due to the symmetry of the rectangle, grasping at θ or $\theta + 180^\circ$ results in the same gripper configuration. Managing this invariance is done by using the sin and cos of twice the angle θ when computing the loss between the ground truth rectangle and the candidate rectangle (Redmon and Angelova, 2015):

$$R' = \{x, y, w, h, \sin(2\theta), \cos(2\theta)\}\,. \tag{5.2.5}$$

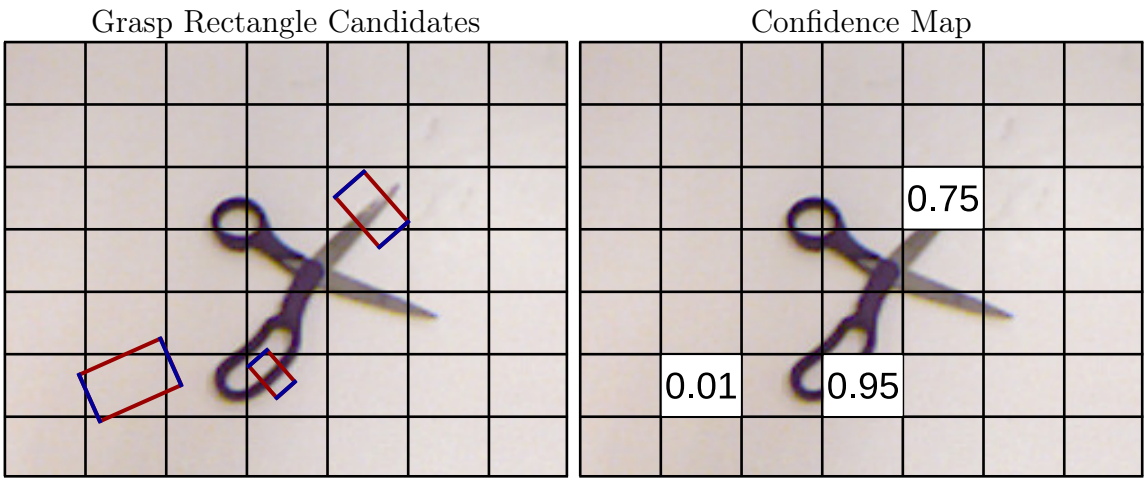


Figure 5.4: Illustration of grasp rectangle predictions by our CNN. On the left figure, we show three instances of predicted rectangles, and on the right, the corresponding graspability score. The CNN is designed to output a grasp rectangle and a graspability score for each cell of the 7×7 grid (49 in total). The rectangle in the cell with the highest score is selected as the candidate grasp rectangle during evaluation.

As a result, the dimensionality of the overall output is $7 \cdot 7 \cdot 6 = 294$ for the grasp rectangles and $7 \cdot 7 \cdot 1 = 49$ for the confidence score.

We designed our network architecture based on the recent advances in residual learning (He et al., 2016b). Specifically, we opted for residual blocks with *full pre-activation* mappings (He et al., 2016b). These blocks are defined as follows:

$$\mathcal{F}(h) = h + \mathcal{H}(h) \tag{5.2.6}$$

$$= h + \text{Conv}_{3\times3}(\text{ReLU}(\text{BN}(\text{Conv}_{3\times3}(\text{ReLU}(\text{BN}(h)))))) \,, \tag{5.2.7}$$

where BN corresponds to batch normalization (Ioffe and Szegedy, 2015), ReLU to the rectified linear unit (Krizhevsky et al., 2012), $\text{Conv}_{w\times h}$ to a convolutional layer with filter size of $w \times h$ and $\circ$ is the standard function composition.

Standard modern CNNs for classifying objects in RGB images uses a Global Average Pooling (GAP) layer at the end of the convolutional stage, i.e. prior to the fully-connected layers. In object recognition, GAP helps to create correspondence between the feature maps and the object categories (ref. Section 3.4.2). However, during our preliminary experiments, we observed that GAP was harmful for grasp detection. The CNN was unable to assign high confidence score to the cells representing regions of high graspability. In fact, the CNN assigned the same confidence score to every cell. We observed that this was caused by the averaging nature of GAP that caused complete spatial information loss in the error signal. The network was thus unable to spatially

correlate its mistakes on the confidence scores. As a result, the CNN was unable to localize the grasp rectangles.

To solve this problem, we substituted GAP before predicting the confidence scores by a standard flattening layer. We kept GAP before predicting the grasp rectangles, since it helps to more easily create a correspondence between the feature maps and the rectangle categories, as it does for object categories in object recognition.

An illustration of our residual network is shown in Figure 5.5. The left side presents the overall architecture of the network, while the right side presents the *full pre-activation* residual block. The full image with all 7 channels is given as input to the network (at the top). The image is then forwarded through the 6 stages of the network, which are composed of a series of ResBlocks. At the bottom, the network outputs two quantities. On the left, it outputs the grasp rectangles after applying global average pooling. On the right, it outputs the confidence map after applying a sigmoid activation function.

5.3 Experimental Framework

In this section, we present our experimental framework. We start with a description of the Cornell task (Jiang et al., 2011), which was the standard dataset for grasp recognition and detection at the time when we performed the experiments. We then elaborate on the experimental framework related to sparse dictionary learning and finally on the experimental framework related to convolutional neural network.

5.3.1 Cornell Dataset

The Cornell Grasping Dataset (Jiang et al., 2011) contains 885 RGBD images of 240 distinct objects [1]. The images are taken using a Kinect and have a dimensionality of $640 \times 480 \times 4$. As shown in Figure 5.6, each image contains a certain number of grasp rectangles labelled as either positive or negative. Positive grasp rectangles identify object regions with a high graspability score, while negative grasp rectangles identify object regions with a low graspability score. The grasp rectangles vary in terms of size, orientation and position, but do not cover every scenario. Indeed, the majority of high and low graspability regions are not labelled.

The Cornell dataset defines two learning tasks. The first one is grasp recognition and the second one is grasp detection. For grasp recognition, the task is to determine whether a

[1] The dataset can be downloaded at `http://pr.cs.cornell.edu/deepgrasping/`

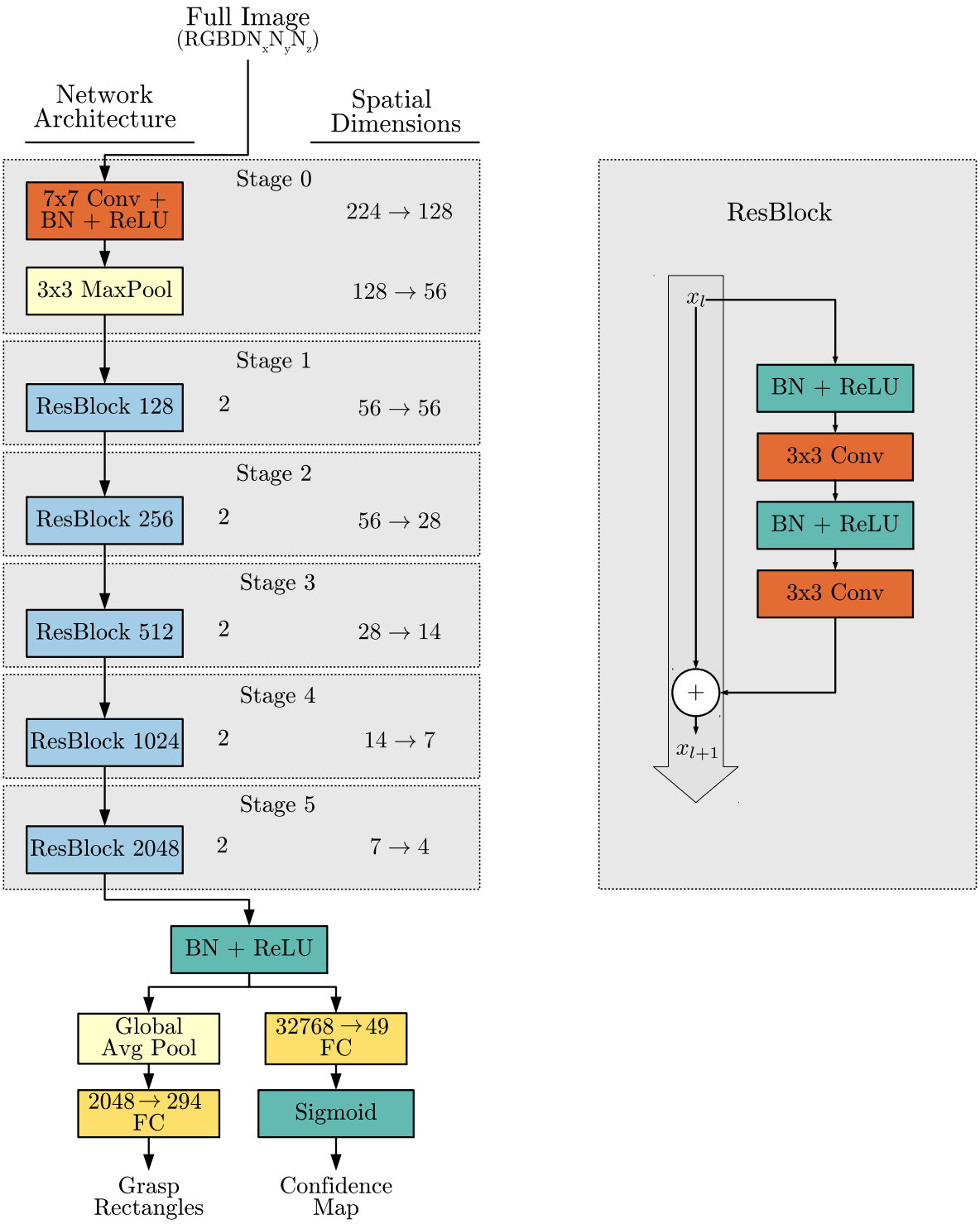


Figure 5.5: An illustration of our residual network (left) and the residual block (right) from which it is composed. A block has a residual mapping f_i and an identity skip connection. The output x_i of block i is processed twice by a series of batch normalization, ReLU and convolution layers, then is added to x_i. This gives the output x_{i+1}, which in forwarded to the next block. Our network does not use the standard global average pooling for predicting the confidence map, only for the rectangle configurations.

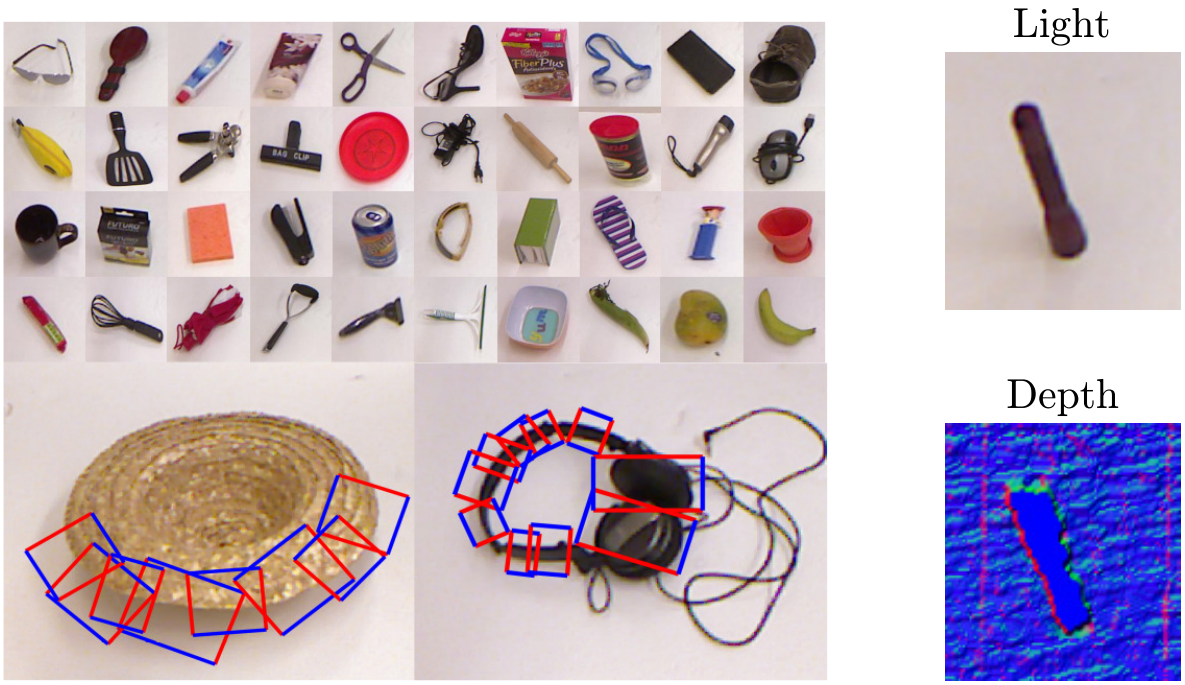


Figure 5.6: The Cornell grasping dataset contains images of a wide variety of everyday objects. This database defines two tasks: grasp recognition and grasp detection. For grasp recognition, the goal is to determine whether a grasp rectangle R is a good or a bad candidate. For grasp detection, the goal is to predict the configuration of the best grasp rectangle R^*.

grasp rectangle is positive or negative. For grasp detection, the task is to generate grasp rectangles at regions of high graspability. We evaluate our sparse dictionary learning approaches on both grasp recognition and grasp detection, but we only evaluate our convolutional neural network on the problem of grasp detection. Our network is already designed to perform grasp detection, so there is no need to evaluate it on grasp recognition.

5.3.2 Sparse Dictionary Learning

As we have mentioned in Section 5.2.2, it is not sufficient to simply perform feature coding once on the entire input image. We have to rely on a more complex feature extraction process known as spatial pyramid matching. We illustrated the approach in Figure 5.2. We first extract randomly 100,000 small $6 \times 6 \times 8$ sub-images from all $24 \times 24 \times 8$ grasp rectangle images of the training set. Then, we apply channel-wise standardization (subtract the mean and divide by the standard deviation) followed by ZCA whitening (Hyvärinen et al., 2004). The selected sub-images are vectorized into

288-dimensional vectors and used as observations $x^{(i)}$. We finally select a dictionary learning approach and train the dictionary.

After training is done, we use the trained dictionary D to perform grasp recognition and grasp detection. We now elaborate on the method that we used to perform the two.

Grasp Recognition Classifier

For grasp recognition, we opted for a ℓ_2-linear Support Vector Machine (SVM) classifier (Cortes and Vapnik, 1995). We optimized the SVM using a standard L-BFGS solver from Schmidt's *minFunc* toolbox (Schmidt, 2005). We cross-validated the regularization parameter C with $C \in \{1, 10, 100, 1000\}$.

Grasp Detection Regressor

For grasp detection, we opted for a standard sliding window search in grasp rectangle space. We used a horizontal and vertical stride of 10 pixels to slide the window. We also varied the size of the rectangle from 10 pixels to 90 pixels with a stride of 10 pixels, and varied the orientation of the rectangle from 0 degree to 180 degrees with a stride of 15 degrees. For each of these grasp rectangle images, we performed feature coding and used the SVM to compute the graspability score of the grasp rectangle. For feature coding, we did not perform grid search for the hyper-parameters, but rather used the ones that were selected by the grid search during our grasp recognition experiment. The rectangle having the highest score was chosen as the candidate grasp rectangle.

5.3.3 Convolutional Neural Network

Despite the recent advances that improved the ease with which CNNs learn, training a deep network still remains a challenge. Representing complex observations is difficult, particularly when the amount of data is small. In this case, the CNN can easily overfit on the training data if no proper care is taken during training. When overfitting occurs, the learned feature representation is usually not able to generalize well to the complete and unobserved data distribution. Avoiding overfitting then becomes a central concern. We address this issue partially by using *data augmentation* techniques, which we now discuss.

Managing Overfitting with Data Augmentation

Data augmentation has proven many times that it can prevent CNNs from overfitting. Its central idea is to artificially increase the amount of training images by generating new images from existing ones. To do this, data augmentation defines a data generation process that uses a series of class-preserving random transformations. A transformation is class-preserving when the generated image represents the same category as the original image. For instance, a small image rotation of a digit is class-preserving, but a 180° rotation is not (a 6 becomes a 9).

Data augmentation can be done either off-line or on-line. In the off-line case, data augmentation is performed before training. A fixed number of images is generated from the original observations, which are then added to the training dataset. The network is trained as usual, but this time using this enlarged dataset. In the on-line case, data augmentation is done on-the-fly during training. Observations are first sampled from the original dataset, then passed through the data generation process. The newly generated observations are sent to the network, but are discarded once the gradient has been computed. These steps are repeated every iteration, which makes the network see new images at each iteration.

The main difference between off-line and on-line data augmentation is the number of new images at each epoch. As we mentioned above, off-line data augmentation is equivalent to standard training on a larger dataset. In other words, the network sees all images during one epoch, and all images are re-used over and over again at each epoch. In on-line data augmentation, the network sees new images at every iteration. No image is used more than once, although some can be similar. The main advantage of on-line data augmentation is that we do not have to store a very large dataset. This issue can become a major problem when using many augmentation strategies. For this reason, we opted for on-line data augmentation in our experiments.

The quality of the generated images depends on the choice of transformations. For instance, a data generation process that uses too heavy transformations will generate too different images. On the contrary, too weak transformations will generate too similar images. Selecting the proper transformations can make a difference in the added value of data augmentation. We now elaborate on the transformations that we used in our experiments. The transformations are presented according to their order in the preprocessing:

1. Standardization: From the original RGBD image, we estimate the depth normal N_x, N_y, N_z. We then subtract the mean and divide by the standard deviation (computed from the training set) of each 7 channels.

2. Rotation: The image is rotated by a random angle uniformly sampled in the interval $[-15°, 15°]$.

3. Random center crop: A 320×320 crop is taken from the center of the original 640×480 image, translated by up to ± 50 pixels in both the horizontal and vertical directions.

4. Square scale: The 320×320 crop is resized to 224×224 using standard bilinear interpolation.

5. Color jitter: We randomly change the brightness, contrast and saturation with a blend uniformly sampled in the interval $[0.6, 1.4]$.

6. Lighting: We perform Krizhevsky et al. (2012)'s Fancy PCA color augmentation on all RGB-D-$N_x N_y N_z$ channels.

Note that we also apply the rotation, translation, cropping and scaling transformations on the grasp rectangles in order to preserve the correct graspable regions.

Training Details

Training our network in Figure 5.5 for grasp detection requires special considerations (Redmon and Angelova, 2015). We have to consider that each image has a variable number of positive grasp rectangles. The spatial division of the predictions causes certain cells to have zero, one or more grasp rectangles during training. In order to take into account this particularity, we employed the following training scheme. At each step, we selected uniformly up to five ground truth positive rectangles in five different cells. We then constructed the corresponding confidence map by placing a score of 1 at the cells associated with the selected rectangles and by placing a score of 0 at the other cells. We then computed the loss between each predicted and ground truth scores, but only computed the loss between the predicted grasp rectangles associated with the selected five ground truth grasp rectangles. This way, the network can learn to localize the regions with high graspability as well as generate grasp rectangles conditioned on the content of each cell.

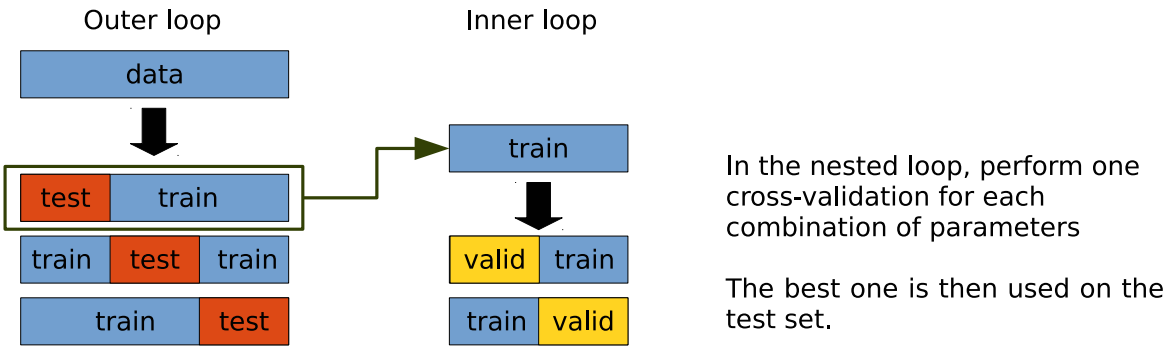


Figure 5.7: In nested cross-validation, the grid-search in parameter space in performed in the inner loop. For a given train-test split in the outer loop, we perform one standard cross-validation over new splits on the train set for each combination of parameters. The best parameters are then kept for the training and test sets of the outer loop. The process continues until all splits in the outer loop have been processed. This particular example shows a 3-2 nested cross-validation because its uses three train-test and two train-valid splits.

5.3.4 Grasp Recognition Evaluation

The Cornell dataset does not define a training set and a test set, so we employ cross-validation to report our results. More precisely, we use five-folds nested cross-validation. An illustration of the method is presented in Figure 5.7. The difference between standard and nested cross-validation is that nested cross-validation has a second layer of cross-validation. The purpose of the second layer is to perform hyper-parameter search. As shown on the left of Figure 5.7, nested cross-validation starts by dividing the data into five folds, as in standard cross-validation. We select four of them as the training set and the fifth one as the test set. This is called the outer loop. Then, nested cross-validation consider the training set of the outer loop as a new dataset and apply a second five-fold cross-validation on it. This step is called the inner loop and is shown on the right of Figure 5.7. The goal of the inner loop is to perform five-fold cross-validation for each hyper-parameter configuration. The one that gives the highest performance inside the inner loop is kept and used to train the approach in the outer loop. These steps repeat for all folds and the final performance is given by the average over all folds of the outer loop.

5.3.5 Grasp Detection Evaluation

To evaluate the quality of a candidate grasp rectangle, we used the *rectangle metric* (Jiang et al., 2011; Lenz et al., 2015). Specifically, if the rectangle metric between

the candidate grasp rectangle and any of the ground truth grasp rectangles is positive, then grasp detection is considered as a success. In more detail, the metric is positive if:

- The candidate orientation is within 30° of the ground truth rectangle.
- The Jaccard index between the candidate and the ground truth is greater than 25%, where the Jaccard index between two rectangles R_1 and R_2 is defined as:

$$J(R_1, R_2) = \frac{\text{area}(R_1 \cap R_2)}{\text{area}(R_1 \cup R_2)}. \tag{5.3.1}$$

The rectangle metric is more discriminative than a standard Jaccard index evaluation because it also considers the rectangle orientation.

In contrast to object recognition, we only perform a standard five-folds cross-validation. The reason we did not opt for nested cross-validation is because of the computational complexity of the problem. Training a convolutional neural network to perform object detection is time-consuming, as well as performing a grid-search over the space of grasp rectangles with sparse dictionary learning.

In addition, we employed two commonly used learning scenarios for grasp detection. The first one is *image-wise splitting* and the other one is *object-wise splitting*. In image-wise splitting, we split the images randomly during the creation of the cross-validation folds. In object-wise splitting, we split the objects randomly, while gathering all images of each object in the same fold. Image-wise splitting studies the ability to generalize to new positions and orientations of an object that has already been seen. This scenario is representative of a typical industrial context, because the set of objects is known beforehand. Object-wise splitting examine the capability to generalize to novel, unseen objects. This scenario is more realistic in the sense of general purpose robotics, since training on all possible objects is usually impossible. Using both image-wise and object-wise splitting, we can more accurately ascertain the performance of the approaches.

5.4 Experimental Results

In this section, we present the performance of the sparse dictionary learning approaches and our convolutional neural network. We report their grasp recognition and detection accuracy on the Cornell dataset.

Table 5.2: Cross-validation results of all combinations of dictionary learning and feature coding, for Cornell dataset. Numbers are grasp recognition accuracies in percent (%).

Dictionary	**Features**					
	LARS	mLARS	OMP	mOMP	ST	N
ODL-0	96.68	96.71	96.60	96.69	96.50	96.58
ODL-1	96.63	96.74	96.61	96.61	96.73	96.65
GSVQ	96.72	96.50	96.66	96.70	96.50	95.65
NKM	**96.86**	96.64	96.58	96.64	96.42	96.52
RP	96.43	96.51	96.35	96.20	96.28	96.37
R	95.84	95.52	95.34	95.15	95.78	95.90

5.4.1 Grasp Recognition Results

The nested cross-validation accuracies (in %) for grasp recognition using a dictionary of $d = 300$ atoms are reported in Table 5.2. The best dictionary learning + feature coding combination is NKM-LARS with an accuracy of 96.86%. The worst combination is R-mOMP with an accuracy of 95.15%. As a comparison, previous approaches achieved 89.6% with a hand-designed score function (Jiang et al., 2011), and 93.7% with a cascade of multi-layer perceptrons (Lenz et al., 2015).

Explicitly modelling the mask noise in the observations may not be as important as it seems. As we see in Table 5.2, mLARS and mOMP are not significantly better alternatives then their standard version LARS and OMP. For instance, modelling the mask noise with LARS improved the performance of ODL-0 from 96.68% to 96.71%, but reduced the accuracy of GSVQ from 96.72% to 96.50%. Similarly, modelling the mask noise for OMP did not change the performance of ODL-1, as both alternatives got 96.61%. These results provide additional experimental evidence to previous work that observed that standard feature coding approaches are already well-adapted to deal with mask noise (Bo et al., 2014).

Interestingly, the accuracies vary by no more than 1%. All combinations have an accuracy around 96.5%, apart from random dictionaries (R) which have an accuracy around 95.5%. This makes selecting the combinations to keep for grasp detection more challenging. We have to take into account accuracy as well as other characteristics. One is the number of hyper-parameters. For instance, GSVQ, NKM and RP could be considered for dictionary learning instead of ODL-0 and ODL-1 because they are hyper-parameter free. Similarly, the natural feature coding of NKM, GSVQ and RP may also be considered for feature coding instead of the others because they are hyper-parameter

free.

Another characteristic is computational complexity. For instance, LARS and mLARS were the most time-consuming, while ST and the natural feature coding of NKM and RP were fast. In addition, LARS and mLARS obtained a similar accuracy than the others. This makes ST, natural NKM and natural RP more appealing for real-time scenarios.

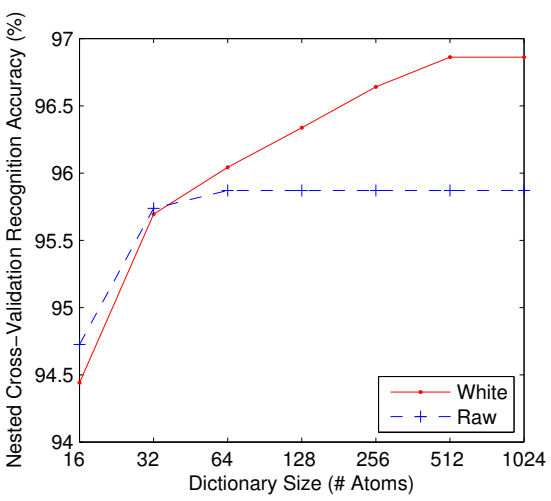
Given these considerations, we selected fives combinations for grasp detection. We chose NKM-N, RP-N, GSVQ-ST, NKM-LARS and ODL-0-N. NKM-N and RP-N appear to be the most appealing combinations because they are both hyper-parameter free and have fast feature coding and dictionary learning algorithms. We also included GSVQ-ST, NKM-LARS and OMP-N even though they need hyper-parameter tuning because NKM-LARS was the top performer, and GSVQ-ST and OMP-N have fast feature coding approaches.

The Effects of Whitening and Dictionary Size

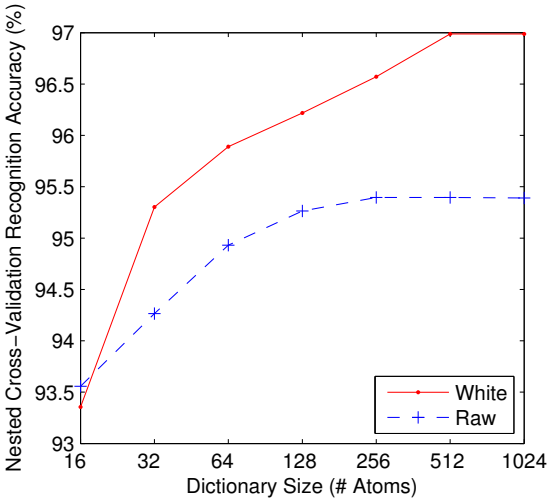
The number of atoms d has a direct influence on the performance of the classifier. Small dictionaries are easy to learn, but are more limited in representational power than large dictionaries. On the contrary, large dictionaries are more flexible, but can have problems with *dead atoms* (Mairal et al., 2009). To evaluate the effect of dictionary size on accuracy, we varied the number of atoms d and looked at how performance changed.

The results are presented in Figure 5.8. As we can see, increasing d improves the results up to a limit where no additional gains are possible. The plateau indicates that dictionary learning is unable to learn new useful features and has reached a limit in its representative capability. Figure 5.8 also suggests that using $d = 300$ atoms is a good trade-off between performance and computational complexity. We will therefore use $d = 300$ atoms in the following experiments.

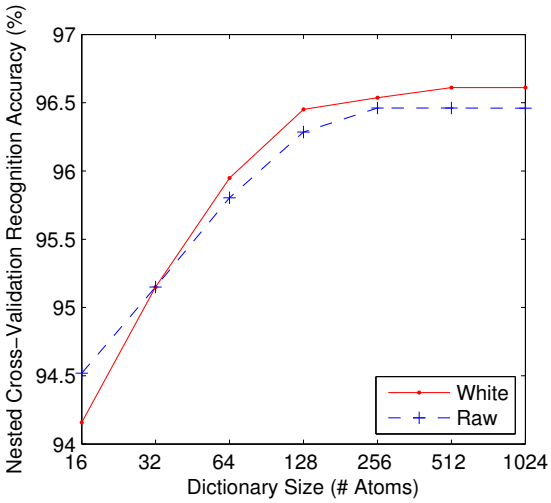
We also looked at the effects of ZCA whitening. As shown in Figure 5.8, we trained dictionaries with various sizes by removing whitening. As we can see, whitening has a great influence on performance. The accuracy improves when whitening is used, and not using it degrades the accuracy. We can see visually the impact of whitening in Figure 5.9 and Figure 5.10. The first figure shows a dictionary trained without whitening, while second figure shows a dictionary trained with whitening. The dictionaries are presented in four groups: the gray channel, the RGB channels, the depth channel and the depth normals. As expected, using whitening allows learning dictionary atoms that look like



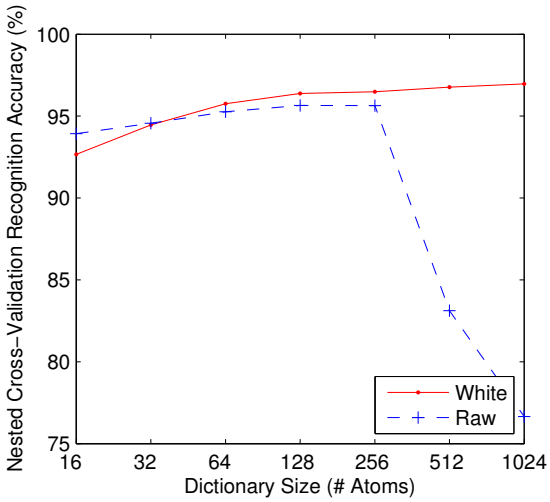
NKM-LARS



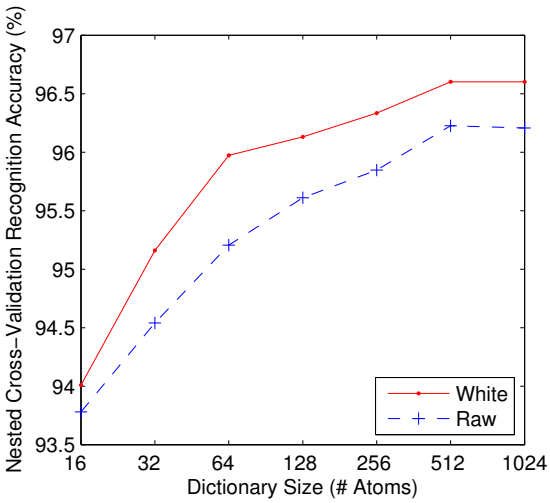
GSVQ-ST



OMP-N



NKM-N



RP-N

Figure 5.8: The effects of dictionary size on recognition accuracies. Increasing the dictionary size improves the results up to a limit (plateau) where no more performance increase is possible.

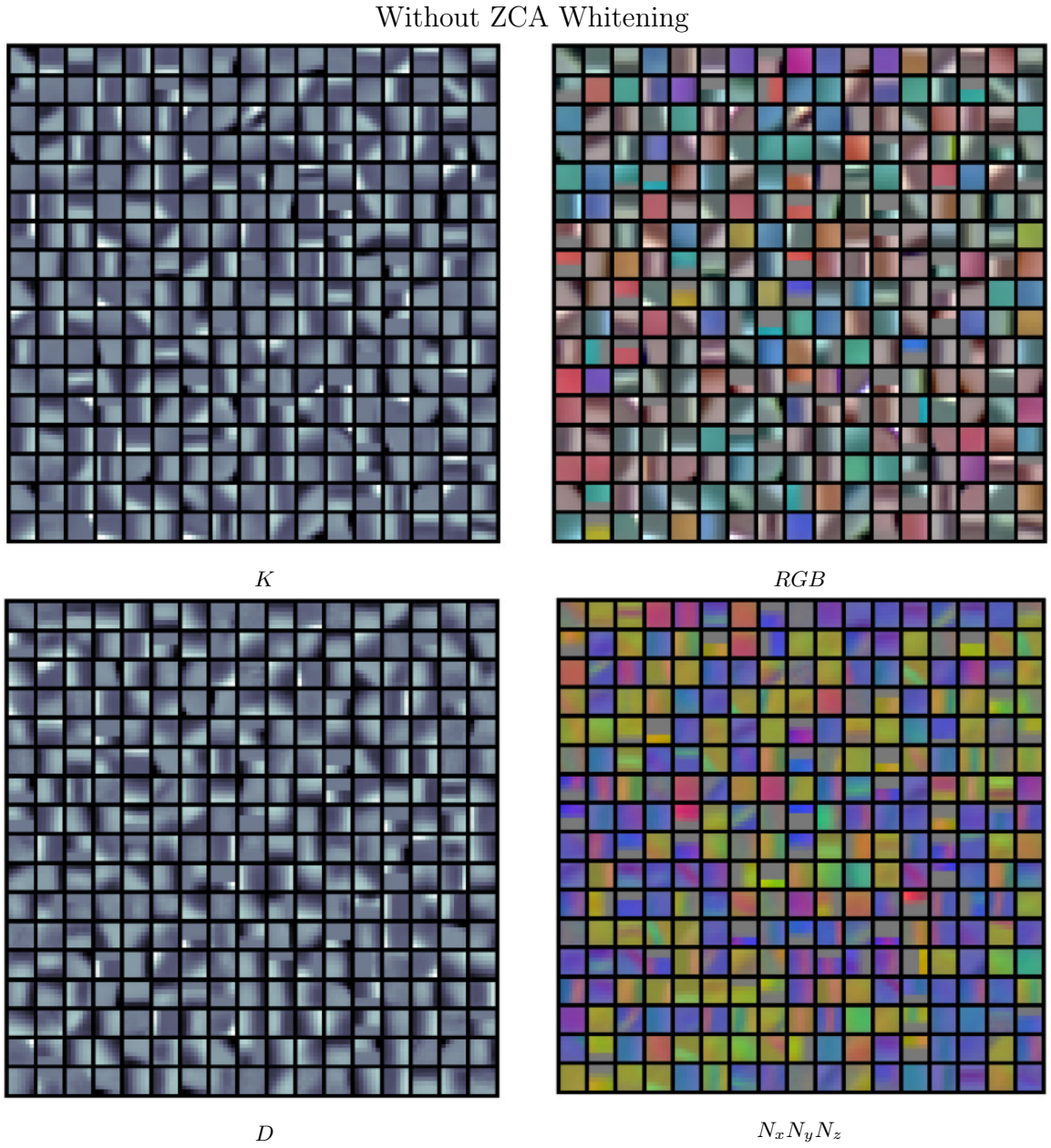


Figure 5.9: A dictionary D of 300 atoms (each square is an atom $D^{(j)}$) trained on the Cornell dataset. The dictionary is presented in four groups: the gray (K) channel, the RGB channels, the depth (D) channel and the depth normals ($N_x N_y N_z$). The dictionary was trained without ZCA whitening. The dictionary has learned atoms that do not look like localized and oriented Gabor filters.

With ZCA Whitening

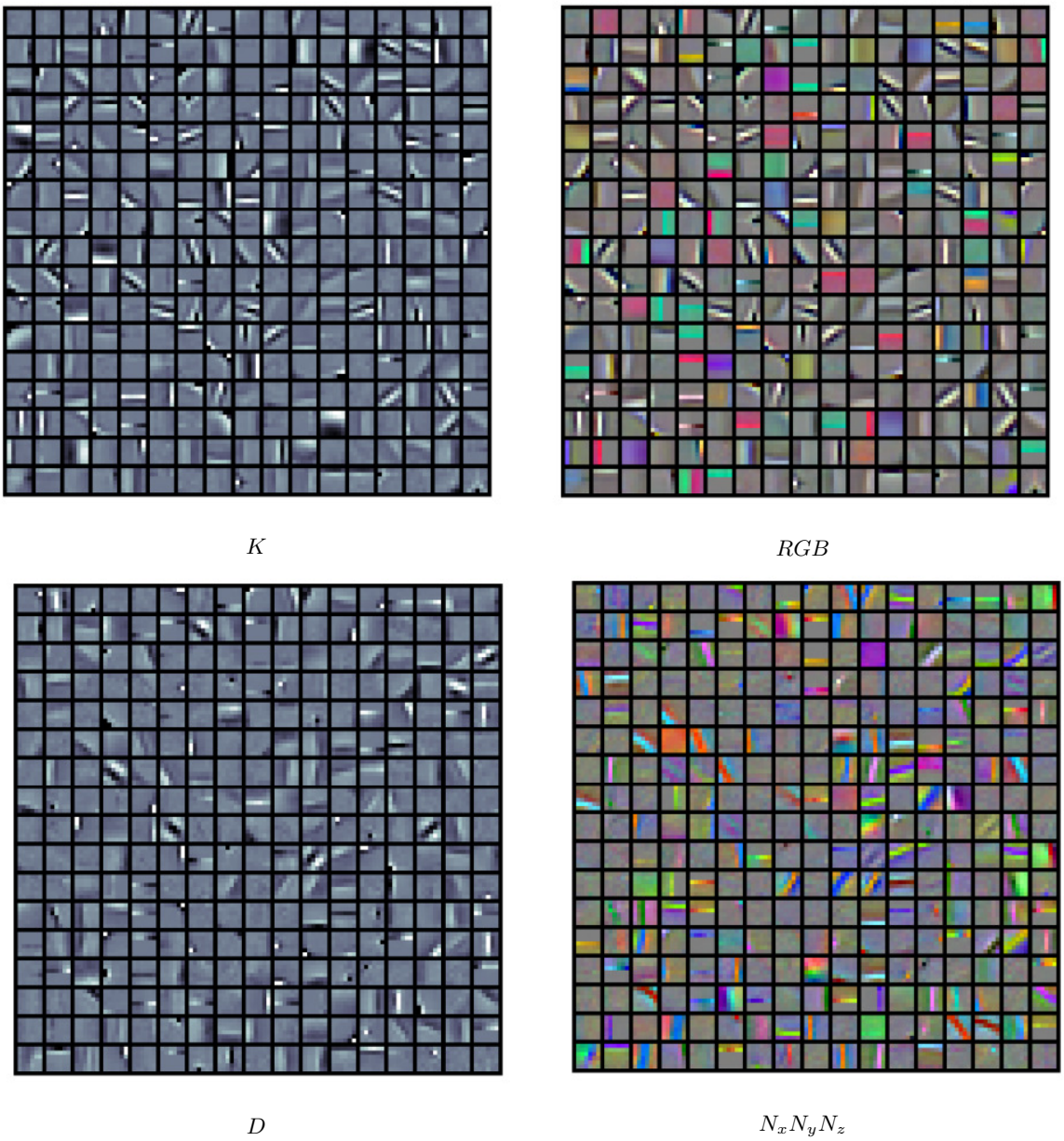


Figure 5.10: A dictionary D of 300 atoms (each square is an atom $D^{(j)}$) trained on the Cornell dataset. The dictionary is presented in four groups: the gray (K) channel, the RGB channels, the depth (D) channel and the depth normals ($N_xN_yN_z$). The dictionary was trained with ZCA whitening. The dictionary has learned atoms that look like localized and oriented Gabor filters.

Gabor filters (Marĉelja, 1980). When whitening is removed, the dictionary atoms do not look like Gabor filters. This is because whitening removes linear dependencies from the data and help the dictionary learning approach to better capture the nonlinear dependencies.

5.4.2 Grasp Detection Results

The cross-validation error rates (in %) for grasp detection are reported in Table 5.3. For image-wise and object-wise split respectively, the lowest detection errors were obtained by NKM-N with 10.60% and GSVQ-ST with 11.21% using a dictionary of $d = 300$ atoms. As comparison, the cascade of multi-layer perceptrons of Lenz et al. (2015) obtained 26.10% and 24.40%, the pre-trained AlexNet of Redmon and Angelova (2015) achieved 12.00% and 12.90%, and our ResNet obtained 10.85% and 11.86%.

One particular result that we can see in Table 5.3 is that Redmon and Angelova (2015)'s AlexNet has relatively similar accuracies than our ResNet. We expected our ResNet to have better performance, even though it was not pre-trained on ImageNet. This is because our ResNet used all input modalities, while Redmon and Angelova (2015) only relied on RGD due to pre-training on ImageNet. Also, our network has residual connections that facilitate training, while AlexNet is a standard network without residual connections.

We therefore experimented with AlexNet in order to better understand the difference between our experimental framework and the one of Redmon and Angelova (2015). The main difference between the two frameworks is the number of training images generated by data augmentation. We used on-line data augmentation while they opted for off-line data augmentation. They generated 3000 images per training observations, which resulted in a training fold containing 2,124,000 unique images. They then fine-tuned their network for 25 epochs, which amounted to processing 53.1 M images. In our case, we trained for 200 epochs on the original fold size of 708 images, but generated new observations at each epoch. Our network only saw a total of 141,600 images, which corresponds to 15x fewer observations.

We fine-tuned AlexNet pre-trained on ImageNet using the same hyper-parameters and the same input modalities (RGD) as Redmon and Angelova (2015). We obtained 18.30% and 18.68% rectangle metric errors for the image-wise and object-wise splits. This represents absolute performance reductions of 6.30% and 5.78% respectively. Although our ResNet had performance similar to Redmon and Angelova (2015)'s AlexNet, our ResNet

Table 5.3: Cross-validation detection results for the Cornell dataset. Numbers are grasp detection accuracies in percent (%).

Approaches	Detection Error (%)	
	Image-wise	**Object-wise**
Jiang et al. (2011)	39.50	41.70
HOG + ELM Kernel (Sun et al., 2015)	35.20	-
SAE, struct. reg. two-stage (Lenz et al., 2015)	26.10	24.40
Two-stage closed-loop (Wang et al., 2016)	14.70	-
Pre-trained AlexNet (Redmon and Angelova, 2015)	12.00	12.90
NKM-LARS	11.33	11.93
GSVQ-ST	11.28	**11.21**
OMP-N	10.66	11.44
NKM-N	**10.60**	11.83
RP-N	12.30	13.39
AlexNet (ours)	18.30 (19.21)	18.68 (19.53)
ResNet (ours)	10.85 (12.20)	11.86 (12.26)

had better performance than AlexNet trained with our framework. Our experimental framework was therefore more difficult.

Investigating Overfitting in ResNet

One important aspect to consider when training deep networks is overfitting. Our ResNet has millions of parameters, which makes it relatively easy for the network to overfit to the training data. This is particularly true when the number of observations is small, as with the Cornell task. To make sure that our ResNet did not overfit, we monitored the gap between the train and test loss during training. Figure 5.11 presents the convergence curves of our ResNet for each five cross-validation folds. As shown in the left column, the train and test losses have a steady and similar decrease. Note that we did not use Dropout (Srivastava et al., 2014) to further regularize the network. These results showed that our network was not in an overfitting regime.

5.5 Discussion

5.5.1 Limitations of the Cornell Task

One particular result that we observed from the detection results in Table 5.3 is that the sparse dictionary learning approaches have similar accuracies than the CNN approaches.

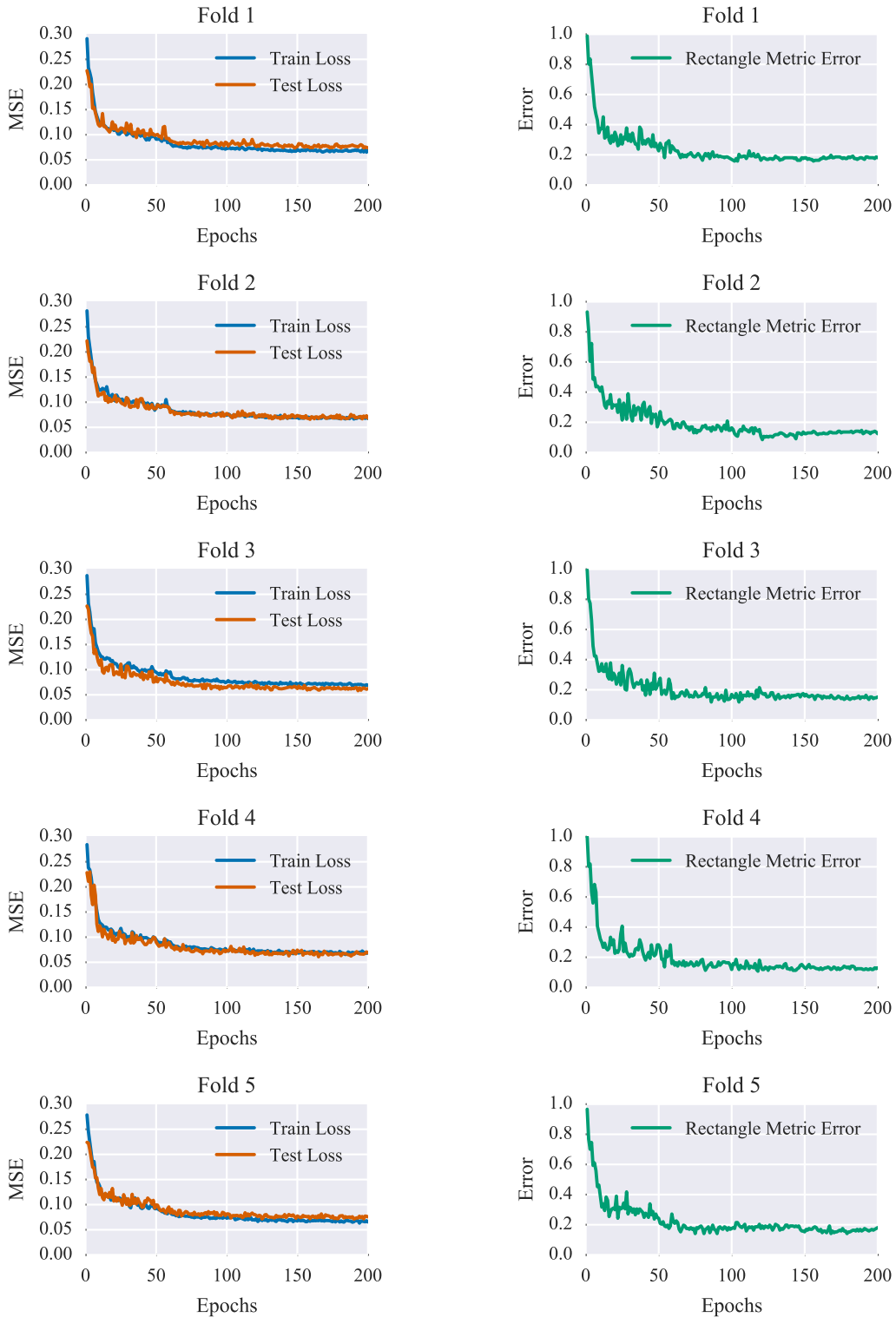


Figure 5.11: Convergence curves of our ResNet during training. The left column shows both the train and test loss convergence for each fold, while the right column shows the rectangle metric error convergence. The small difference between the train and test loss indicates no overfitting.

Figure 5.12: Illustration of the low diversity of the Cornell task. The object on the left was labelled as *large spoon*, while the object on the right was labelled as *spatula*. The images look similar even though the objects are different, which reduces diversity.

We expected the CNNs to obtain higher accuracy than the simpler sparse dictionary learning approaches, since many recent works using CNNs pushed forward the state of the art in several domains. Our work on the Cornell dataset has however led us to three explanations that can explain why this is the case.

The first one is low diversity. As we can see with the examples shown in Figure 5.6, all images of the Cornell dataset were captured in the same environment. The background table, the lighting conditions and the angle of the kinect camera to the table were fixed throughout the whole duration. This removes many instances with complex light conditions and depth map structures that would arise from real-world scenarios. As a result, images of different objects can look similar, as shown in Figure 5.12. This is further illustrated by the small difference in detection accuracy between image-wise split and object-wise split. For instance, NKM-LARS obtained 88.67% on image-wise split and 88.07% on object-wise split. Judging only with the accuracy results, this suggests that grasping a novel object is only a little harder than grasping a known object from another point of view. Intuitively, we should have observed a larger accuracy gap between the two splits due to the great diversity of graspable and non-graspable objects.

The second one is the small size of the Cornell dataset. The Cornell dataset contains 885 RGBD images and each image is labelled with few grasp rectangles. This places an incentive on training smaller approaches that do not require a lot a data, like sparse dictionary learning, or using elaborated regularization methods to train a CNN. For instance, Redmon and Angelova (2015) relied on pre-training to avoid overfitting. They

Table 5.4: Speed comparison at test time on the Cornell dataset images. The sparse dictionary learning approaches were implemented on CPU, so they can only be compared directly with Jiang et al. (2011), "SAE, struct. reg. two-stage" and "HOG + ELM Kernel".

Approach	Speed (fps)
Jiang et al. (2011)	-
SAE, struct. reg. two-stage (Lenz et al., 2015)	0.07
HOG + ELM Kernel (Sun et al., 2015)	0.09
Two-stage closed-loop (Wang et al., 2016)	7.11
Pre-trained AlexNet (Redmon and Angelova, 2015)	13.15
NKM-N	< 0.01
NKM-LARS	< 0.01
GSVQ-ST	< 0.01
OMP-N	< 0.01
NKM-N	< 0.01
RP-N	< 0.01
AlexNet (ours)	**13.72**
ResNet	11.5

first pre-trained their CNN on ImageNet, then fine-tuned it with off-line data augmentation on the Cornell task. In order to take in account the fact that ImageNet contains RGB images instead of RGBD images, they replaced the blue (B) channel with the depth (D) channel during fine-tuning and evaluation. Such pre-training approach is clearly sub-optimal, since light information of the blue channel has little in common with depth information of the depth channel. Also, removing the blue channel takes away information correlation between the blue channel and the other red and green light-based channels, as well as the depth channel and depth normals. This goes against the recommendations of previous works that provided empirical evidence that using all channels always results in the highest accuracy (Lenz et al., 2015; Bo et al., 2013).

Finally, the third one is the use of sliding window to perform grasp detection with sparse dictionary learning. As we explained in Section 5.2.2, detection with sliding window works by performing a grid search in grasp rectangle space. For each grasp rectangle candidate, we compute the graspability score using the underlying grasp rectangle image. We first extract the feature vector with spatial pyramid matching then pass it to the linear SVM to compute its score. The rectangle with the highest score is selected as predicted grasp rectangle. Detection with sliding window is computationally

demanding, but in return is exhaustive. We can easily trade off computational speed with detection accuracy by increasing the granularity of the grid search. The more grasp rectangle candidates we generate, the more opportunities we have to find the regions with high graspability. As we can see in Table 5.4, sparse dictionary learning paid a computational price to obtain high detection accuracies. The grid search took several minutes per image to complete, even though the selected LARS, OMP and ST feature coding approaches had relatively low computational complexities. In comparison, CNNs can process high-dimensional images with more ease. This is possible because CNNs have efficient GPU implementations and rely on one shot computation instead of grid search in grasp rectangle space.

5.5.2 Limitations of the Grasp Rectangle Metric

The grasp rectangle is a useful tool to tackle grasp detection as a vision problem. However, the grasp rectangle metric used to evaluate the grasp detection accuracy has some limitations. To illustrate them, we show in Fig 5.13 examples of predicted grasp rectangles from our ResNet. The predicted grasp rectangle for the squeegee was the second bold rectangle from the left, the predicted grasp rectangle for the scissors was the second bold rectangle in the middle, while the predicted grasp rectangle of the sunglasses was the leftmost bold rectangle near the lens.

According to the rectangle metric, the detection for the squeegee is considered a success, while the detection for the scissors and the sunglasses are considered failures. It is however conceivable that grasping both the scissors and the sunglasses using the candidate grasp rectangles would have been a success, even though the rectangle metric labels them as failures. A gripper with two parallel plate-gripping fingers would have certainly succeeded, even though the candidate grasp rectangles are wider than their corresponding closest ground truth rectangle. Although the rectangle metric is convenient due to its resemblance to bounding box in vision, it can negatively bias the reported performances.

5.6 Conclusion

In this chapter, we explored representation learning on the problem of grasp localization. We performed a comparative study using two popular representation learning frameworks: sparse dictionary learning and convolutional neural networks. We compared several approaches in both frameworks by evaluating them on both grasp recog-

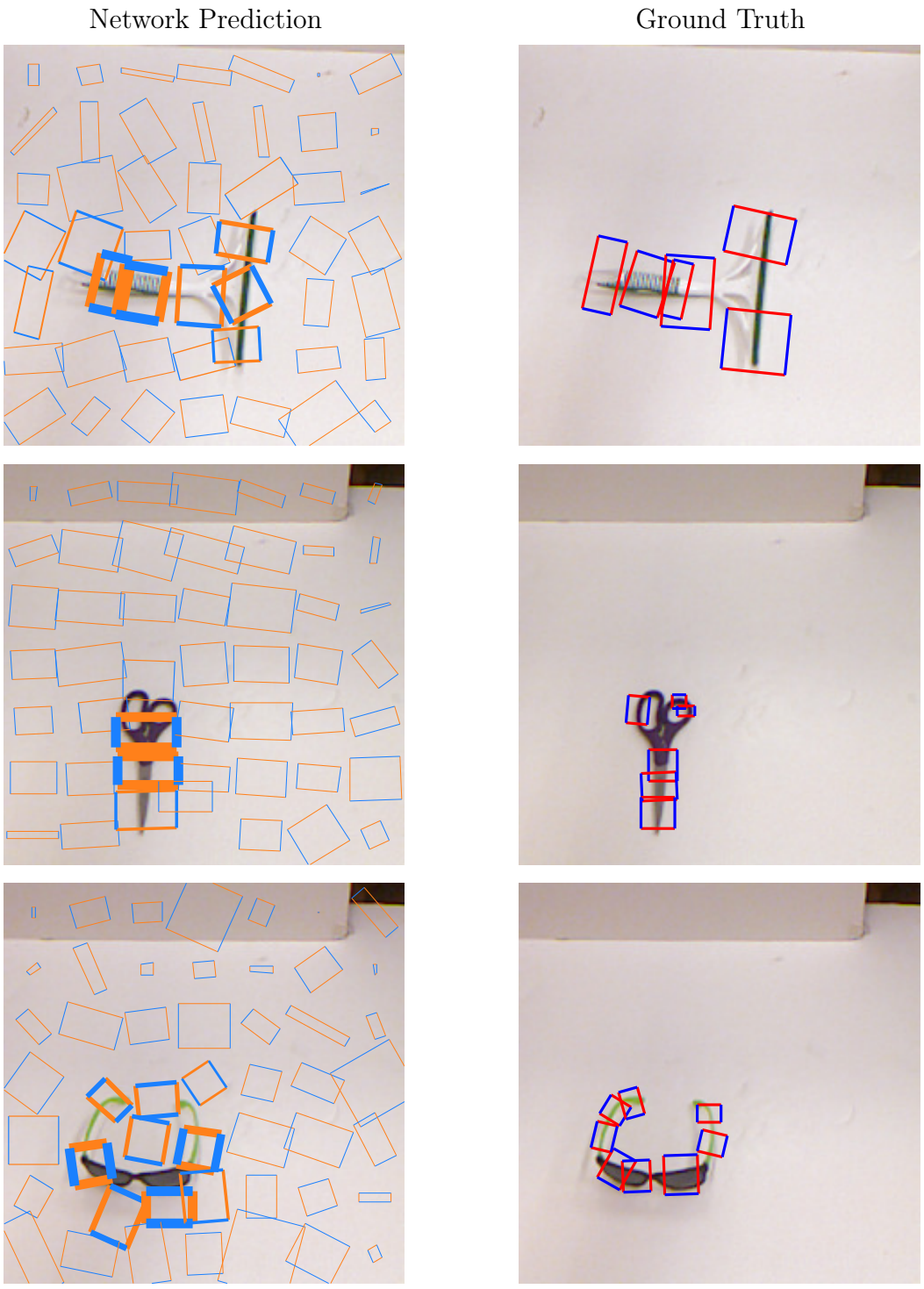


Figure 5.13: Examples of a detection success (top example) and failures (middle and bottom). The left column shows the candidate rectangles predicted by our network, while the right column shows the ground truth. A rectangle with a large width indicates that the network is confident about its prediction. Although the middle example is considered a failure according to the rectangle metric, the grasping system would still have been able to grasp the scissors using the predicted rectangle.

nition and detection problems. The analysis of our empirical evaluation illustrated the advantages and disadvantages of both frameworks.

The main result is that sparse dictionary learning approaches obtained similar accuracy as convolutional neural networks on the Cornell task. There are three reasons for this. The first two are that the Cornell task has low variability and few observations. This benefits simpler approaches, like sparse dictionary learning, that do not require a lot of data. And the third one is the use of sliding window for grasp detection with sparse dictionary learning. We can easily trade off computational speed for accuracy, but it makes the approach unsuitable for real-time processing.

In conclusion, the Cornell dataset is a reminder that a high performance on a dataset can sometimes be misleading. It does not always reflect the true limitations of an approach, particularly when the dataset is small and the task is simple. Indeed, we were able to obtain high accuracy as much with the *shallow* approaches of sparse dictionary learning as with the *deep* approaches of convolutional neural networks. It remains however that convolutional neural networks will have the upper hand on shallow approaches when the tasks get more complex and more data become available.

Chapter 6

Deep Learning Development: Activation Function

In this chapter, we present our contribution on parametric activation functions. We propose a parameterization of the Exponential Linear Unit (ELU) that controls different aspects of its shape and propose learning it during training. We refer to our activation function as Parametric ELU (PELU). We also present an experimental evaluation on the tasks of auto-encoding and object recognition, and perform a series of experiments to better understand our parameterization.

6.1 Introduction

The activation function is a central component in deep learning. As we saw in Section 3.2, it must be carefully chosen to avoid problems like *dead units*, *vanishing gradients* and *bias shift*. We also presented several activation functions that were proposed to deal with these problems, and highlighted the family of *parametric activation functions* that have trainable parameters controlling different aspects of their shape.

The main advantage of using a parametric activation function is that it adds another layer of flexibility. The parameters of the activation function can be seen as being part of the parameter space of the overall network, which includes the weights and biases of the linear transformations. Minimizing the loss function in this new parameter space allows the data to guide training towards a more proper shape for the task than a pre-defined one. Starting with an initial configuration, the shape changes at every iteration until it converges to a local minimum of the loss function. We can see a parametric activation function as a kind of evolutive activation that adapts its shape during training based

on the information provided by the gradient.

In this chapter, we investigate the effect of parameterizing the Exponential Linear Unit (ELU) (Clevert et al., 2016). As we explained in Section 3.2, the ELU is a non-saturated activation function that addresses the problem of bias shift. It behaves like identity for positive arguments, to lower vanishing gradients, and has an exponential decay towards -1 for negative arguments, to lower bias shift. Given that the ELU is a fixed activation function, our goal in this chapter is to define parameters controlling different aspects of its shape and learn them during training. We refer to our parametric version of the ELU as Parametric ELU (PELU).

To investigate the effects of our parameterization of ELU, we opted for an experimental evaluation on the problem of object recognition. The goal in object recognition is to recognize the class of an object in a RGB image. Applications such as face verification, robotic grasping or autonomous driving all require the fundamental skill of object recognition in order to carry out their tasks. They must perform complex scene understanding by first identifying the different elements of the scene, then by identifying the class of the objects. The performance of the overall visual system depends on the quality of the representation extracted from the image. For this reason, it is an excellent problem to evaluate novel advances in representation learning.

Here is a summary of the extent of our work on parameterizing ELU:

1. We define parameters controlling different aspects of the shape of ELU and show how to learn them during training. The particularity of our parameterization is that it preserves differentiability by acting on both the positive and negative parts of the function. It has the same computational complexity as ELU and adds only $2L$ additional parameters, where L is the number of activation functions in the network.

2. We perform an experimental evaluation of our parameterization on the problem of object recognition. We evaluate our work on the MNIST, CIFAR-10/100 and ImageNet tasks using ResNet (Shah et al., 2016), Network in Network (NiN) (Lin et al., 2013), All-CNN (Springenberg et al., 2015), VGG (Simonyan and Zisserman, 2014) and Overfeat (Sermanet et al., 2014). Overall, our results indicate that our parameterization obtains better performances than ELU.

3. We evaluate the impact of using Batch Normalization (BN) before our PELU activation, following the experiment of Clevert et al. (2016) that observed empir-

ically that BN before ELU was detrimental. We show that BN before PELU is also detrimental, although the increase in error rate is smaller.

4. We experiment with different configurations of our parameterization. We evaluate the impact of learning the inverse of each parameter instead of the original one. We show empirically that the proposed configuration obtains the best performance and we provide intuitions to explain why it is the case.

5. We finally show the parameter convergence of each PELU activation during training in a VGG network. We observe different behaviors that highlight the effects of our parameterization.

The content of this chapter is organized as follows. In Section 6.2, we elaborate on our parameterization of ELU. We start with a reminder of previous parametric activation functions, then introduce our parameterization with its gradient calculations. In Section 6.3, we present our experimental evaluation. We start with the problem of auto-encoding in the context of unsupervised learning on the MNIST task. We then follow up with object recognition in the context of supervised learning on the CIFAR-10, CIFAR-100 and ImageNet 2012 tasks. In Section 6.4, we present a discussion on our activation function and experimental results. We elaborate on the effect of batch normalization and investigate vanishing gradients. We finally conclude this chapter in Section 6.5.

6.2 Model

In this section, we introduce our parameterization of ELU. We start with a reminder of previous parametric activation functions, and refer the reader to Section 3.3 for more details. We then elaborate on our parameterization of ELU, which we refer to as PELU. We finally detail the gradient calculation for backprop to perform gradient descent.

6.2.1 Parametric Activation Function

The Rectified Linear Unit (ReLU) is nowadays the most common activation function in deep learning (Nair and Hinton, 2010; Jarrett et al., 2009; Glorot et al., 2011; Krizhevsky et al., 2012). It quickly gained popularity after Glorot et al. (2011) trained a state-of-the-art deep network solely based on labelled data. It was then adopted by most people in the deep learning community after Krizhevsky et al. (2012) won the ILSVRC 2012

challenge. Since then, the ReLU has become the standard activation function in deep networks.

Recall from Section 3.2 that the ReLU lowers the likelihood of vanishing gradients, but increases the likelihood of dead units. Indeed, its derivative of zero for negative arguments can cause a unit to always be activated to zero on all observations from the dataset. To address this problem, the Leaky ReLU (LReLU) (Maas et al., 2013) proposes to use a small positive slope for the negative part of the ReLU. However, grid search must be used to select its value, which can make the overall training phase too computationally intensive.

Parametric ReLU (PReLU) (He et al., 2015) proposed a simple yet elegant solution to remove grid search. The idea of PReLU is to include the slope in the parameter space of the network and learn it during training. PReLU is part of the broad family of *parametric activation functions*.

Other types of parameterization for the ReLU have been previously proposed. For instance, the Adaptive Piecewise Linear (APL) unit (Agostinelli et al., 2014), the S-Shaped ReLU (SReLU) (Jin et al., 2016) or the Maxout (Goodfellow et al., 2013). These parameterizations have however the limitation that they have points at which the function is non-differentiable. For SReLU, the number of these points is fixed at two. For APL, it increases linearly with the number S of Hinged-shaped functions, while for Maxout, it increases linearly with the number K of linear transformations. Gradient descent can be affected by these non-differentiable points when the slopes of the two linear segments around the non-differentiable points are relatively different from the chosen sub-derivative. A variation of the input around these points can cause a large jump in the derivative, and thereby cause oscillations (LeCun et al., 2015). Maxout also has the additional requirement of multiplying the number of weights and biases to be learned by K. This increases the likelihood of overfitting by a great amount for deep networks who already have a lot of parameters. It can even prevent training altogether because of hardware limitations.

In view of these recent works that showed that parameterizing an activation function improves performance, we seek in this chapter to parameterize the Exponential Linear Unit (ELU). Our goal is to define different parameters controlling various aspects of its shape by making sure that the function stays differentiable at all points during training. We now present the ELU as well as our parameterization.

6.2.2 Parametric Exponential Linear Unit

The Exponential Linear Unit (ELU) (Clevert et al., 2016) is an alternative to ReLU that was proposed to deal with the problem of *bias shift*. When bias shift happens, an update of the weight during training can cause oscillations due to the propagation effect. ReLU is more likely to experiment this problem because the function only outputs non-negative values. The idea of ELU is to output negatives values for negative arguments, while still being identity for positive arguments to reduce vanishing gradients. The presence of negative values in the vector of activated neurons helps to lower the mean neuron activation, and thereby, reduce oscillations. Note that bias shift can still happen in deep networks because the largest negative value is bounded to -1, while the largest positive values is unbounded.

The ELU is defined as identity for positive arguments and has an exponential decay towards -1 for negative arguments:

$$f_{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(\exp(x) - 1) & \text{otherwise} \end{cases}, \tag{6.2.1}$$

where parameter $a \in \mathbb{R}^+$ controls the decay. Clevert et al. (2016) opted for a decay of $a = 1$ in order to have differentiability at $x = 0$. For other values $a \neq 1$, the function becomes non-differentiable at $x = 0$. As we have explained in the previous section, non-differentiable points can affect training and should preferably be avoided when possible. For this reason, a parameterization of ELU that simply learns parameter a would break differentiability at $x = 0$ and would risk hinder training.

For this reason, we start by first adding two additional parameters b and c to ELU:

$$f(x) = \begin{cases} cx & \text{if } x \geq 0 \\ a(\exp(\frac{x}{b}) - 1) & \text{if } x < 0 \end{cases}, \quad a, b, c \in \mathbb{R}^+, \tag{6.2.2}$$

We have cx for positive arguments ($x \geq 0$) and $a(\exp(\frac{x}{b}) - 1)$ for negative arguments ($x < 0$). The original ELU can be recovered when $a = b = c = 1$. As shown in Figure 6.1, each parameter controls different aspects of the activation. Parameter c changes the slope of the linear function in the positive quadrant. The larger c, the steeper the slope. Parameter b affects the scale of the exponential decay. The larger b, the smaller the decay. Parameter a acts on the saturation point in the negative quadrant. The larger a, the lower the saturation point. We further constrain the parameters to be positive so that the activation function is monotonic. This way, a reduction in the magnitude of the weights during training always lowers the magnitude of the activation.

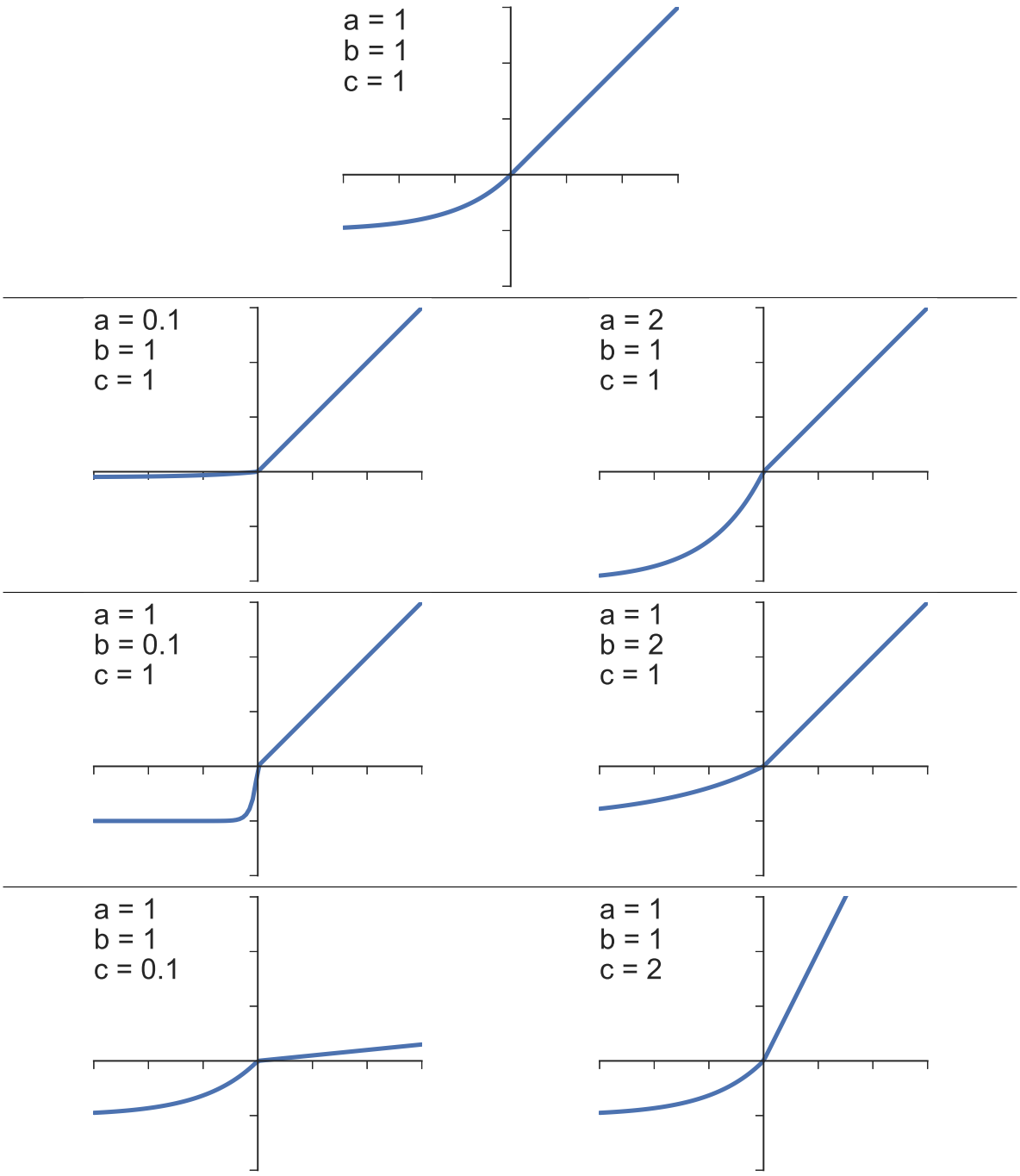


Figure 6.1: Effects of parameters a, b and c on the Exponential Linear Unit (ELU) activation function. The original ELU is shown at the top, where $a = b = c = 1$. Parameter a controls the height of the saturation, parameter b controls the decay towards the saturation and parameter c controls the slope of the linear function.

The parameterization defined in Eq. (6.2.2) allows training to learn different activation functions. For instance, it can converge to a function with a large slope by increasing the value of c, to a function with a slow decay by increasing b or to a function with a low saturation point by increasing a. However, the use of gradient descent with the parameterization of Eq. (6.2.2) would break differentiability at $x = 0$ and possibly hinder training. Thus, we constrain our parameterization to always have differentiability at $x = 0$. We use as constraint the derivative of the negative side to be equal to the derivative of the positive side, both evaluated at $x = 0$:

$$\left.\frac{\partial cx}{\partial x}\right|_{x=0} = \left.\frac{\partial a(\exp(x/b) - 1)}{\partial x}\right|_{x=0} . \tag{6.2.3}$$

Solving Eq. (6.2.3) for c gives:

$$c = \frac{a}{b} . \tag{6.2.4}$$

Incorporating the constraint of Eq. (6.2.4) into the parameterization defined in Eq. (6.2.2) results in the proposed Parametric Exponential Linear Unit (PELU):

$$f_{PELU}(x) = \begin{cases} \frac{a}{b}x & \text{if } x \geq 0 \\ a(\exp(\frac{x}{b}) - 1) & \text{if } x < 0 \end{cases}, \quad a, b \in \mathbb{R}^+ . \tag{6.2.5}$$

With this parameterization, in addition to changing the saturation point and exponential decay respectively, both a and b adjust the slope of the linear function in the positive part to ensure differentiability at $x = 0$. It has the positive side effect of reducing the number of trainable parameters as well.

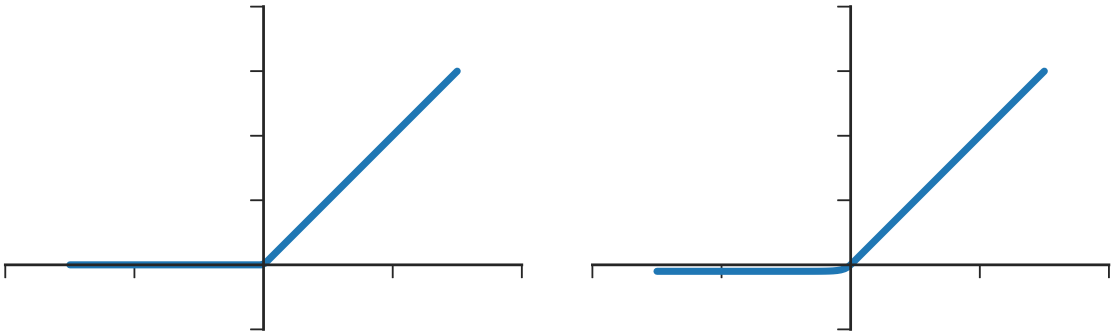
6.2.3 Optimization by Gradient Descent

The parameters of our parameterization are trained simultaneously with all the other parameters of the network. The update rules are defined as follows. Using the chain rule of derivation, the derivative of objective E with respect to a and b for one layer is:

$$\frac{\partial E}{\partial a} = \sum_i \frac{\partial E}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial a}, \qquad \frac{\partial E}{\partial b} = \sum_i \frac{\partial E}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial b} , \tag{6.2.6}$$

where i sums over all elements of the tensor x on which f_{PELU} is applied. The terms $\frac{\partial E}{\partial f(x_i)}$ are the gradients propagated from the above layers, while $\frac{\partial f(x)}{\partial a}$ and $\frac{\partial f(x)}{\partial b}$ are the gradients of f with respect to a, b:

$$\frac{\partial f(x)}{\partial a} = \begin{cases} \frac{x}{b} & \text{if } x \geq 0 \\ \exp(x/b) - 1 & \text{if } x < 0 \end{cases} , \tag{6.2.7}$$



ReLU PELU with $a = 0.1$ and $b = 0.1$

Figure 6.2: PELU is similar to ReLU when both parameters a and b are closed to zero. As a result, the network does not suffer a weight decay loss when both a and b converge near zero.

$$\frac{\partial f(x)}{\partial b} = \begin{cases} -\dfrac{ax}{b^2} & \text{if } x \geq 0 \\ -\dfrac{a}{b^2}\exp(x/b) & \text{if } x < 0 \end{cases}. \tag{6.2.8}$$

In addition, we impose the constraint that the parameters are positive by projecting them back to the positive quadrant after the gradient update. In other words, we always make sure that the parameters are larger than a small positive value. In our experiment, we used 0.1. Thus, given that we use stochastic gradient descent with momentum (Sutskever et al., 2013), the update rules are as follows:

$$\begin{aligned} \Delta a &\leftarrow \mu \Delta a - \alpha \frac{\partial E}{\partial a} \\ a &\leftarrow \max\{a + \Delta a, 0.1\} \end{aligned}, \qquad \begin{aligned} \Delta b &\leftarrow \mu \Delta b - \alpha \frac{\partial E}{\partial b} \\ b &\leftarrow \max\{b + \Delta b, 0.1\} \end{aligned}, \tag{6.2.9}$$

where μ is the momentum and α is the learning rate.

Unless specified otherwise, we always use a ℓ_2 weight decay regularization in our experiments. We apply the regularization on both the weights and biases of the linear transformations, as well as on the parameters of the parametric activation functions throughout the network.

The effect of weight decay on PELU can be seen as placing a prior on converging to ReLU. Indeed, weight decay encourages the parameters to have a low magnitude. As we can observe in Figure 6.2, PELU looks similar to ReLU when both a and b are close to zero. As a result, the network does not suffer a weight decay loss when PELU converges to ReLU.

6.3 Experiments

In this section, we present our experimental evaluation of our parameterization of ELU. We performed all our experiments on supervised learning, except for the first experiment which was on unsupervised learning. For supervised learning, we opted for object recognition, while for unsupervised learning, we opted for auto-encoding. In all cases, we used datasets containing RGB images as observations.

In addition to these experiments on performance evaluation, we also performed a series of three experiments to better understand our parameterization. The first one is related to the effect of Batch Normalization (BN) (Ioffe and Szegedy, 2015), the second one is related to the effect of learning the inverse of parameters a and b, while the third one is related to the progression of the shape of each PELU activations.

6.3.1 MNIST Unsupervised Learning

As first experiment, we performed unsupervised learning to evaluate our approach. We opted for image auto-encoding on the MNIST (LeCun et al., 1998) task without labels. The MNIST dataset contains 28×28-dimensional images of handwritten digits from 0 to 9. The train set has 60,000 observations, while the test set has 10,000 observations. It is one of the most popular datasets in the machine learning community and is a great starting point to compare approaches.

We trained an auto-encoder to regenerate the input observation under compressive constraints. We refer to this network as DAA-Net. The encoder is composed of four fully connected layers of sizes 1000, 500, 250, 30, while the decoder is symmetrical without tied weights (Desjardins et al., 2015). Each linear transformation (except the last one) is followed by an activation layer, which is either ELU alone, PELU alone or a combination of Batch Normalization (BN) (Ioffe and Szegedy, 2015) followed by ReLU. We also added Dropout (Srivastava et al., 2014) with a drop probability of 0.2 after each activation layer. We trained DAA-Net with RMSProp (Tijmen Tieleman, 2012) at a learning rate of 0.001, smoothing constant of 0.9 and a batch size of 128.

We do not use BN before ELU because Clevert et al. (2016) obtained higher performance when using ELU alone. Thus, we remove BN before PELU since PELU is based on ELU. We will provide additional experimental evidence that BN before ELU and PELU is detrimental in Section 6.3.3. Note that we still use BN before ReLU as suggested by Ioffe and Szegedy (2015).

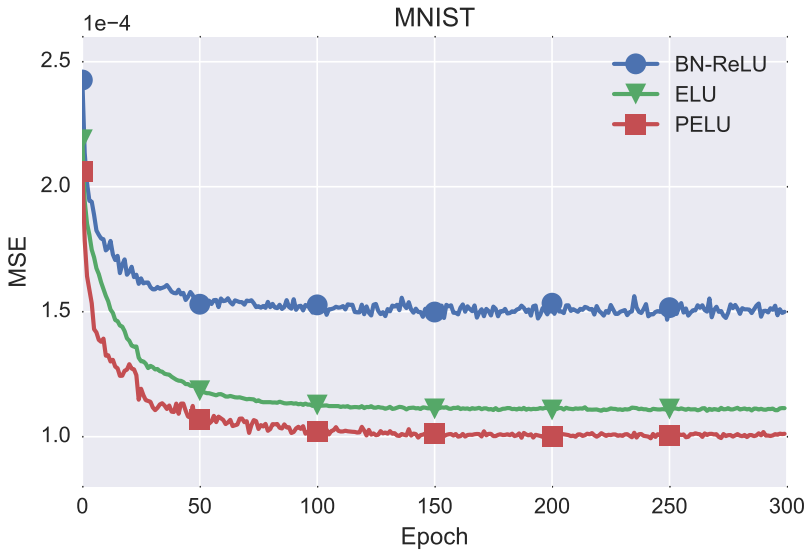


Figure 6.3: Auto-encoder results on the MNIST task. We compare PELU to ELU, and include BN-ReLU as additional reference. Compared to ELU, PELU obtained a lower test mean squared error.

Figure 6.3 presents the results of our experiment on the MNIST dataset. The curves illustrate the progression of the mean squared error (MSE) on the test set, averaged over five tries. The blue curve (dot marker) represents BN-ReLU, the green curve (downward triangle marker) represents ELU and the red curve (square marker) represents PELU. The results show that our parameterization of ELU improves the reconstruction error over standard ELU and ReLU. PELU converged at a MSE of approximately 1.04e−4, while ELU converged at a MSE of 1.12e−4 and ReLU at a MSE of 1.49e−4.

This first experiment on MNIST suggests that our parameterization of ELU improves performance over standard ELU. However, the MNIST task is known to be easy nowadays. A good performance on this task is usually not sufficient to show the validity of a proposed approach. For this reason, we chose to do more experiments in supervised learning.

6.3.2 CIFAR-10/100 Object Recognition

As a second experiment, we performed object recognition on the CIFAR-10 and CIFAR-100 tasks (Krizhevsky, 2009). Both tasks contain $3 \times 32 \times 32$-dimensional RGB images with a train set of 50,000 observations and a test set of 10,000 observations. CIFAR-10 regroups the images into 10 classes, while CIFAR-100 regroups the images into 100 classes.

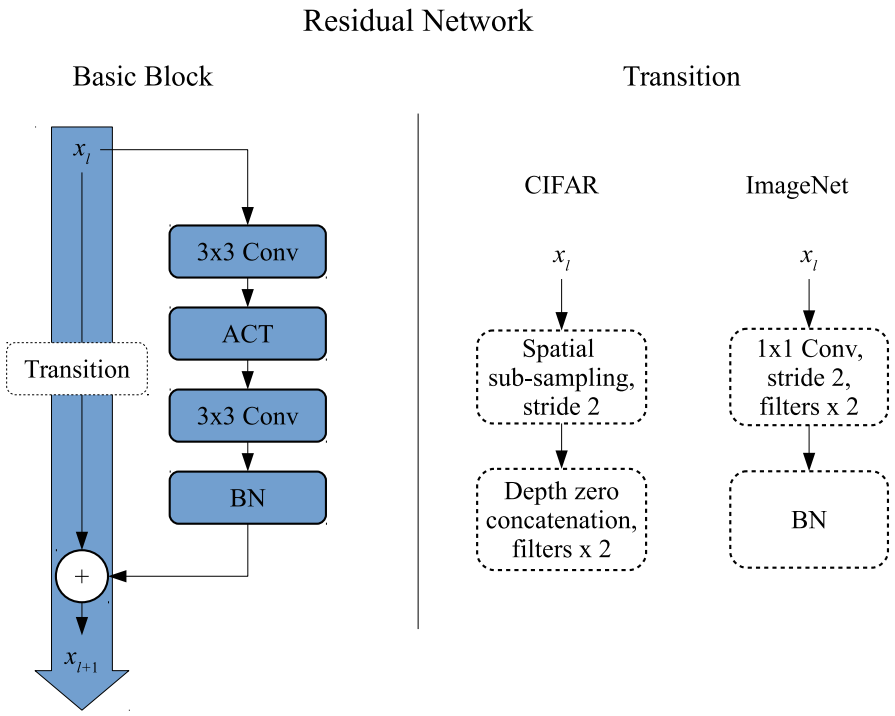


Figure 6.4: Residual network building block structure. On the left, the main basic block structure, and on the right, the transition block structure for reducing the input spatial dimensions and increasing the number of filters. For our CIFAR experiments, we opted for sub-sampling followed by zero concatenation as transition block, while for our ImageNet experiments, we opted for strided convolution followed by batch normalization as transition block.

We trained a 110-layer residual network (ResNet) (He et al., 2016b; Shah et al., 2016) with the building block structure shown in Figure 6.4. The image on left of Figure 6.4 presents the basic block structure, while the image on the right of Figure 6.4 presents the transition block structure. The goal of the transition block is reducing the spatial dimension of the feature maps, while increasing the number of feature maps. For our CIFAR experiments, we used spatial sub-sampling with a stride of 2 followed by zero concatenation to multiply the number of feature maps by two.

In order not to favor PELU to the detriment of the other activations, we performed minimal changes to the network when changing it. To this effect, we keep the same number of blocks and only replace the ACT module. The ACT module can be either ELU alone, PELU alone, BN-ReLU or BN-PReLU.

To train the network, we used stochastic gradient descent with a weight decay of 1e−3, momentum of 0.9 and batch size of 256. The learning rate starts at 0.1 and is divided by 10 after epoch 81, and by 10 again after epoch 122. We apply standard data aug-

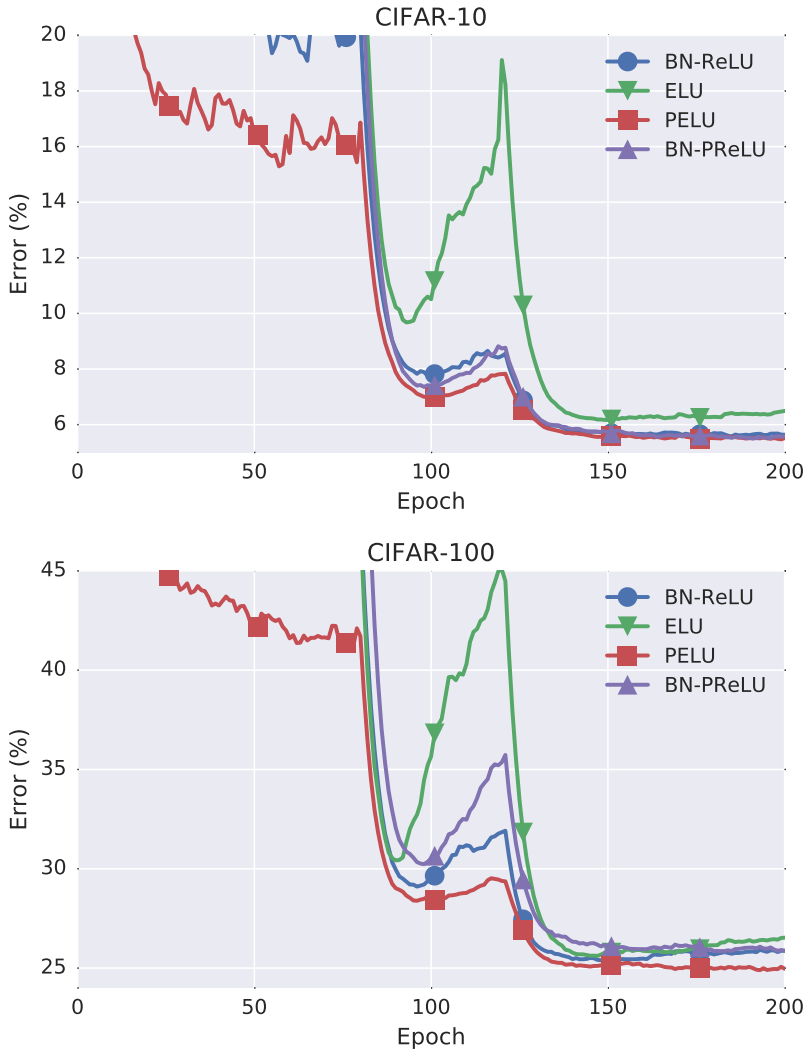


Figure 6.5: ResNet 110 layers test error (in %) medians over five tries on both CIFAR-10 and CIFAR-100 datasets. We compare PELU to ELU, and also include BN-ReLU and BN-PReLU as additional references. PELU has a better convergence and lower recognition errors than ELU.

mentation for this task, which is random crop + horizontal flipping: four pixels are added on each side of the image and a random 32 x 32 crop is extracted, which is then flipped horizontally at random. The original image is used during the test phase.

The results of our experiments on the CIFAR-10 and CIFAR-100 tasks are presented in both Figure 6.5 and Table 6.1. The curves in Figure 6.5 correspond to the median test error rate (in %) over five tries for each epoch on both CIFAR-10 (top image) and CIFAR-100 (bottom image). The blue curve (circle marker) represents BN-ReLU, the green curve (downward triangle marker) represents ELU, the red curve (square marker)

Table 6.1: ResNet 110 layers test error (in %) on both CIFAR-10 and CIFAR-100 datasets. We report the mean error over the last five epochs and the minimum error over all epochs (inside parenthesis) of the median error over five tries. Our parameterization of ELU improves performance over standard ELU.

ACT	CIFAR-10	CIFAR-100
BN-ReLU	5.67 (5.41)	25.92 (24.99)
ELU	6.55 (5.99)	26.59 (25.08)
PELU	**5.51 (5.36)**	**25.02 (24.55)**
BN-PReLU	5.61 (**5.36**)	25.83 (25.50)

represents PELU and the purple curve (upward triangle marker) represents BN-PReLU. In addition, we report the numerical values of their performance in Table 6.1. In each cell of Table 6.1, the value on the left corresponds to the mean error over the last five epochs of the median error over five tries, while the value inside parenthesis corresponds to the minimum over all epochs of the median error over five tries.

Based on the results reported in Table 6.1, we can see that our parameterization obtains the best performances. On CIFAR-10, ResNet obtained a minimum median error rate of 5.36% with PELU, while it obtained a minimum median error rate of 5.99% with ELU, 5.41% with BN-ReLU and 5.36% with BN-PReLU. On CIFAR-100, ResNet obtained a minimum median error rate of 24.55% with PELU, while it obtained a minimum median error rate of 25.08% with ELU, 24.99% with BN-ReLU and 25.50% with BN-PReLU. In comparison to ELU, PELU obtained a relative improvement of 10.52% and 2.11% on CIFAR-10 and CIFAR-100 respectively. It is interesting to note that PELU only adds 112 additional parameters, a negligible increase of 0.006% over the total number of parameters.

The results in Figure 6.5 show that PELU has a better convergence behavior than ELU. Indeed, we can see that ELU has a large test error rate increase at the end of the second stage of the training phase on both CIFAR-10 and CIFAR-100 datasets. PELU also has a small error rate increase around the same epoch, but the increase is lower than ELU. We further observe a small test error rate increase at the end of the training phase for ELU, while PELU converges in a steady way without a test error rate increase. These results show that training a ResNet with our parameterization can improve both the performance and the convergence behavior over a ResNet with ELU activation.

Moreover, PReLU also obtained better performances than ReLU. On CIFAR-10, PReLU obtained a minimum median error rate of 5.36% compared to 5.41% for ReLU. On

Table 6.2: ResNet 110 layers test error (in %) on both CIFAR-10 and CIFAR-100 datasets. We report the mean error over the last five epochs and the minimum error over all epochs (inside parenthesis) of the median error over five tries. The results show that BN before ELU and PELU is detrimental, although it is less harmful before our PELU.

ACT	CIFAR-10	CIFAR-100
PELU	5.51 (5.36)	25.02 (24.55)
BN-PELU	6.24 (5.85)	26.04 (25.38)
ELU	6.55 (5.99)	26.59 (25.08)
BN-ELU	11.20 (10.39)	35.51 (34.75)

CIFAR-100, PReLU obtained a minimum median error rate of 25.83% compared to 25.92% for ReLU. PReLU obtained the same minimum median error rate than PELU on CIFAR-10, but it was significantly higher on CIFAR-100. Note that our main contribution is showing performance improvement over ELU with our parameterization, and that we only add PReLU as an additional reference. Nonetheless, we observe that our PELU parameterization of ELU obtains higher relative improvements than the PReLU parameterization of ReLU.

6.3.3 Understanding the effect of Batch Normalization

In this section, we present our experiments on the effect of Batch Normalization. As we explained in Section 6.3.1, we did not use BN before ELU because Clevert et al. (2016) observed that BN before ELU was detrimental for performance. Thus, we removed BN before PELU since PELU is based on ELU. In this section, we present experimental evidence that BN before PELU is also detrimental, just as it is before ELU.

To this effect, we trained a 110-layer ResNet using BN before ELU and PELU following the same experimental framework of Section 6.3.2. We used the same hyper-parameters and network structure, to the difference that we added BN before ELU and PELU in the ACT module. Note that in all cases, we keep BN after the second convolutional layer, as we showed in Figure 6.4.

The results of this experiment are reported in both Figure 6.6 and Table 6.2. The plots in Figure 6.6 represent the median test error over five tries at each epoch on CIFAR-10 and CIFAR-100 tasks. The curves in blue (circle marker) refer to ELU and PELU without BN, and the curves in green (downward triangle marker) refer to ELU and PELU with BN before. The numerical values of their performance are reported

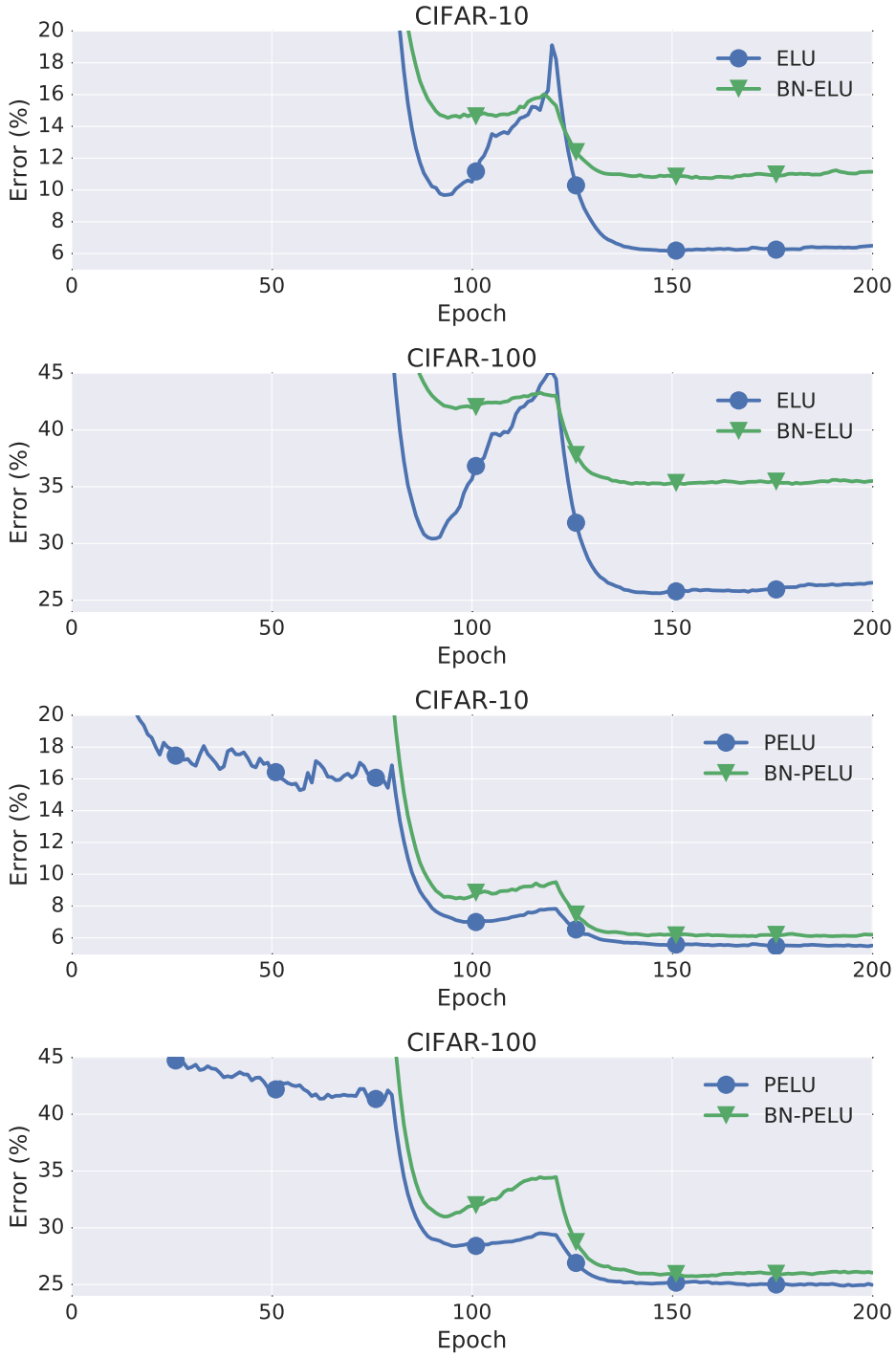


Figure 6.6: Effect of using BN before ELU (first two rows) and PELU (last two rows) activations in a ResNet with 110 layers on both CIFAR-10 and CIFAR-100 datasets. We show the convergence behavior of the median test error over five tries. In both cases, BN worsen performance of ELU and PELU. Note that we still use BN after the second conv layer, as seen in Figure 6.4.

in Table 6.2. In each cell of Table 6.1, the value on the left corresponds to the mean error over the last five epochs of the median error over five tries, while the value inside parenthesis corresponds to the minimum over all epochs of the median error over five tries.

The results show a large error rate increase on both CIFAR-10 and CIFAR-100 datasets for both ELU and PELU activations. On CIFAR-10, the minimum median test error for ELU increases from 5.99% without BN to 10.39% with BN, while for PELU it increases from 5.36% without BN to 5.85% with BN. On CIFAR-100, the minimum median test error for ELU increases from 25.08% without BN to 34.75% with BN, while for PELU it increases from 24.55% without BN to 25.38% with BN. As previously observed (Clevert et al., 2016), BN before ELU is detrimental, and as we expected, BN before PELU is also detrimental.

It is worth noting that the relative error rate increase of BN before our PELU is smaller than that of BN before ELU. On CIFAR-10, ELU has a relative error rate increase of 73%, while PELU is 9%. On CIFAR-100, ELU has a relative error rate increase of 39%, while PELU is 3%. These results show that our parameterization of ELU can reduce the harmful effects of BN before PELU. This was an unexpected result, since we did not design our parameterization to have this effect. Finally, note that BN can still be used when choosing ELU or PELU, but it should appear at a different location than before the activation. For instance, we applied BN after the second convolutional layer in the basic block structure (as shown in Figure 6.4), and it was not detrimental.

6.3.4 ImageNet Object Recognition

We performed a second object recognition experiment, this time on the ImageNet 2012 task (ILSVRC 2012) (Russakovsky et al., 2015), in order to evaluate our approach on a large-scale task. To this effect, we trained four different networks: ResNet18 (Shah et al., 2016), Network in Network (NiN) (Lin et al., 2013), All-CNN (Springenberg et al., 2015) and Overfeat (Sermanet et al., 2014). The block structure of ResNet18 is the same as the one we used in Section 6.3.2, but with a different transition block. As shown in Figure 6.4, the transition block is defined as a convolutional layer with 1×1-dimensional filters, a stride of two, and twice the number of filters, followed by batch normalization.

Again, in order not to favor PELU to the detriment of the other activations, we performed minimal changes to the networks. We only replace the activation function, which

Table 6.3: ImageNet training regimes for modifying the learning rate and the weight decay. We trained NiN and ResNet18 using the first one, and trained All-CNN and Overfeat using the second one. The first regime starts at a higher learning rate, but has a larger decay than the second regime.

Regime #1 (ResNet18, NiN)				
Epoch	1	10	20	25
Learning Rate	1e-1	1e-2	1e-3	1e-4
Weight Decay	5e-4	5e-4	0	0

Regime #2 (Overfeat, All-CNN)					
Epoch	1	19	30	44	53
Learning Rate	1e-2	5e-3	1e-3	5e-4	1e-4
Weight Decay	5e-4	5e-4	0	0	0

can be either ELU alone, PELU alone or BN-ReLU. Note that we still use BN after the second convolutional layer in the basic block structure, as we did in Section 6.3.2.

We trained each network with momentum stochastic gradient descent following the training regimes shown in Table 6.3. The table on the top of Table 6.3 corresponds to the training regime of ResNet18 and NiN, while the table on bottom corresponds to the training regime of Overfeat and All-CNN. The main difference between both regimes is that regime #1 starts at a higher learning rate than regime #2, and has a higher learning rate decay. We used the Torch implementation of Chintala from imagenet-multiGPU.torch [1] to perform our experiments.

The results of this experiment are reported in Figure 6.7. The plots represent the test error rate (in %) of one try for each epoch. The curves in blue (circle marker) refer to BN-ReLU, the curves in green (downward triangle marker) refer to ELU, while the curves in red (square marker) refer to PELU. The image on the top left corresponds to NiN, the image on the top right corresponds to ResNet18, the image on the bottom left corresponds to All-CNN, while the image on the bottom right corresponds to Overfeat.

The results of Figure 6.7 show that the networks using PELU outperformed the networks using ELU. Overall, NiN obtained the best result with 36.06% error rate with PELU, compared to 40.40% with ELU. This corresponds to a relative improvement of 7.29% for PELU over ELU. Interestingly, PELU obtained this relative improvement by

[1] https://github.com/soumith/imagenet-multiGPU.torch

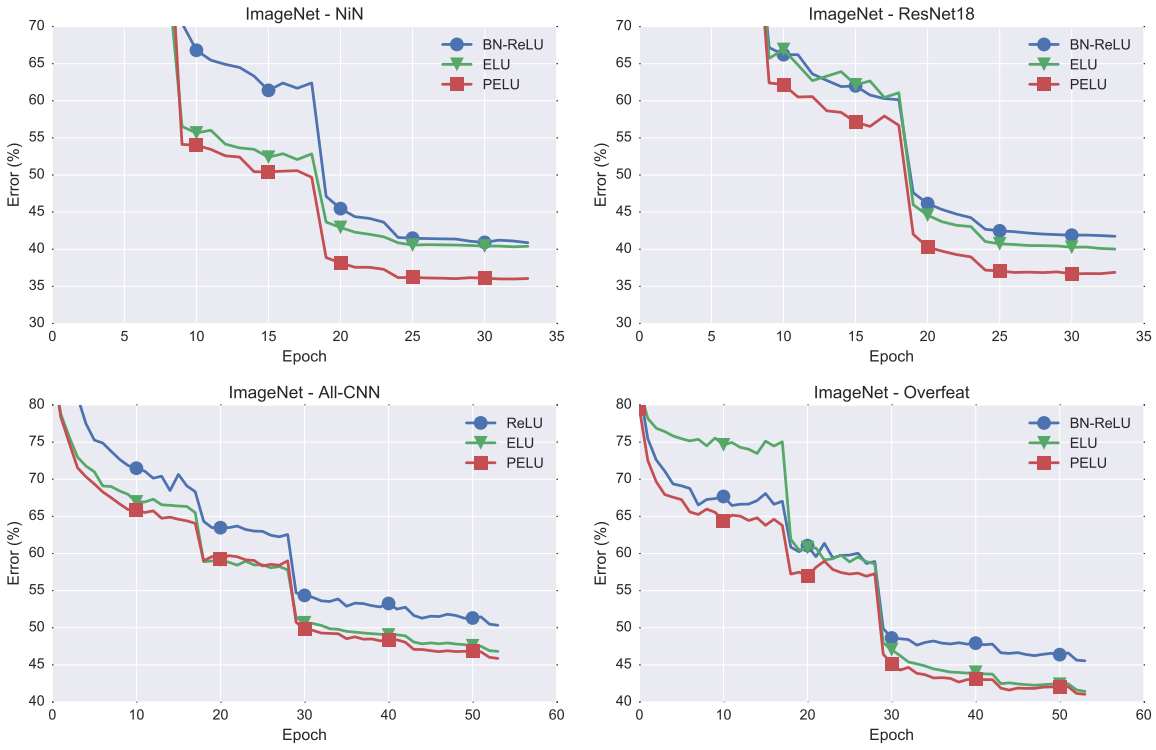


Figure 6.7: Test error rate progression (in %) of ResNet18, NiN, Overfeat and All-CNN on ImageNet 2012. NiN and ResNet18 (top row) used training regime #1, while All-CNN and Overfeat (bottom row) used training regime #2 (see Table 6.3). PELU has the lowest error rates for all networks. Regime #1 shows a greater performance gap between ELU and PELU than regime #2.

only adding 24 additional parameters (NiN uses 12 ACT modules). This corresponds to an increase in the number of parameters of 0.0003% when using PELU over ELU. Intuitively, we would not have observed a relative improvement of 7.29% if we had simply adding 24 additional weights throughout the network. Such a low number of weights cannot significantly increase the expressive power of the network to have this effect. These results rather show that the parameters of PELU affect the network in a different way than the standard weights and biases of the linear transformations.

The training regimes have an interesting effect on the convergence of the networks. As shown in Figure 6.7, the networks trained with regime #1 (All-CNN and Overfeat) have similar performance when using ELU or PELU, while the networks trained with regime #2 (ResNet18 and NiN) have a significantly different performance when using ELU or PELU. Also, the convergence behavior of the networks trained with regime #1 is not as steady as the networks trained with regime #2. We observe a small error rate increase starting at epoch 44 with All-CNN and Overfeat using PELU, even though the

Table 6.4: ResNet 110 layers test error (in %) on both CIFAR-10 and CIFAR-100 datasets. We report the mean error over the last five epochs and the minimum error over all epochs (inside parenthesis) of the median error over five tries. We compare each four PELU configurations. Our proposed PELU configuration $(a, 1/b)$ obtained the best performance.

Configuration	CIFAR-10	CIFAR-100
$(a, 1/b)$	**5.51 (5.36)**	**25.02 (24.55)**
$(1/a, 1/b)$	5.73 (5.60)	25.68 (25.17)
$(1/a, b)$	6.51 (6.00)	26.33 (25.48)
(a, b)	6.74 (6.12)	26.20 (25.24)

error rate is steady for ELU and ReLU. These results suggest using a training regime with a larger initial learning rate and learning rate decay (like regime #1) when using PELU.

6.3.5 Experimenting with Parameter Configuration

In this section, we present our experiments on the effect of learning the inverse of each parameter. The proposed PELU activation function defined in Eq. (6.2.5) has two parameters a and b, where we multiply by a and divide by b. A priori, any of the four configurations (a, b), $(a, 1/b)$, $(1/a, b)$ or $(1/a, 1/b)$ could be used as parameterization. For instance, we could multiply both a and b, we could divide both a and b, or we could multiply one and divide the other.

Note that due to the use of weight decay regularization, the four configurations are not reciprocal. This is because weight decay has a different effect on PELU depending on the selected configuration for a and b. Recall that weight decay encourages the weights to have a low magnitude. In the case of PELU, it encourages both parameters a and b to be closed to zero. When choosing configuration $(1/a, 1/b)$, weight decay encourages the PELUs to have a steep slope $1/(ab)$. But when choosing configuration (a, b), weight decay encourages the PELUs to have a low slope ab. The experiment we conduct in this section will allow us to better understand the difference between the four configurations. To this end, we performed an experimental evaluation on the CIFAR-10 and CIFAR-100 tasks using ResNet-110 following the same experimental framework that we detailed in Section 6.3.2.

The results of this experiment are presented in both Figure 6.8 and Table 6.4. The plots in Figure 6.8 represents the median test error over five tries at each epoch. The

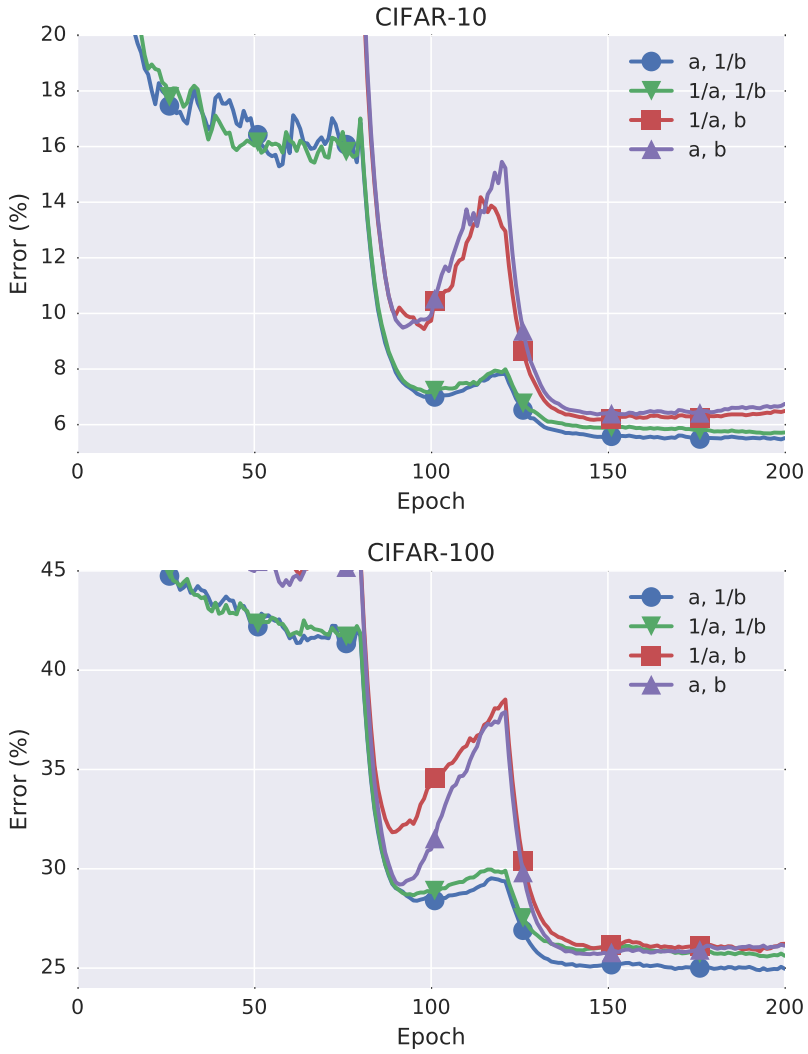


Figure 6.8: Experimenting with the PELU parameter configuration in a ResNet with 110 layers on both CIFAR-10 and CIFAR-100 datasets. We show the convergence behavior of the median test error over five tries. Our proposed parameterization $(a, 1/b)$ obtained the best performance.

curves in blue (circle marker) refer to the proposed configuration $(a, 1/b)$, the curves in green (downward triangle marker) refer to configuration $(1/a, 1/b)$, the curves in red refer to configuration $(1/a, b)$, while the curves in purple (upward triangle) refer to configuration (a, b). The figure on the top refer to the CIFAR-10 task, while the figure on the bottom refer to the CIFAR-100 task. The numerical values of their performance are reported in Table 6.4. In each cell of Table 6.4, the value on the left corresponds to the mean error over the last five epochs of the median error over five tries, while the value inside parenthesis corresponds to the minimum over all epochs of the median error over five tries.

As we can see in Table 6.4, our proposed parameterization $(a, 1/b)$ obtained the best performance overall. On CIFAR-10, the proposed configuration $(a, 1/b)$ obtained a minimum test error median of 5.36%, while configuration $(1/a, 1/b)$ obtained 5.60%, configuration $(1/a, b)$ obtained 6.00% and configuration (a, b) obtained 6.12%. On CIFAR-100, the proposed configuration $(a, 1/b)$ obtained a minimum test error median of 24.55%, while configuration $(1/a, 1/b)$ obtained 25.17%, configuration $(1/a, b)$ obtained 25.48% and configuration (a, b) obtained 25.24%.

The plots in Figure 6.8 show that the proposed configuration $(a, 1/b)$ obtained the best convergence behavior. We can see that the two configurations using b exhibit a large error rate increase during the second phase of the training process, while the other two configurations using $1/b$ have a steady convergence. Also, the two configurations using $1/b$ obtained a significantly lower error rate than the two configurations using b.

These results concur with our observation in Section 6.2.3 on the effect of weight decay on the parameters of PELU. As we explained, weight decay encourages PELU to be similar to ReLU when using configuration $(a, 1/b)$. When both parameters a and b are closed to zero, the PELUs acts as ReLUs but the network do not suffer from a weight decay loss. This helps the network to use as much PELUs that look like ReLUs as needed without incurring a large penalty. This is important because there can be an incentive to use ReLU in order to have sparse activations that could be impaired with a large weight decay loss. In the next section, we provide additional experimental evidence to support that claim.

6.3.6 Parameter Progression

In this section, we present our experiment on the progression of the PELU parameters during training. The results of previous section has shown that configuration $(a, 1/b)$ gives the best performance among the four possible configurations. We argued that configuration $(a, 1/b)$ gave the best performance because PELU does not suffer a weight decay loss when converging to a shape that looks like ReLU. Therefore, we expect several PELUs to look like ReLU after convergence, which we can show with a visual inspection of the shapes after training.

To do so, we trained a VGG network (Simonyan and Zisserman, 2014) on the CIFAR-10 task following the experimental framework that we detailed in Section 6.3.2. The results of this experiment are shown in Figure 6.9. The plots show the value of the parameters at each step for each 14 activations throughout the network. Activation at layer 1 is

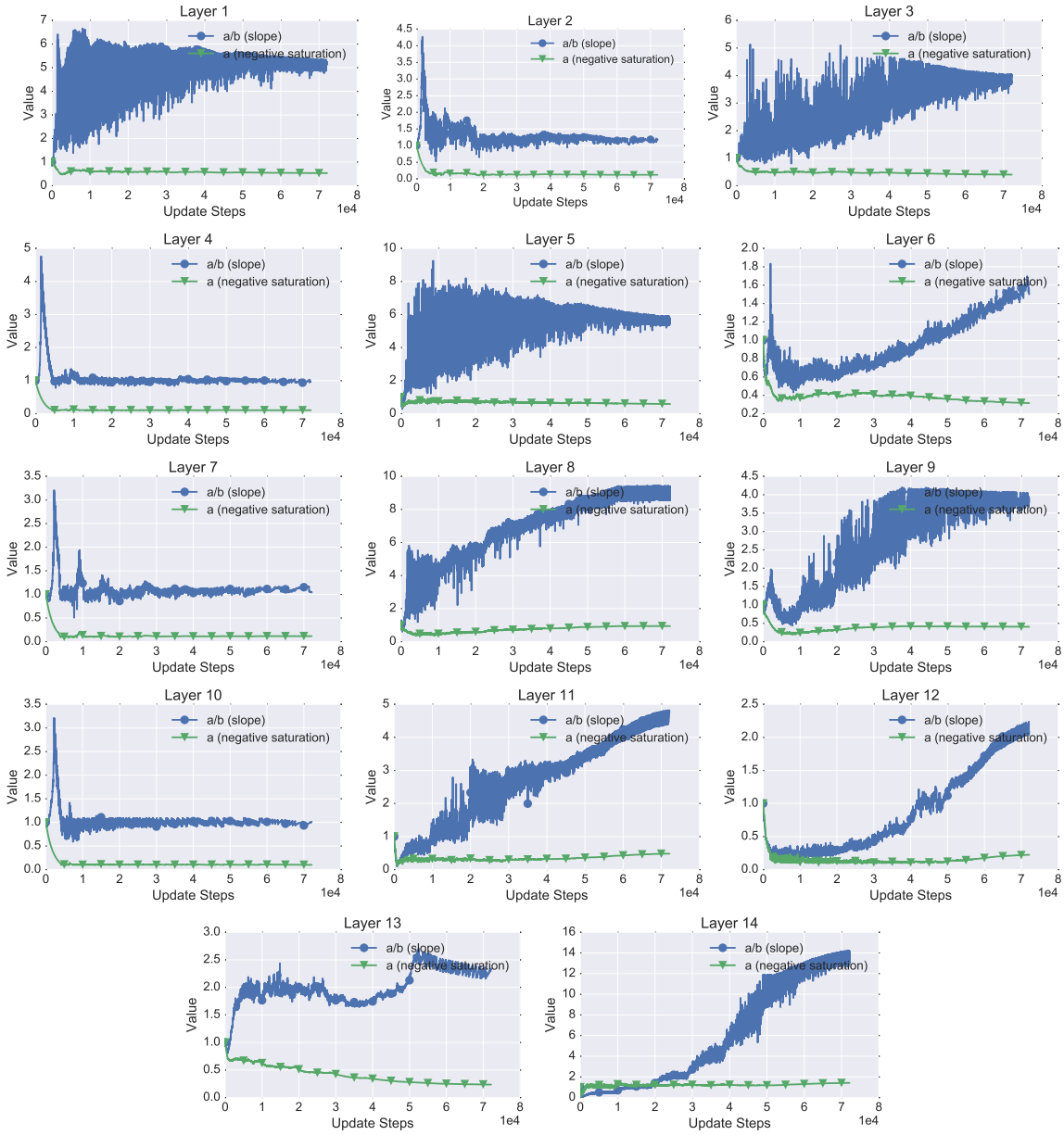


Figure 6.9: PELU parameter progression at each layer of VGG trained on CIFAR-10. We present the variation of the slope ($\frac{a}{b}$) and the negative saturation (parameter a). The network adopted different nonlinear behaviors throughout the training phase.

the closest to the input, while activation at layer 14 is the closest to the output. The curves in blue (circle marker) represent the slope a/b of the positive part, while the curves in green (downward triangle marker) correspond to a, the negative of the point of saturation of the negative part.

The results in Figure 6.9 illustrate two different behaviors. The first one is seen at layers 2, 4, 7 and 10. We can observe that the slope increased to a large value, then decreased and converged to a value close to 1. We can also observe that parameter a quickly converged to a value near 0. A slope value close to 1 and a negative saturation close to 0 indicates that the activations have converged to a shape that looks similar to ReLU. This result constitutes an empirical evidence that ReLU is an important activation function. It was found to be optimal by several PELUs that could have converged to other parametric forms. It is understandable that there is an incentive to converge to ReLU because we know that ReLU has the important effect of promoting activation sparsity (Nair and Hinton, 2010; Maas et al., 2013) (ref. Section 3.2.2).

The second behavior that we observe from the results in Figure 6.9 is seen at all other layers (apart from 2, 4, 7 and 10). We can observe that the negative value of the saturation converged to other than zero. At layer 1, it converged to a value near 0.5, while it converged to a value near 2 at layer 14. A negative saturation different from zero indicates that the activations have favored outputting negative values for negative arguments, like the standard ELU. It was found to be optimal in most of the PELUs, even though it was possible to converge to a saturation value near zero like the ReLU. This result constitutes an empirical evidence that having a negative saturation is important, which corroborates with our understanding of the problem of bias shift (Clevert et al., 2016).

6.4 Discussion

As we explained in previous sections, we did not use Batch Normalization (BN) during our experiments with ELU and PELU. This was due to the detrimental effect of preceding PELU and ELU with BN, as we observed in Section 6.3.3. This detrimental effect was also previously observed with the ELU (Clevert et al., 2016), but it remains unclear why it happens.

One avenue is to look at the difference between the properties of ELU and the properties of ReLU. An important one that we can observe is that ReLU is positively scale invariant

and ELU is not. Indeed, for ReLU we have $\max\{0, kx\} = k \max\{0, x\}$, where $k \geq 0$, while for ELU, which can be expressed as $\max\{0, x\} + \min\{0, \exp(x) - 1\}$, we have $\min\{0, \exp(kx) - 1\} \neq k \min\{0, \exp(x) - 1\}$.

The fact that ReLU is positively scale invariant and ELU is not may be part of the reason why BN has positive effects before ReLU but negative effects before ELU. Using a positively scale invariant activation function may be essential for BN to properly reduce internal covariate shift (Ioffe and Szegedy, 2015) through mean and standard deviation scaling. We could validate this hypothesis by experimenting with a new positively scale invariant activation function and observe whether BN helps or not.

6.5 Conclusion

The activation function is a central element in deep learning. Its main purpose is to break the chain of linear dependencies between subsequent linear transformations by introducing nonlinearity. This allows the creation of deep neural networks that are nonlinear functions of their input by sequentially applying a linear transformation followed by a nonlinear activation function. Parametric activation functions have been previously shown to help training. They have parameters controlling different aspects of their shape that can be learned as training progresses, which provide additional flexibility during training.

In this chapter, we presented our parameterization of the Exponential Linear Unit (ELU), which we refer to as PELU. We defined parameters to control different aspect of the shape of ELU, but constrained the parameterization to preserve differentiability at zero. We performed several experiments to evaluate our activation function. For example, we studied auto-encoding in unsupervised learning and object recognition in supervised learning in order to show the advantage of our parameterization. The overall results showed that the performance improved with our parameterization over using standard ELU.

We also performed three sets of experiments to better understand the behavior of our parameterization. First, we looked at the effect of using Batch Normalization (BN) before the activation. The results showed that BN before PELU was detrimental, but was not as detrimental as before ELU. We suggested that it may be related to ELU not being positively scale invariant. Second, we experimented with learning the inverse of each parameter of our parameterization. The results showed that the proposed one was

the best over all four possible configurations. The intuition is that the network does not suffer a weight decay when PELU converge to a shape that looks like ReLU. And third, we looked at the parameter progression of all PELUs in a VGG network. The results showed that some PELUs converged to a shape that looked like ReLU, favoring sparsity, while others converged to a negative saturation, favoring bias shift.

In conclusion, our work in this chapter has shown that parameterizing a fixed activation can improve performance. It goes in line with previous works showing the advantage of parametric activation functions and supports research efforts in this avenue.

Chapter 7

Deep Learning Development: Multitask Learning

In this chapter, we present our contribution on soft parameter sharing in Multitask Learning (MTL). We propose a sharing mechanism defined as a trainable component of the network based on nonlinear task relations, following the recent advances in residual learning. We refer to our component as the Collaborative Block. We also perform an experimental evaluation on the problem of facial landmark detection in a MTL settings, an ablation study to show experimental evidence of domain information sharing and investigate the potential of mixing hard and soft parameter sharing.

7.1 Introduction

Multitask Learning (MTL) is a field of machine learning that has a long-standing literature (Caruana, 1997; Evgeniou and Pontil, 2004; Argyriou et al., 2007; Liu et al., 2009; Zhang et al., 2012, 2014). Approaches based on deep neural networks for MTL usually fall into the following two categories: hard parameter sharing and soft parameter sharing (Ruder, 2017). Soft parameter sharing is gaining in popularity over hard parameter sharing, due to the sharing mechanism that can provide more flexibility on how information sharing is implemented. We reviewed in Section 4.2 approaches that implement their sharing mechanism as either a regularization term or as a trainable component of the network. The approaches based on a trainable component had however a limitation. We saw that the sharing mechanism of the cross-stitch block (Misra et al., 2016), the sluice block (Ruder et al., 2017) and the cross-residual block (Jou and Chang, 2016) were all implemented as trainable linear combinations between task-specific feature rep-

resentations. As we explained, they are limited to linear task relations, which are more limited than nonlinear task relations and can constrain information sharing between the tasks.

Our goal in this chapter is to address the issue of the sharing mechanism being constrained to linear relations. To this effect, we present a novel approach based on nonlinear relations. We implement the sharing mechanism in terms of the Collaborative Block that uses two nonlinear transformations with lateral connections. The first transformation aggregates all task-specific feature representations into a global feature representation, and the second transformation merges back the global feature representation with each task-specific feature representation. Our transformations have the particularity of containing skip connections with additions and concatenations. As we saw in Section 3.4, skip connections enable residual learning and have many advantages over standard connections. Our collaborative block is defined as a trainable component of the network, so its parameters can be learned during training as the other components.

To evaluate the performance of our approach, we opted for an experimental evaluation on the problem of Facial Landmark Detection (FLD) in a MTL framework (Sun et al., 2013; Zhang et al., 2016; Jourabloo and Liu, 2016; Baltrušaitis et al., 2016). FLD can be described as follows: given the image of a face of a person, the goal is to predict the (x, y)-position of specific landmarks associated with key features of the face. Examples of facial landmarks include the center of the eyes, the corners of the mouth, the tip of the nose and the earlobes. Approaches that deal with this problem are essential components in many face-related tasks. For instance, face recognition (Ding and Tao, 2015), face validation (Taigman et al., 2014), facial feature detection and tacking (Zhang and Zhang, 2014) all rely on the ability to correctly find the location of these distinct facial landmarks in order to succeed.

Localizing facial landmarks is a challenging problem. The presence of many lighting conditions, head poses, facial expressions and occlusions creates diversity in the face images, which increases difficulty. A FLD approach must also take into account a certain number of correlated factors during the estimation process that further increases difficulty. For instance, an angry person will display pinched lips while a sad person will display sunken mouth corners, but both of them will have frowned eyebrows (Fabian Benitez-Quiroz et al., 2016). The presence of frowned eyebrows is correlated with both the presence of pinched lips and sunken mouth corners, but the presence of frowned eyebrows alone cannot be used to discriminate between an angry and a sad person.

Moreover, an interesting particularity of datasets geared towards FLD is that they are well-suited for MTL. In addition to containing the position of the facial landmarks, these datasets contain a number of other labels that can be used to define auxiliary tasks. For instance, tasks such as gender recognition, smile recognition, glasses recognition or face orientation are often used as auxiliary tasks to define the MTL training framework. For these reason, FLD is an excellent problem to evaluate novel advances in MTL.

Here is a summary of the extent of our work on our collaborative block:

1. We propose a sharing mechanism that addresses the limitation of previous approaches of being limited to linear task relations. Our sharing mechanism implements information sharing with two nonlinear transformations based on skip connections, in order to exploit the full potential of nonlinear relations.

2. We perform an experimental evaluation on the problem of facial landmark detection. We compare our approach to the others of the state of the art on the datasets Multitask Facial Landmark (MTFL) (Zhang et al., 2014), Annotated Facial Landmarks in the Wild (AFLW)(Koestinger et al., 2011), LFWNet (Sun et al., 2013), CelebA (Liu et al., 2015b) and WIDER (Yang et al., 2016) using AlexNet (Krizhevsky et al., 2012) and ResNet (He et al., 2016b) as underlying networks. Overall, the results show that our collaborative block obtains better performances over the other state-of-the-art sharing mechanisms.

3. We evaluate the effect of our collaborative block in a context of data scarcity. Overall, the results show that our approach outperforms single-task learning, which indicates that our approach can leverage the training signals of related tasks in order to improve generalization.

4. We further illustrate an instance of knowledge sharing between the task-specific networks with an ablation study. We observed that the *facial landmark* network relied on the high-level feature representations of the *face profile* network. These results corroborate with the well-known fact that the position of facial landmarks is directly dependent on face orientation and constitute an empirical evidence of information sharing between the task-specific networks.

5. We finally explore the idea of mixing hard parameter sharing with soft parameter sharing. We obtained performance improvement with a hard parameter sharing

part that spans the first half of the underlying network and a soft parameter part the spans the second half of the underlying network.

The content of this chapter is organized as follows. In Section 7.2, we elaborate on our collaborative block. We start with a reminder of important soft parameter sharing approaches, then introduce ours and present an instantiation of a soft parameter sharing network using ResNet as underlying network. In Section 7.3, we present our experimental evaluation on facial landmark detection. We start with a description of the problem and of our experimental framework. We then present our results on standard facial landmark detection datasets, such as MTFL (Zhang et al., 2014), AFLW (Koestinger et al., 2011), LFWNet (Sun et al., 2013), CelebA (Liu et al., 2015b) and WIDER (Yang et al., 2016). We used AlexNet and ResNet as underlying networks. We further present our results on our data scarcity experiment, on our ablative study to illustrate knowledge sharing and on mixing hard parameter sharing with soft parameter sharing. We finally conclude this chapter in Section 7.4.

7.2 Model

In this section, we present our contribution in soft parameter sharing. We start with a reminder of previous soft parameter sharing approaches, and refer the reader to Section 4.2 for more details. Then, we elaborate on our proposed novel sharing mechanism.

7.2.1 Soft Parameter Sharing

Over the past few years, soft parameter sharing has been growing in popularity over hard parameter sharing (Ruder, 2017). The idea of the sharing mechanism is to connect all T task-specific Convolutional Neural Networks (CNNs) together in order to share all information with every CNN. The sharing mechanism can connect the CNNs together by defining a component as a block with a lateral transformation. The lateral transformation takes as input all task-specific feature representations (at a certain depth) and combines them together. It then outputs a new feature representation for each task-specific CNN that incorporates information from all tasks.

The kind of information sharing that occurs in the sharing mechanism depends on the implementation of its forward pass. Let x_t, where $t \in \{1 \dots T\}$, be defined as the feature map for the CNN of task t at a certain depth. Without loss of generality, we drop the

depth index of feature map x_t. The forward pass takes as input all task-specific feature maps x_t and processes them into new feature maps y_t.

We have presented in Section 4.2 several sharing mechanisms defined as a block with a lateral transformation. One approach was cross-stitch (Misra et al., 2016), which defines its forward pass as follows:

$$y_t = \sum_{k=1}^{T} \alpha_{t,k} x_k \,, \tag{7.2.1}$$

where x_k are the input feature maps for task k, y_t are the output feature maps for task t, T is the total number of task and $\alpha_{t,k}$ are the weights of the linear combinations between task t and task k. In other words, the cross-stitch block implements information sharing by computing a linear combination of the task-specific feature maps x_t. The weights $\alpha_{t,k}$ determine the amount of task-specific information that is shared among each CNN and are learned during training.

The cross-residual block (Jou and Chang, 2016) is an extension of the cross-stitch block based on the recent advances in residual learning (He et al., 2016b,a). The cross-residual block defines its forward pass as follows:

$$y_t = \mathcal{F}_t(x_t) + \sum_{t=1}^{T} x_t \,, \tag{7.2.2}$$

where $\mathcal{F}_t$ is a transformation known as the residual mapping.

In our experimental evaluation in Section 7.3, we opted for the *original* residual mapping $\mathcal{F}$ instead of the *full pre-activation* (He et al., 2016b). The original residual mapping ends with Batch Normalization (Ioffe and Szegedy, 2015), is then added to the input and finally forwarded to ReLU (He et al., 2016a). We opted for this mapping because we used the PyTorch (Paszke et al., 2017) implementation of ResNet that comes with a version pre-trained on ImageNet (Russakovsky et al., 2015). As a result, we use an additional ReLU after the summation in Eq. (7.2.2). The forward pass of our implementation of the cross-residual block is thus defined as follows:

$$y_t = ReLU(\mathcal{F}_t(x_t) + \sum_{t=1}^{T} x_t) \,. \tag{7.2.3}$$

The cross-residual block can be seen as extending the standard forward pass of the residual block $ReLU(\mathcal{F}_t(x_t) + x_t)$ with a sharing mechanism based on a sum. It uses T identity skip connections, one for each task, and connects the feature maps together

by summing them together and adding them to the output of the residual mapping $\mathcal{F}$. Thus, the cross residual block incorporates all the advantages of residual learning (ref. Section 3.4.7).

The sluice block (Ruder et al., 2017) is another extension of the cross-stitch block that is also based on linear combinations. It takes a step further by defining another set of linear combinations on layer-specific feature maps. Assuming that $x_{t,l}$ is the feature representation of task t computed at layer l, the forward pass is defined as follows:

$$y_t = \sum_{l=1}^{L} \beta_{t,l} x_{t,l} \,, \tag{7.2.4}$$

where $\beta_{t,l}$ are the weights of the linear combination. The output value y_t is then directly given as input to the last layer, which is usually the softmax for classification. Note that the sluice block also uses linear combinations between the task-specific feature maps in the same way as the cross-stitch block of Eq. (7.2.1).

The main drawback of all the previous approaches is that the sharing mechanisms are defined as linear combinations, which limits the expressiveness of the information shared between the task-specific networks. In the next section, we present our proposed collaborative block that addresses this limitation by defining the sharing mechanism with nonlinear transformations.

7.2.2 Collaborative Block

The forward pass of our *collaborative block* is defined using two distinct nonlinear transformations. One aggregates task-specific features into global features, and the other merges back these global features into each task-specific CNN. Using the notation introduced in the previous section, our approach is defined as follows:

$$z = \mathcal{H}([x_1, \ldots, x_T]) \,, \qquad\qquad y_t = ReLU\left(x_t + \mathcal{F}_t([x_t, z])\right) \,, \tag{7.2.5}$$

where $\mathcal{H}$ is the central aggregation, $\mathcal{F}_t$ are the task-specific aggregations and $[x_t, z]$ denotes depth-wise concatenation between feature maps x_t and z. The goal of $\mathcal{H}$ is to combine all task-specific feature maps x_t into a global feature map z representing unified knowledge, while the goal of $\mathcal{F}$ is to merge back the global feature map z with each task-specific input x_t.

The use of an additive skip connection during the computation of y_t with $\mathcal{F}_t$ allows to learn the identity mapping by simply pushing the parameters of $\mathcal{F}_t$ to zero. In this case,

it makes the output value y_t equal to x_t, which removes the contribution of the global features z. Note that z is computed the same way, regardless of whether $\mathcal{F}_t$ outputs zero or not. This provides additional flexibility on how to implement domain information sharing, because the network can more easily deal with the case where integrating z back into the task-specific features x_t would not be helpful. In other words, when blending the domain information would be detrimental.

The reason why it is important to give this flexibility comes from the fact that depth influences the relevance of each task towards another (Misra et al., 2016; Jou and Chang, 2016; Ruder et al., 2017; Lu et al., 2017; Ranjan et al., 2017). Some task-specific CNNs benefit more when they share their low-level features, while others benefit more when they share their high-level features (Ranjan et al., 2017). Our collaborative block can take into account this dependence on depth with the task-specific aggregations $\mathcal{F}_t$. The network can learn identity mappings for all tasks that do not benefit from sharing low-level features when the collaborative block appears at a low level, and similarly for high-level features. An example of such specialization will be shown in our ablative study in Section 7.3.4.

Our definition of the collaborative block does not impose any restriction on the structure of the central aggregation $\mathcal{H}$ and the task-specific aggregation $\mathcal{F}_t$. The only requirement is that they are nonlinear functions of their input. However, as we mentioned in the previous section, we used the PyTorch (Paszke et al., 2017) implementation of ResNet in our experimental evaluation in order to take advantage of the available pre-trained version on ImageNet (Russakovsky et al., 2015). We therefore chose a structure for the central aggregation $\mathcal{H}$ and for the task-specific aggregation $\mathcal{F}_t$ that is compatible with the residual block of the network. For this reason, we opted for the following one:

$$\mathcal{H}(\cdot) = (ReLU \circ BN \circ Conv_{(3\times3)} \circ ReLU \circ BN \circ Conv_{(1\times1)})(\cdot)\,, \tag{7.2.6}$$

$$\mathcal{F}(\cdot) = (BN \circ Conv_{(3\times3)} \circ ReLU \circ BN \circ Conv_{(1\times1)})(\cdot)\,, \tag{7.2.7}$$

where BN stands for Batch Normalization (Ioffe and Szegedy, 2015), $Conv_{(h\times w)}$ for a standard convolutional layer with filters of size $(h \times w)$, and $\circ$ is the usual function composition. The first $Conv_{(1\times1)}$ layer in $\mathcal{H}$ divides the number of feature maps by a factor of 4, while the first $Conv_{(1\times1)}$ layer in $\mathcal{F}$ divides it to match the size of x_t.

An illustration of our collaborative block for a soft parameter sharing network with two tasks is shown in Figure 7.1. The figure is divided into different parts in order to improve visibility. The input feature maps x_1 and x_2 of each task are shown in part ①. The central aggregation, shown in part ②, takes as input x_1 and x_2 and computes the

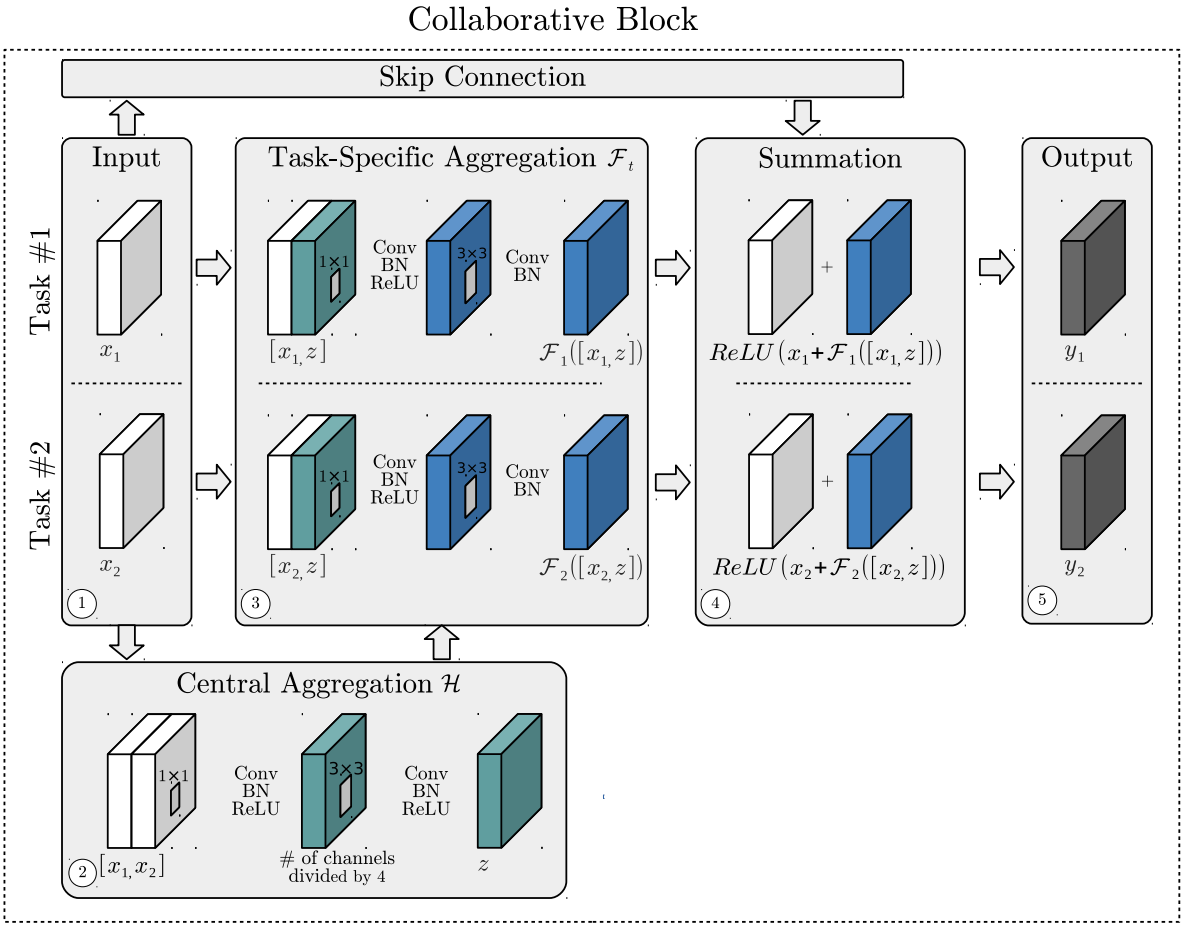


Figure 7.1: Example of our collaborative block applied on the feature maps of two task-specific networks. The input feature maps (shown in ①) are first concatenated depth-wise and transformed into a global feature map (②). The global feature map is then concatenated with each input feature map individually and transformed into task-specific feature maps (③). Each resulting feature map is then added back to the input feature map using a skip connection (④), which gives the final outputs of the block (⑤).

global feature map z (colored in green). Then, z is concatenated to each feature map x_1 and x_2 and forwarded to each task-specific aggregations $\mathcal{F}_1$ and $\mathcal{F}_2$. The output value of $\mathcal{F}_1$ and $\mathcal{F}_2$ are colored in blue and are shown in part ③. Each output value of $\mathcal{F}_1$ and $\mathcal{F}_2$ is then added to each input feature map x_1 and x_2 in part ④. We finally apply a last ReLU and obtain the final output values y_1 and y_2 of the collaborative block. The output values y_1 and y_2 are shown in part ⑤.

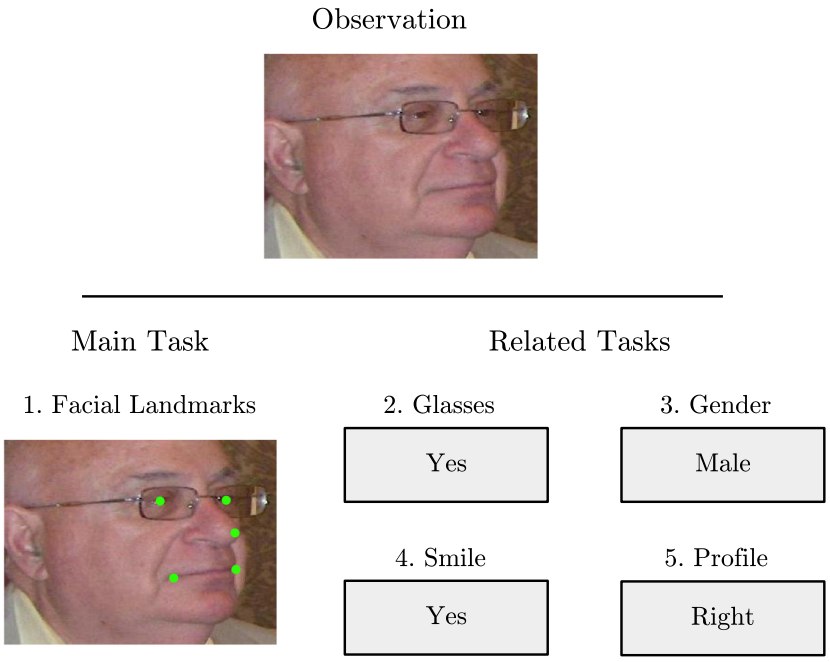


Figure 7.2: Example of FLD with four related tasks. The related tasks are glasses recognition, gender recognition, smile recognition and profile detection. The main task is to predict five facial landmarks.

7.3 Experiments

In this section, we detail our Multitask Learning (MTL) training framework and present our experiments in Facial Landmark Detection (FLD). We further evaluate the effect of data scarcity on performance, illustrate an example of knowledge sharing between task-specific CNNs with an ablation study and investigate mixing hard parameter sharing with soft parameter sharing.

7.3.1 Multitask Learning Experimental Framework

Problem Definition

The goal of Facial Landmark Detection (FLD) is to predict the (x, y)-position of specific landmarks associated with key features of the visage. While the number and type of landmarks are specific to each dataset, examples of standard landmarks to be predicted are the corners of the mouth, the earlobes, the center of the eyes and the tip of the nose. Each dataset further defines a number of related tasks in addition to the facial landmarks. These related tasks also vary from one dataset to another, but are typically gender recognition, smile recognition, glasses recognition or face orientation. An example of a FLD problem with related tasks is shown in Figure 7.2.

Training Framework

We define our training framework in such a way that we treat each task as a classification problem. We want them to use the same cross-entropy loss function to avoid loss imbalance and possible convergence problems between the tasks. For gender recognition, smile recognition and glasses recognition, this is straightforward because they are already classification tasks. However, it is different for face orientation and landmark detection. For face orientation, the goal is to predict the roll, yaw and pitch attributes of the orientation of the face. This problem is usually treated as a regression problem, since the goal is to predict real values. In our case, we instead create a classification problem from the regression problem. Instead of directly predicting the original real value of each attribute, we divide the range $[0, 360]$ degrees into bins and predict the label of the bin associated with the original real value. For instance, for 30 degrees-wide bins, the original face orientation problem is defined as a classification problem with 14 classes.

Similarly for FLD, the goal is to predict the (x, y) position of each landmark of the face. This problem is also usually treated as a regression problem, since the goal is to predict real values. In our case, we instead predict the row position and column position of the pixel where the landmark falls in. For instance, for a FLD task that requires to predict 5 landmarks, and for an input image with dimensionality 112×112, the original FLD problem is defined as 10 classification problems with 113 classes each (from 0 to 112).

Note that in all cases, we still use the original real values to report our results on the test set. For face orientation, we compare the ground truth value with the center value of the bin. For FLD, we compare the ground truth value with the row-column position of the pixel. Thus, we incorporate our approximation errors in the final score when comparing our prediction with the ground truth.

Landmark Failure Rate Metric

We report our results using the landmark failure rate metric (Zhang et al., 2014). The metric is defined as follows. First, compute the mean distance between the predicted landmarks and the ground truth landmarks. Then, compute the interocular distance from the center of the eyes. Finally, normalize the landmark mean distance by the interocular distance. A normalized mean distance greater than 10% is reported as a failure, while a normalized mean distance lower than or equal to 10% is reported as a success.

Underlying Network Architecture

In our experiments, we use two popular CNNs as underlying network: AlexNet (Krizhevsky et al., 2012) and ResNet (He et al., 2016a). As we mentioned in the previous section, we opted for the PyTorch (Paszke et al., 2017) implementation of both AlexNet and ResNet in order to take advantage of the available pre-trained version on ImageNet (Russakovsky et al., 2015). Furthermore, we apply a soft parameter sharing block at different locations in both networks. For AlexNet, we apply it after each max pooling layer, which gives a total of three soft parameter sharing blocks. For ResNet, we apply it after each series of residual blocks with a matching spatial dimensionality, which gives a total of five soft parameter sharing blocks.

7.3.2 Facial Landmark Detection on the MTFL Task

As a first experiment, we performed facial landmark detection on the Multitask Facial Landmark (MTFL) task (Zhang et al., 2014). The dataset contains 12,995 face images annotated with five facial landmarks and four related attributes of gender, smiling, wearing glasses and face profile. The face profile task is defined as predicting the yaw attribute of the orientation of the face, which is divided into five bins (five profiles in total). The training set has 10,000 images, while the test set has 2,995 images. We performed a total of four sets of experiments with a different underlying network: 1. AlexNet initialized randomly, 2. AlexNet pre-trained on ImageNet, 3. ResNet with 18 layers (ResNet18) initialized randomly, and 4. ResNet18 pre-trained on ImageNet.

We show an illustration of ResNet18 as underlying network for the MTFL task in Figure 7.3. The top part shows an overview of the network divided into a series of blocks interleaved with our collaborative blocks. The five rows correspond to the underlying network instantiated five times, one for each task. Each collaborative block receives as input the output of each task-specific block, processes them as detailed in Eq. (7.2.5), and sends the result back to each task-specific network. Note that adding any soft parameter sharing block to any underlying network can be done by following the same pattern of interleaving the network block structure with the soft parameter sharing block. Finally, the architecture of ResNet18, as well as the composition of its residual blocks, are shown at the bottom left, while the structure of each task-specific fully connected section is shown at the bottom right.

We first show examples of predictions with ResNet18 pre-trained on ImageNet as underlying network in Figure 7.4. The top part of the figure corresponds to the MTFL (Zhang

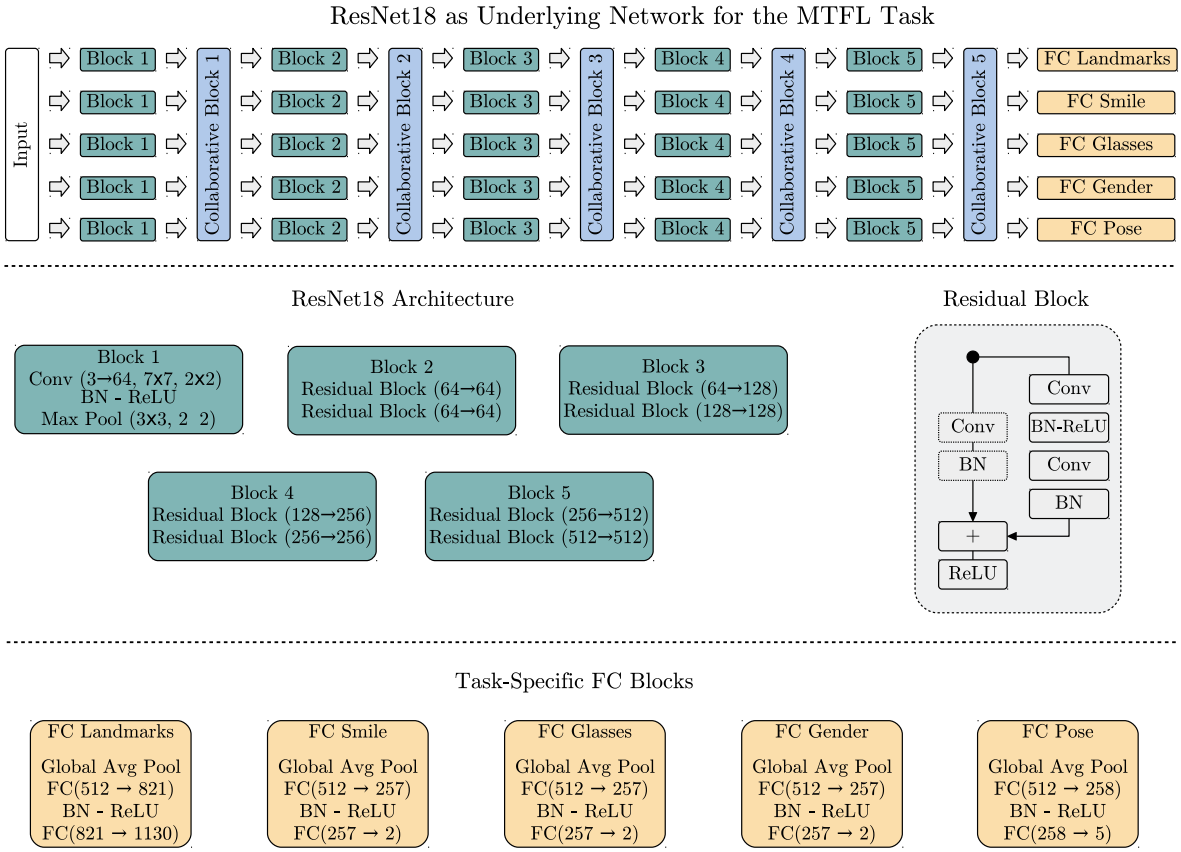


Figure 7.3: ResNet18 as underlying network for the MTFL task. The top part shows the block structure of ResNet18 interleaved with our proposed collaborative block, while the bottom part details each residual and task-specific FC block.

et al., 2014) task, while the bottom part corresponds to the AFLW (Koestinger et al., 2011) task (we will introduce the AFLW task in Section 7.3.3). For MTFL, the first two examples were reported as successes, while the last two examples as failures. The ground truth elements are colored in green and the network predictions are colored in blue. We also include the labels of the related tasks: gender, smiling, wearing glasses and face profile. As we can see, over-exposure can have a large impact on the quality of the predictions.

We then compared our approach to several other approaches of the state of the art. We included single-task learning (AN-S when using AlexNet as underlying network, RN-S when using ResNet18), hard parameter sharing MTL (AN and RN), widened hard parameter sharing MTL where the central section is widened to match the number of parameters of our approach (ANx and RNx), HyperFace (HF) (Ranjan et al., 2017), Tasks-Constrained Deep Convolutional Network (TCDCN) (Zhang et al., 2014), Cross-

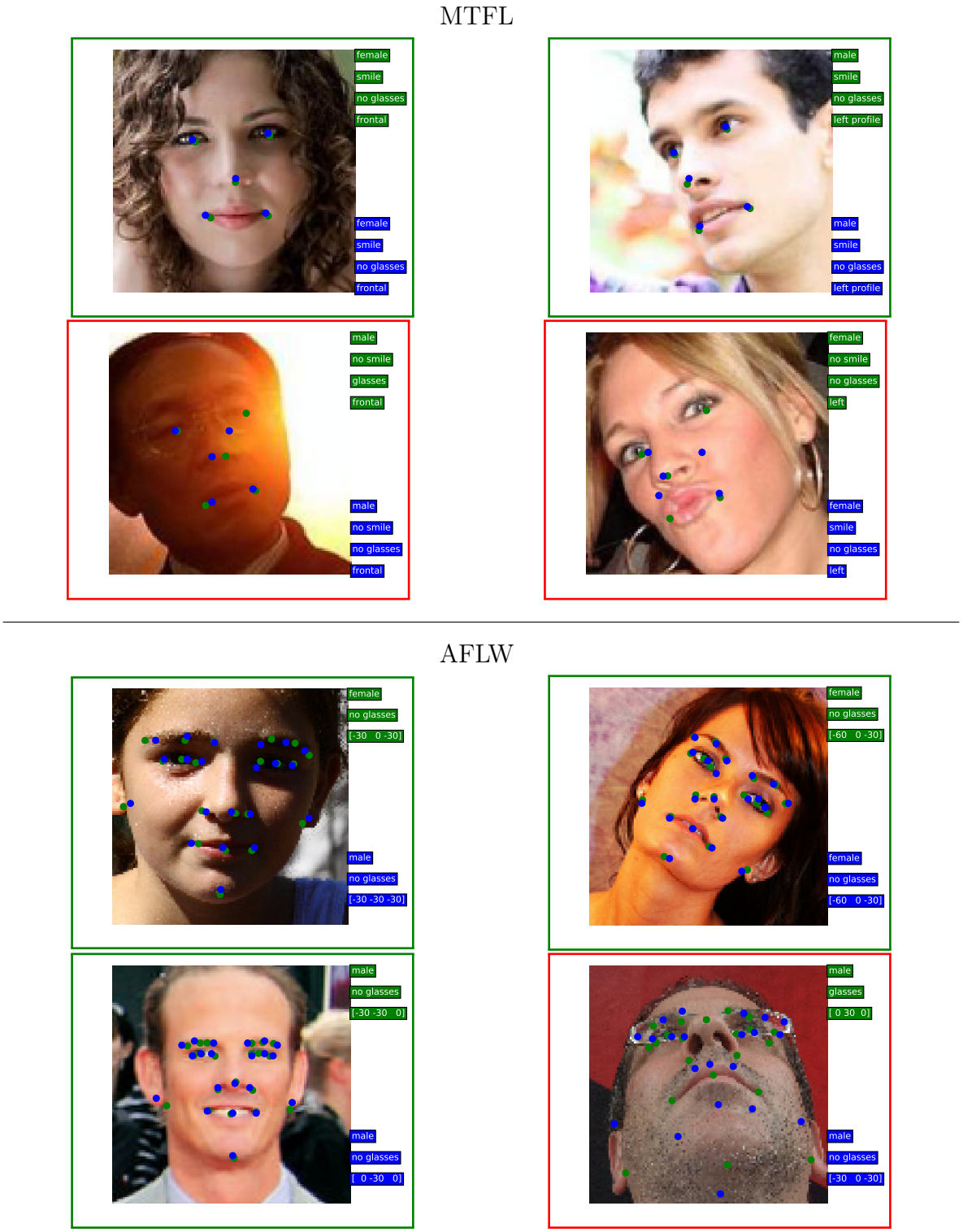


Figure 7.4: Example predictions with ResNet18 pre-trained on ImageNet as underlying network on the MTFL and AFLW tasks (better seen in color). For MTFL, the first two examples are successes, while last two are failure cases. For AFLW, the first three examples are successes, while the last one is a failure case. Elements in green correspond to ground truth, while those in blue correspond to predictions. Facial landmarks are shown as small dots, and related tasks labels are displayed on the side. As we can see, over-exposition and tilted face profile can have a large impact on the prediction quality.

Stitch (XS) (Misra et al., 2016), widened XS to match the number of parameters of our approach (XSx) and finally Cross-Residual (XR) (Jou and Chang, 2016). For TCDCN, we directly report the results of Zhang et al. (2014). For the others, we trained each one three times for 300 epochs. We report the landmark failure rate and the normalized landmark mean error averaged over the last five epochs, further averaged over the three tries.

Figure 7.5 and Figure 7.6 presents our FLD results on the MTFL dataset. The first figure corresponds to using AlexNet as underlying network, while the second figure corresponds to ResNet18. In each figure, the top part reports the landmark failure rate, while the bottom part reports the mean error. In each plot, the left bar (blue) is for un-pretrained network, while the right bar (green) is for ImageNet pre-trained network.

The results of Figure 7.5 and Figure 7.6 show that our proposed approach obtained the lowest failure rates in each case. With AlexNet, our approach obtained 19.67% when initialized at random and 19.96% when pre-trained on ImageNet. As a side note, Zhang et al. (2014) reported 25.00% failure rate with their TCDCN approach, which used a small CNN similar to AlexNet. With ResNet18, our approach obtained 14.95% when initialized at random and 13.52% when pre-trained on ImageNet. The results also show that our approach outperforms hard parameter sharing (with a matching number of parameters), which adds to the existing experimental evidence that soft parameter sharing is gaining over hard parameter sharing.

7.3.3 Effect of Data Scarcity on the AFLW Task

As a second experiment, we evaluated the impact of data scarcity on the Annotated Facial Landmarks in the Wild (AFLW) task (Koestinger et al., 2011). The dataset contains 21,123 Flickr images and defines 21 facial landmarks with gender recognition, glasses recognition and face orientation as related tasks. To perform our experiment, we first preprocessed each image and extracted all faces with visible landmarks using the provided bounding boxes. This step is required because the original images of the dataset can have more than one face. The result of the preprocessing gave a total of 2,111 face images. For face orientation, we divided the roll, yaw and pitch into 30 degrees wide bins, and predicted the label corresponding to each real value. This corresponds to 3 classification problems with 12 classes each.

Our experiment works as follows. We trained the soft parameter sharing network with ResNet18 pre-trained on ImageNet as underlying network on a varying number of im-

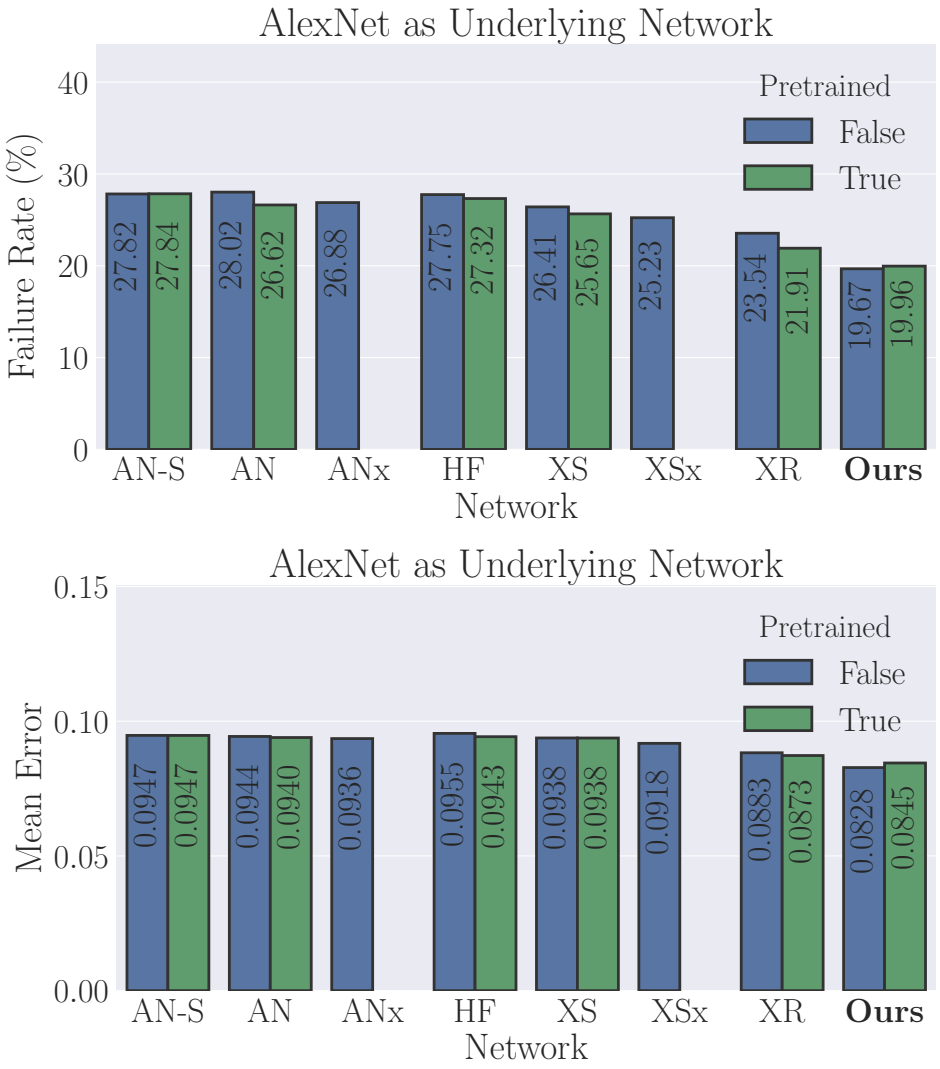


Figure 7.5: Landmark failure rates (%) and mean error on the MTFL task using AlexNet as the underlying network. The reported values are the average over the last five epochs, further averaged over three tries. AN-S stand for single-task training, AN for multitask training with a single central network, ANx for multitask training with a single central network widen to match the number of parameters of our approach, HF for HyperFace, XS and XSx for Cross-Stitch and widened Cross-Stitch, and XR for Cross-Residual. In each plot, the left column (blue) is for random initialization of the underlying network, while the right column (green) is for underlying networks pre-trained on ImageNet. Our proposed approach obtains the lowest failure rates overall.

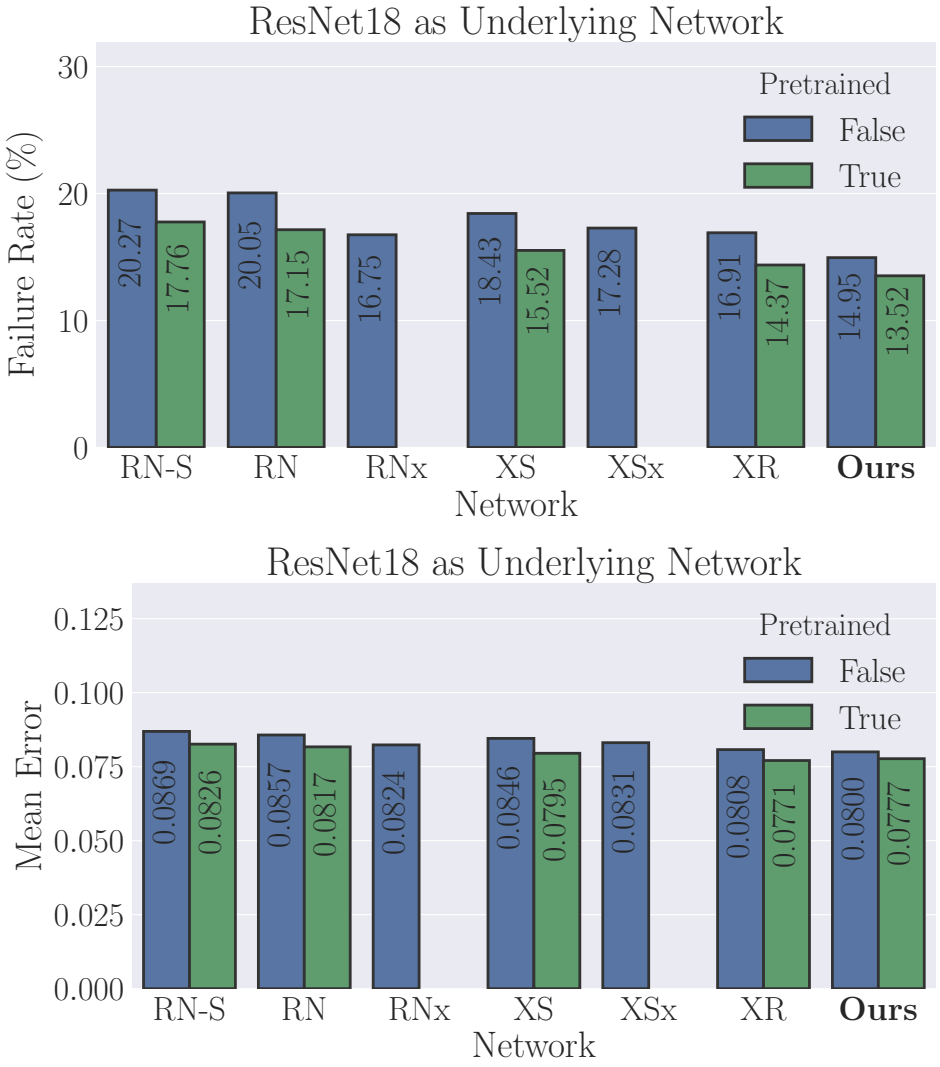


Figure 7.6: Landmark failure rates (%) and mean error on the MTFL task using ResNet18 as the underlying network. The reported values are the average over the last five epochs, further averaged over three tries. RN-S stand for single-task training, RN for multitask training with a single central network, RNx for multitask training with a single central network widen to match the number of parameters of our approach, HF for HyperFace, XS and XSx for Cross-Stitch and widened Cross-Stitch, and XR for Cross-Residual. In each plot, the left column (blue) is for random initialization of the underlying network, while the right column (green) is for underlying networks pre-trained on ImageNet. Our proposed approach obtains the lowest failure rates overall.

Table 7.1: Landmark failure rate results on the AFLW dataset using a pre-trained ResNet18 as underlying network. The presented values are averaged over the last five epochs, further averaged over three tries. The first column is the train / test ratio, and the subsequent ones are the networks: single-task ResNet18 (RN-S), multitask ResNet18 (RN) and Cross-Stitch network (XS). Our approach obtains the best performance in all cases, except the first one where we observe over-fitting.

Train / Test Ratio	**Landmark Failure Rate (%)**			
	RN-S	RN	XS	Ours
0.1 / 0.9	**57.39**	58.00	73.06	60.64
0.3 / 0.7	31.84	32.00	36.24	**29.73**
0.5 / 0.5	23.41	23.31	26.02	**20.77**
0.7 / 0.3	21.47	21.92	22.37	**18.50**
0.9 / 0.1	13.03	12.80	13.51	**10.82**

ages. We trained on the first $r\%$ of the available images and tested on the other $(1-r)\%$, then repeated for different train / test ratios. We used a total of five train / test ratios, which corresponds to 0.1 / 0.9, 0.3 / 0.7, 0.5 / 0.5, 0.7 / 0.3 and 0.9 / 0.1. We compared our collaborative block with single-task ResNet18 (RN-S), multitask ResNet18 (RN) and Cross-Stitch network (XS) using the same training framework as in Section 7.3.2. We trained each approach three times for 300 epochs, and report the landmark failure rate averaged over the last five epochs, further averaged over the three tries. Example predictions on the AFLW task are shown in Figure 7.4.

The results of this experiment are presented in Table 7.1. The reported values show that our approach obtained the best performance in all cases expect the first one. Indeed, we observe improvement between 1.98% and 6.51% with train / test ratios from 0.3 / 0.7 to 0.9 / 0.1, while we obtain a negative relative change of 3.25% with train / test ratio of 0.1 / 0.9. However, since all MTL approaches obtained higher failure rates than the STL approach with train / test ratio of 0.1 / 0.9, this suggests that the networks are over-fitting the small training set. Nonetheless, these results show that our approach can obtain better performance in the context of data scarcity.

One particularity that we can observe in Table 7.1 is that XS has relatively high failure rates. In the previous experiment of Sec 7.3.2, XS had either similar or better performance than the other approaches (except ours) on the MTFL task. We were concerned that our MTL experimental framework for AFLW was unfavorable towards XS. Therefore, we investigated whether this was the case by performing the following experiment. We trained a soft parameter sharing network using ResNet18 pre-trained on ImageNet

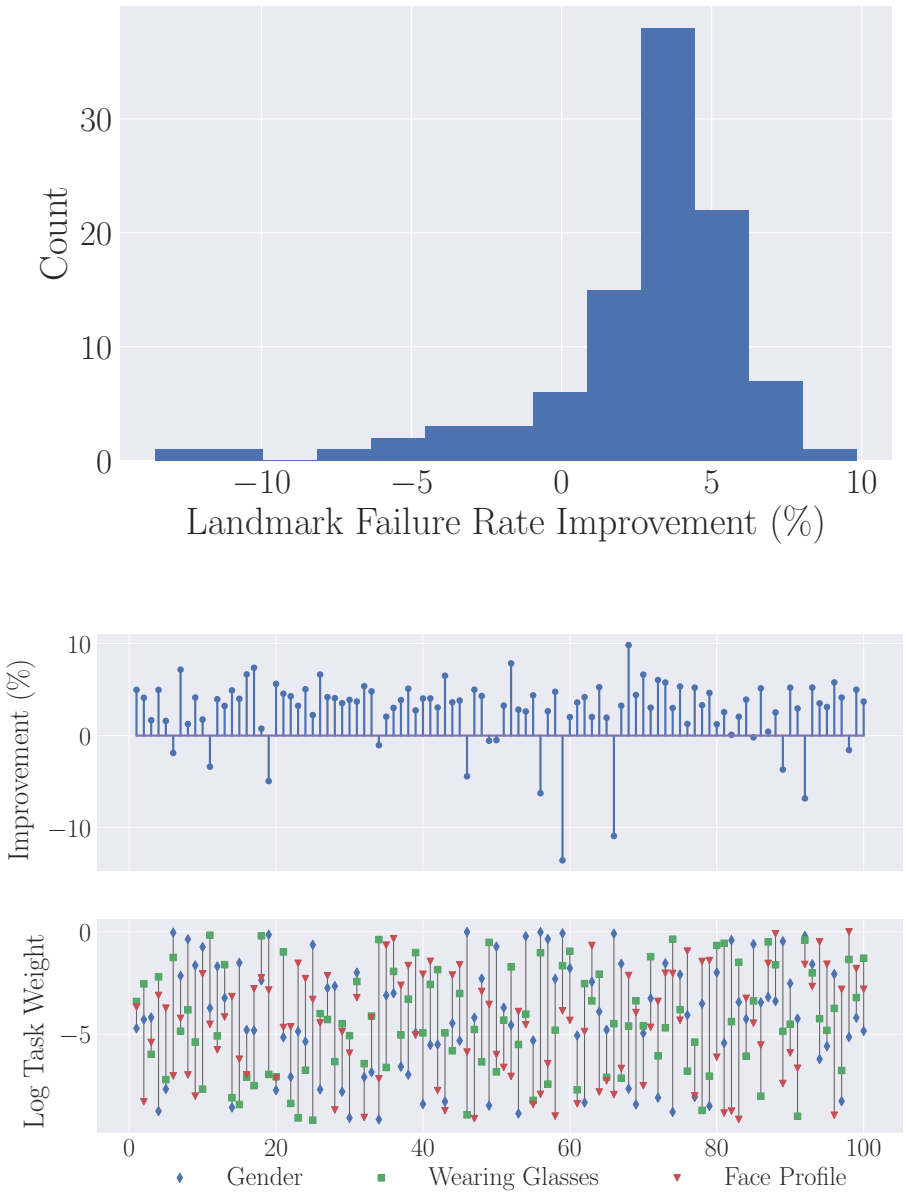


Figure 7.7: Landmark failure rate improvement (in %) of our approach compared to XS when sampling random task weights. We used a pre-trained ResNet18 as underlying network. The histogram on the top and the plot in the middle represent performance improvement achieved by our collaborative block over XS (positive value means lower failure rates), while the plot at the bottom corresponds to the log of the sampled task weights. Our approach outperformed XS in 86 out of the 100 tries. These results show that our learning framework was not unfavorable towards XS, which shows that our approach is less sensitive to task weights λ_t.

as underlying network using random task weights. The weights were randomly sampled from a log-uniform distribution as follows. We first sampled γ_t from a uniform distribution $\gamma_t \sim \mathcal{U}(\log(1\text{e}{-4}), \log(1))$, then used $\lambda_t = \exp(\gamma_t)$ as the task weight for task t. As a reminder, task weights are hyperparameters that control the importance of minimizing each task-specific objective. We compared our collaborative block to XS over 100 tries using a train / test ratio of 0.5 / 0.5. We used the same training framework as previously.

The results of this experiment as presented in Figure 7.7. The plot in the middle of the figure represents landmark failure rate improvements (in %) of our collaborative block over XS, while the plot at the bottom corresponds to the log of the sampled task weights for each try. In 86 out of the 100 tries, our approach had a positive failure rate improvement over XS. Moreover, we can see in the histogram at the top of Figure 7.7 that the improvement rates are normally distributed. The distribution has a mean improvement of 2.80%, a median improvement of 3.66% and a maximum improvement of 9.83%. Our approach outperformed XS in the majority of the cases even though we sampled the task weights of the related tasks at random. As a result, our training framework was not unfavorable toward XS.

7.3.4 Illustration of Knowledge Sharing With an Ablation Study

As a third experiment, we verified that our collaborative block effectively enabled knowledge sharing between the task-specific CNNs. To do so, we performed the following ablation study on the MTFL task. First, we trained a soft parameter sharing network with ResNet18 initialized at random, following the same training framework as described previously in Section 7.3.1. Then, we evaluated the impact of removing the contribution of each task-specific feature maps x_t during the computation of the global features z. That is, we zeroed out each feature map x_t before concatenation at the input of the central aggregation $\mathcal{H}$ and evaluated the impact on landmark failure rate. The ablation was performed at test time and the failure rates are reported on the test set after training is done.

The results of this experiment are presented in Figure 7.8. The blocks correspond to all collaborative blocks of the soft parameter sharing network and follows the same pattern as Figure 7.3. The blocks are ordered from left (input) to right (output), while the task-specific networks are ordered from top (main task) to bottom (related tasks).

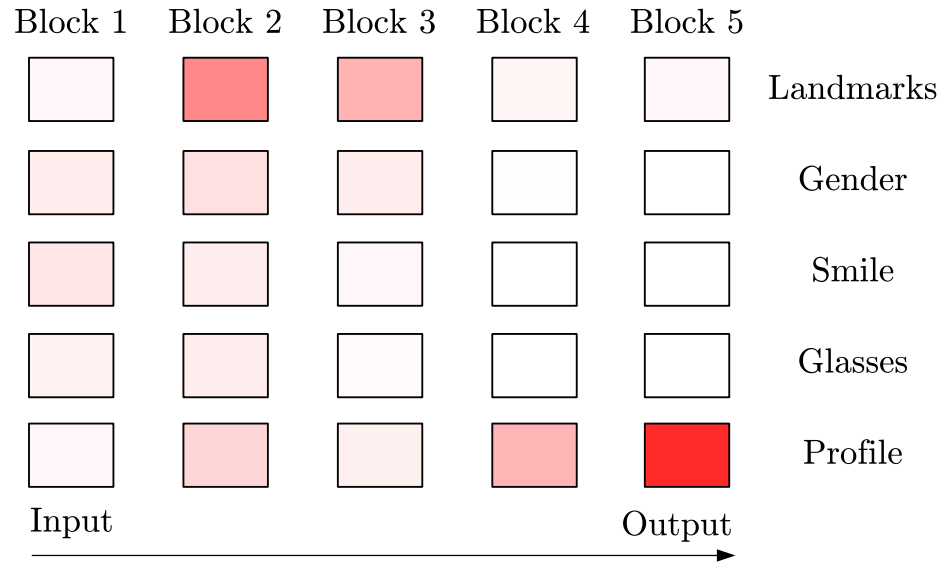


Figure 7.8: Results of our ablation study on the MTFL dataset with an un-pretrained ResNet18 as underlying network. We remove each task-specific features from each respective central aggregation layer and evaluate the effect on landmark failure rate. The rows represent the task-specific CNNs, while the columns correspond to the network block structure. Blocks with a high saturated color were found to have a large impact on failure rate. In particular, this ablative study shows that the influence of high-level face profile features is large within our proposed architecture. This corroborates with the well-known fact that the location of facial landmarks is closely dependent on the orientation of the face. This constitutes an empirical evidence of domain-specific information sharing via our approach.

The color saturation indicates the influence of removing the corresponding task-specific feature map at the corresponding depth from the central aggregation $\mathcal{H}$ in Eq. (7.2.5). A high saturation reflects high influence on failure rate, while a low saturation reflects low influence.

The first observation that we can make from the results of Figure 7.8 is that removing features from the facial landmark detection network significantly increases landmark failure rate. For instance, we observe a negative (worse) relative change of 29.72% and 47.00% in failure rate by removing features from Block 3 and Block 2 respectively. This result was expected, since we expected the FLD features to be influential for predicting facial landmarks. However, these results also illustrate that the FLD network contributed and fed from the global features computed by the central aggregation $\mathcal{H}$. The FLD network had the possibility to isolate itself from the other CNNs (by learning identity skip-connections in all collaborative blocks) but the opposite occurred. The task-specific features from the FLD network influenced the global features z, which in

turn influenced its own task-specific features. These results constitute a first indication of knowledge sharing between the network of the main task and the networks of the related tasks.

The second observation that we can make from the results of Figure 7.8 is that *Block 5* of task *Profile* has the highest influence on failure rate. We observe a negative relative change as high as 83.87% by removing its corresponding feature maps from the central aggregation. What is particularly interesting with this result is that the increase in error rate occurred at block 5, which is the highest block of the network. As we saw in Section 3.1, the features computed at the top of the hierarchy in a deep network have a high level of abstraction. In our case, these features are associated with the network for task Profile and we expect them to represent high-level factors of variation of face orientation. In particular, we expect them to look like a rotation matrix, since the goal is to predict the orientation of the face. Therefore, the network for landmark prediction can either learn features representing the orientation of the face in order to predict the correct orientation of the facial landmarks, or it can leverage the features computed by the face profile network that already represent face orientation. The results that we observe in Figure 7.8 show that it is the latter. These results constitute a second indication of knowledge sharing between the network of the main task and the networks of the related tasks.

7.3.5 Mixing Soft and Hard Parameter Sharing

As a fourth experiment, we investigated the potential research avenue of mixing hard parameter sharing with soft parameter sharing [1]. As illustrated in Fig. 7.9, the idea is to divide parameter sharing into two sections. The first section implements hard parameter sharing with a shared central network. This section starts at the input and ends at a certain layer, which is a hyper-parameter chosen by hand. The second section implements soft parameter sharing with one network per task equipped with a sharing mechanism. The section starts after the first section and ends at the output.

The idea of mixing hard parameter sharing with soft parameter sharing can be advantageous in the case where all tasks use the same observations. We have seen in Section 3.1 that deep networks are hierarchical models that compute features with an increasing level of abstraction. The features at the bottom layers represent low-level concepts, while the features at the top layers represent high-level concepts. Low-level features are

[1] We would like to thank Pr. Christian Gagné for suggesting this experiment during the thesis proposal.

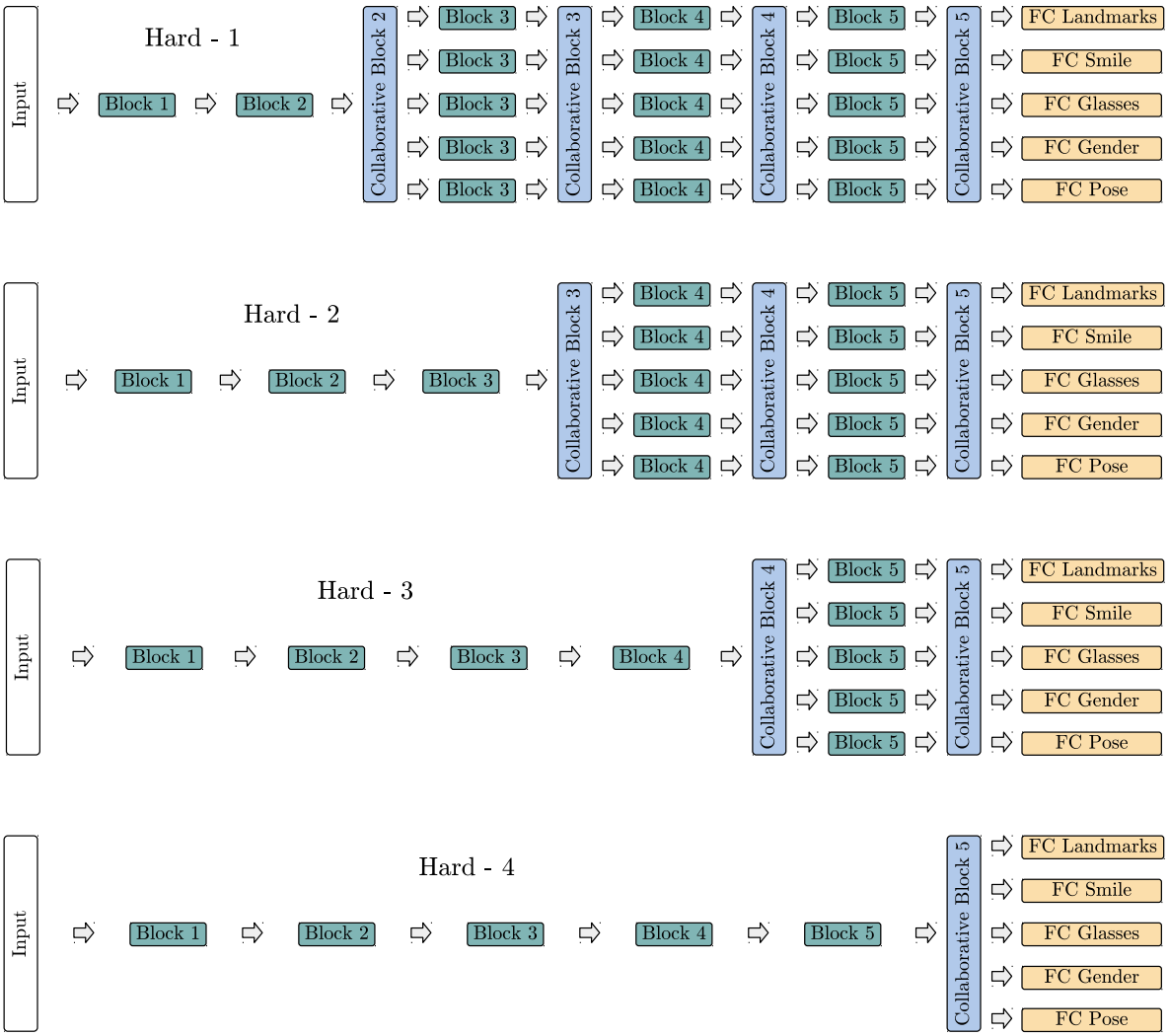


Figure 7.9: Illustration of mixing soft parameter sharing with hard parameter sharing on the MTFL task. Each row represent our soft parameter sharing network with a different hard parameter sharing length. The hard parameter sharing length is defined as the number of collaborative blocks removed from the network. The network at top has one collaborative block removed and has therefore a length of one. The second network has a length of two, the third network has a length of three and the fourth network has a length of four.

by definition simple features, appear in small numbers and are generic enough to be useful for many tasks. On the other hand, high-level features are by definition complex features, appear in large numbers and tend to be more detailed to specific tasks. It is therefore likely that the low-level features of each task will look very much alike, since they use the same observation. We may be able to take advantage of this to lower the complexity of the overall network, by replacing the soft parameter sharing section at the beginning with a simpler hard parameter sharing section.

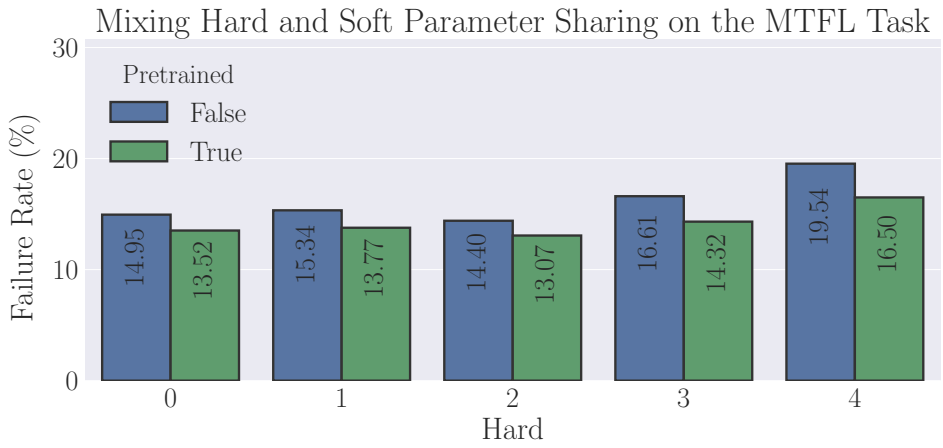


Figure 7.10: Landmark failure rates (%) of our soft parameter sharing network with a varying hard parameter sharing length on the MTFL task (*Hard 0* represents the original soft parameter sharing network). The reported values are the average over the last five epochs, further averaged over three tries. In each instance, the left column (blue) corresponds to random initialization of the underlying network, while the right column (green) is for pre-training on ImageNet. The results show that we can obtain performance improvement by using a hard parameter sharing length of two (Hard 2).

To investigate this idea, we performed a series of experiment on the MTFL task. We trained a soft parameter sharing network with ResNet18 as underlying network by including a hard parameter sharing section at the beginning. We varied the length of the hard parameter sharing section by removing collaborative blocks and by using a single central series of residual blocks.

An illustration of how we mix hard parameter sharing with soft parameter sharing is shown in Figure 7.9. The figure presents four instantiations of our soft parameter sharing network with four different hard parameter sharing length. We define the hard parameter sharing length as the number of collaborative blocks removed from the network. The network at the top of the figure has one collaborative block removed and therefore has a hard parameter sharing length of one. The second network has a hard parameter sharing length of two, the third one has a length of three and the fourth one has a length of four. We trained each network using the same experimental framework as described in Section 7.3.1 and report the landmark failure rate.

The results of this experiment are shown in Figure 7.10. This figure presents landmark failure rates of our soft parameter sharing network with a varying hard parameter sharing length. *Hard 0* represents the original soft parameter sharing network of Figure 7.3, while *Hard 1* to *Hard 4* represent the four soft parameter sharing networks

with a hard parameter sharing section of Figure 7.9. In each plot, the left bar (blue) is for un-pretrained network, while the right bar (green) is for ImageNet pre-trained network.

The results of Figure 7.10 show that we can obtain performance improvement with a hard parameter sharing length of two. Indeed, the original soft parameter sharing network (*Hard-0*) with ResNet18 initialized at random had a landmark failure rate of 14.95%, while the one with a hard parameter sharing length of two (*Hard 2*) obtained 14.40%. Similary with ResNet18 pre-trained on ImageNet, *Hard-0* had 13.52%, while *Hard 2* obtained 13.07%.

Moreover, we also observe that performance starts to degrade starting at *Hard 2*. The error rate increases as the network resemble more and more standard hard parameter sharing. With ResNet18 initialized at random, the error rate increases up to 19.54%, while with ResNet18 pre-trained on ImageNet, the error rate increases up to 16.50%. As a reference, standard hard parameter sharing obtained 20.05% and 17.15% respectively (ref. Figure 7.6). Note that we still observe performance improvement with *Hard-4* in comparison to standard hard parameter sharing, since *Hard-4* is not exactly identical to it. The last collaborative block before the fully connected layers gives a form of flexibility on information sharing that hard parameter sharing does not provide.

These results therefore indicate that mixing soft parameter sharing with hard parameter sharing can bring performance improvement. This opens up the possibility to revisit existing soft parameter sharing approaches, since this has not been considered previously.

7.3.6 Large-Scale Facial Landmark Detection With MTCNN

As a final experiment, we performed a large-scale evaluation using the recent Multi-task Cascaded Convolutional Network (MTCNN) (Zhang et al., 2016). The authors of MTCNN proposed a cascade structure of three stages to perform predictions in a coarse-to-fine manner. The first stage generates (in a fully-convolutional way) many hypotheses about the position of the face and the facial landmarks, and the subsequent second and third stages refines them. Each stage is defined as a hard parameter sharing CNN that performs face classification, face bounding box prediction and facial landmark prediction. The CNNs are trained one after the other with hard-negative mining, where the network at the current stage uses the predictions from the previous stage for training.

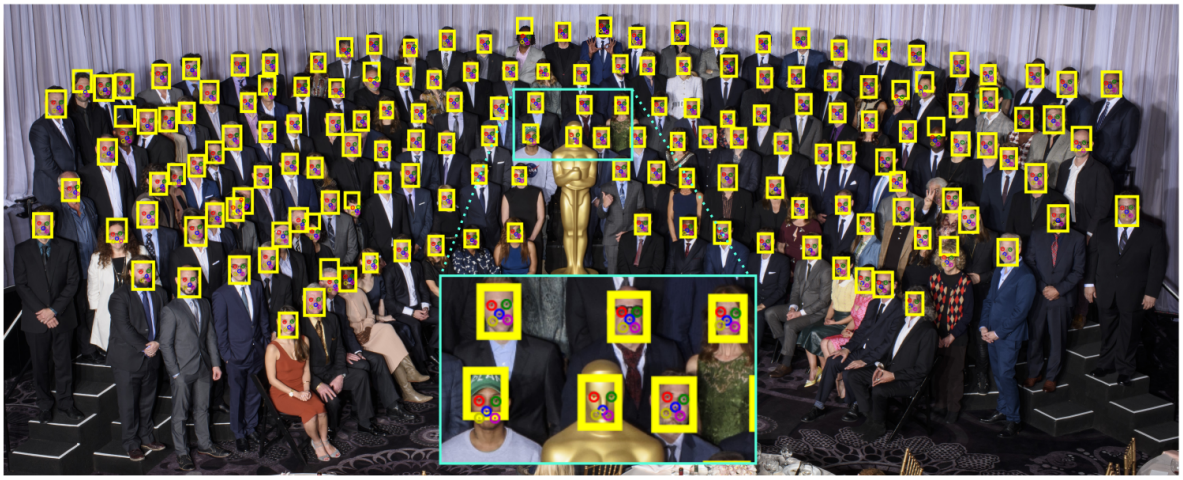


Figure 7.11: MTCNN predictions on the photo of the 2017 Oscar nominees (image resolution of 2983 × 1197). The stage-CNNs are trained using our proposed collaborative block. The coarse-to-fine detection scheme employed by MTCNN allows predicting many faces in high-dimensional images with low computational burden.

We implemented MTCNN using the available code project (Kim, 2018). We compared the original version with hard parameter sharing (MTCNN-H) with our soft parameter sharing version with collaborative blocks (MTCNN-S). We followed the provided hard-negative mining recipe to generate the observations. For landmark detection, we used the LFWNet (Sun et al., 2013) and CelebA (Liu et al., 2015b) datasets, and generated 600k face images with facial landmarks. For face detection, we used the WIDER (Yang et al., 2016) dataset, and generated 1.5M face images with a bounding box. We made sure to widen each stage network of MTCNN-H to match the number of parameters of MTCNN-S, in order to have a fair comparison.

An example of the prediction capability of our MTCNN-S is shown in Figure 7.11. The image corresponds to the official photo of the 2017 Oscar nominees. It has a resolution of 2983 × 1197 pixels, which is considered high-dimensional nowadays. The predictions of the network are shown as yellow bounding boxes with five colored circles. The bounding boxes represent predicted locations of faces, while the circles represent predicted facial landmarks. A zoomed-in crop of the image is shown at the bottom for more details. The coarse-to-fine detection style of MTCNN allowed us to compute all these predictions in less than ten seconds. Note that some bounding box predictions are not consistent with their corresponding facial landmark predictions. This is because MTCNN does not condition its facial landmark predictions on its bounding box prediction. It predicts the two separately. Nonetheless, we only use the predicted facial landmarks to compute the landmark failure rates.

We evaluated both MTCNN-H and MTCNN-S on the MTFL (Zhang et al., 2014) task. The results indicate that our MTCNN-S obtained better performance than the original MTCNN-H. On the test set of the MTFL dataset, MTCNN-H obtained a landmark failure rate of 37.85%, a mean error of 0.0996 and failed to detect a face 112 times, while our MTCNN-S obtained better performances with a landmark failure rate of 28.97%, a mean error of 0.0930 and failed to detect a face 79 times respectively. Note that the performance of both MTCNN-H and MTCNN-S is significantly lower than the performance of ResNet18, as reported in Fig. 7.6. This is because Zhang et al. (2016) carefully designed their MTCNN architecture to balance computational speed and landmark prediction.

7.4 Conclusion

In this chapter, we presented a novel soft parameter sharing mechanism. Our approach implements knowledge sharing as a trainable component of the network based on the recent advances in residual learning. Our *collaborative block* uses two distinct nonlinear transformations to connect the task-specific networks. The first one aggregates task-specific features into global features and employs an identity skip connection with a concatenation. The second one merges back the global features into each task-specific network and employs an identity skip connection with an addition.

We performed several MTL experiments to evaluate our approach. We opted for Facial Landmark Detection (FLD), since many datasets designed for FLD also contain a variety of related labels that can be used to define related tasks. We studied FLD with five landmarks and four related tasks on the MTFL dataset, and FLD with twenty-one landmarks and three related tasks on the AFLW dataset. The overall results show that our collaborative block obtains the best performance over the others of the state of the art.

We also performed four sets of experiments to improve our understanding of our sharing mechanism. We first experimented with data scarcity. Our collaborative block obtained better performance in all data scarcity settings, except the first one where we observed overfitting for the majority the approaches. Second, we performed an ablation study to illustrate knowledge sharing between the task-specific networks. We observed depth-specific influence of tasks that we know are related, which constitutes an empirical evidence that our approach enables leveraging domain-specific information from related tasks. Third, we explored the possibility of mixing hard parameter sharing with

soft parameter sharing. We obtained performance improvement when we used a hard parameter sharing section for the first half and a soft parameter sharing section for the second half. This opens up the opportunity to revisit existing approaches, since this has not been considered previously. Finally, we performed a large-scale experimental evaluation with the MTCNN approach. Our collaborative block obtained the better accuracy than the original hard parameter sharing.

In conclusion, our work in this chapter has shown that soft parameter sharing can be a suitable alternative to hard parameter sharing. It corroborates with the previous observations that the sharing mechanism in soft parameter sharing provides more flexibility on how to implement information sharing than the central network in hard parameter sharing. It supports efforts in designing better sharing mechanism incorporating novel advances in deep learning.

Conclusion

This thesis considers representation learning, which is the task of learning a feature representation from the data to make the learning problem easier. We demonstrated a number of ways to improve existing approaches to learn better feature representations to further facilitate the learning problem. In this chapter, we conclude with a summary of our principal contributions and with a discussion of research avenues for future work.

Summary of the Contributions

Our contributions can be summarized as ways to improve existing representation learning approaches. They can be broken down into three parts, where each part corresponds to a chapter. In Chapter 5, we explored representation learning on the problem of grasp localization. We performed an experimental evaluation of sparse dictionary learning on grasp localization and proposed a residual network for grasp detection. In Chapter 6, we detailed our contribution in deep learning with a novel parametric activation function. We perform experiments on object recognition with state-of-the-art deep convolutional networks. Finally, we presented in Chapter 7 a novel soft parameter sharing mechanism for multitask learning. We perform experiments on facial landmark detection with related tasks. We now elaborate on each contribution and provide research avenues for future work.

Representation Learning Exploration on the Problem of Grasp Localization

Concluding Remarks

In Chapter 5, we explored representation learning on the problem of grasp localization. The difficulty of localizing good grasping points is one of the major causes that limits the range of application of grasping systems. It is a challenging problem that involves

interpreting light and depth information from a complex high-dimensional RGBD image.

We made two contributions in Chapter 5. As first contribution, we presented an empirical evaluation of Sparse Dictionary Learning (SDL) on RGBD images. SDL has a long-standing history of approaches for RGB images, but few have been evaluated on RGBD data. We elaborated a comparative framework based on grasp recognition and grasp detection, and compared the accuracy and speed for a total of thirty-six combinations of features coding and dictionary learning approaches. In the second contribution, we proposed a Convolutional Neural Network (CNN) for grasp detection. We extended the *one-shot* architecture of Redmon and Angelova (2015) to include residual learning and addressed the spatial information loss in the error signal on the confidence map caused by global average pooling.

The results of our comparative study showed that shallow approaches like SDL can obtain good performances. This was possible due to the characteristics of the dataset and due to the use of the exhaustive, but tedious, sliding window search. Indeed, we showed that the Cornell grasping dataset (Jiang et al., 2011) had low variability and was relatively small in comparison to nowadays grasping datasets (Depierre et al., 2018). Also, the use of sliding window was exhaustive, but made the search slow. As a result, SLD approaches obtained similar detection accuracies than our CNN, but were much slower in comparison to the *one-shot* detection style of our CNN.

Future Work

Our work in Chapter 5 has shown that it is conceivable to use another machine learning framework than deep learning when the task is appropriate. The advent of deep learning has shifted the focus of several machine learning communities and most now consider only deep neural networks when tackling new problems. Our work in Chapter 5 is a reminder that, even though deep neural networks will always outperform shallow approaches on complex and large-scale tasks, shallow approaches can still have a role to play on simple and small-scale tasks.

In that sense, our work in Chapter 5 opens up the question of how to determine the difficulty of a task in order to motivate the choice of a shallow or a deep approach. For instance, deep approaches often struggle with overfitting on small-scale tasks, while shallow approaches do not. On the contrary, shallow approaches often struggle with high-dimensional inputs, while deep approaches are flexible enough to accommodate to

the complexity. How can we then justify the choice of a shallow or a deep approach? And more generally, how can we measure the complexity of a task in relation to a shallow and a deep approach? Addressing these questions could help to provide a more informed view on how to make this choice.

Another research avenue is to consider convolutional SDL (Garcia-Cardona and Wohlberg, 2017). The idea is to create a convolutional dictionary similar to a convolutional network, but where the dot product is replaced by feature coding. The concern would be to design a feature coding approach that would incorporate sparsity efficiently, such that it can be used on a large input image. This is an interesting avenue for future works, since it would bridge the gap between SDL and deep learning.

Deep Learning Development: Activation Function

Concluding Remarks

In Chapter 6, we contributed with a novel parametric activation function. The particularity of parametric activation functions is that they define parameters that control different aspects of their shape. We proposed a parameterization of the Exponential Linear Unit (ELU) (Clevert et al., 2016), which has been previously proposed to deal with the problem of bias shift. Our Parametric ELU (PELU) adds two parameters that control the slope of the linear part for positive arguments, as well as the saturation value and the decay towards the saturation for negative arguments.

We showed that our PELU improves performance with an experimental evaluation of several state-of-the-art CNNs on object recognition. We evaluated a total of six networks on the MNIST, CIFAR-10, CIFAR-100 and ImageNet datasets where we showed that our activation function obtained better performances. Moreover, we performed a series of experiments to better understand our parameterization. We evaluated the effect of Batch Normalization (BN), the effect of learning the inverse of each parameter and the progression of the parameters throughout training.

Future Work

Clevert et al. (2016) evaluated the effect of batch normalization (BN) before the ELU and showed that it was detrimental for performance. We also evaluated the effect of BN on our proposed parameterization of the ELU and showed that it was also detrimental, although the performance degradation was smaller. Moreover, we observed that a network using ELU or PELU can still benefit from BN when it is placed at a different

location than before the activation. This contrasts with activation functions like ReLU or PReLU that always benefit from BN, no matter where it is placed.

We pointed out that there is a difference between ReLU and ELU that could explain why BN is beneficial before one, but detrimental before the other. The difference is that ReLU is positive scale invariant, while ELU and PELU are not. In other words, a positive scaling of the unit before ReLU is equivalent to a positive scaling of the unit after ReLU. This is not the case for the ELU and PELU because of the exponential decay towards the saturation.

Positive scale invariance may be important because BN is effectively a scaling transformation. It subtracts the mean activation from the input unit and divide it by the activation variance, followed by a trainable scaling and bias transformation. There may be a link between positive scale invariance for the activation and the ability to reduce internal covariate shift for BN. Positive scale invariance may be needed to benefit from this reduction during training when BN scales the unit positively. One way to investigate the effect of positive scale invariance is to perform an in-depth study of positive scale invariant activation functions and positive scale *variant* activation functions. A study of this kind would bring more understanding of the relation between the activation function and BN.

Deep Learning Development: Multi-Task Learning

Concluding Remarks

In Chapter 7, we contributed with a novel soft parameter sharing mechanism for Multitask Learning (MTL). Approaches in the soft parameter sharing family implement MTL by defining one network for each task and by connecting them with a sharing mechanism. The sharing mechanism allows domain information from all tasks to be shared to every network, such that what is learned for one task can be available to the others.

Previous soft parameter sharing approaches are limited in the type of task relations that they can learn. They define their sharing mechanism as trainable linear combinations, which implies that they can only learn linear task relations. We proposed a sharing mechanism that addresses this limitation by using nonlinear transformations, which we refer to as our Collaborative Block. We design the nonlinear transformations based on skip connections in order to benefit from all the advantages of residual learning. In

particular, residual learning makes learning the identity function easier, which in our case allows to more easily integrate the notion that task relevance depends on depth.

We performed an experimental evaluation on the problem of Facial Landmark Detection (FLD) in a MTL setting. We compared our collaborative block to the others of the state of the art on the MTFL, AFLW, CelebA and WIDER dataset. The results show that our approach obtains the best performance overall. We also performed a series of experiments to better understand the effect of our sharing mechanism. We investigated the effect of data scarcity, the potential of mixing soft and hard parameter sharing and performed a large-scale experiment using the MTCNN architecture. Finally, we showed an illustration of knowledge sharing with an ablation study.

Future Work

The sharing mechanism can be designed in any way, as long as it enables the domain information of each task to be shared with all tasks. We are not limited to connect the task-specific networks using standard transformations, like performing a linear transformation followed by BN and ReLU. However, the main difficulty in designing a sharing mechanism is to ensure that only relevant information is shared and that task-specific information is shared only to relevant networks. This points to a possible research direction of designing a sharing mechanism that directly focuses on these considerations. One recent deep learning advance that may play this role is the attention mechanism (Bahdanau et al., 2014), and in particular the visual attention mechanism (Xu et al., 2015; Hara et al., 2017).

The attention mechanism was originally proposed in the context of neural machine translation to circumvent the need for the network to memorize long sentences (Bahdanau et al., 2014). The goal of this attention mechanism is to create a set of scores that rank each element in relation to all other elements. The scores are positive and sum to one, which means that increasing the score of a specific element lowers the scores of all the others. Only a few elements can have a large score and be influential during the forward pass, which forces the network to select only the most important ones, hence the name *attention mechanism*. The elements can be anything, such as words in a sentence (Bahdanau et al., 2014), feature maps or pixels (Xu et al., 2015; Hara et al., 2017).

It is therefore natural to consider the attention mechanism as a sharing mechanism. The elements of the attention mechanism could be defined as the task-specific feature maps,

such that the attention mechanism would model task relations. It will learn to place attention on specific parts of the feature maps when computing the output features for the main task, and will learn to place attention on other parts when computing the output features for the related tasks. This way, the sharing mechanism could directly model task relation with attention instead of a series of nonlinear transformations. This could prove promising to improve domain-specific information sharing between the tasks. It would also have the beneficial side effect of easing the interpretability of the predictions made by the network. Investigating this research avenue could lead to a better sharing mechanism.

Bibliography

F. Agostinelli, M. D. Hoffman, P. J. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *International Conference on Learning Representations: Workshops (ICLR'14)*, 2014.

A. L. Alter and D. M. Oppenheimer. Predicting short-term stock fluctuations by using processing fluency. *Proceedings of the National Academy of Sciences (NAS'06)*, 103 (24):9369–9372, 2006.

A. L. Alter, D. M. Oppenheimer, N. Epley, and R. N. Eyre. Overcoming intuition: metacognitive difficulty activates analytic reasoning. *Journal of Experimental Psychology: General*, 136(4):569, 2007.

A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems (NIPS'07)*, pages 41–48, 2007.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR'14)*, 2014.

K. Balasubramanian, K. Yu, and G. Lebanon. Smooth sparse coding via marginal regression for learning sparse representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 289–297, 2013.

T. Baltrušaitis, P. Robinson, and L.-P. Morency. Openface: an open source facial behavior analysis toolkit. In *Proceedings of the 2016 IEEE Winter Conference on Applications of Computer Vision (WACV'16)*, pages 1–10, 2016.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8):1798–1828, 2013.

Y. Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

A. Bicchi and V. Kumar. Robotic grasping and contact: a review. In *Proceedings of the 2000 International Conference on Robotics and Automation (ICRA'00)*, pages 348–353, 2000.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

C. Blanco, Y. Xu, K. Brady, G. Pérez-Fuentes, M. Okuda, and S. Wang. Comorbidity of posttraumatic stress disorder with alcohol dependence among us adults: results from national epidemiological survey on alcohol and related conditions. *Drug and Alcohol Dependence*, 132(3):630–638, 2013.

M. Blum, J. T. Springenberg, J. Wülfing, and M. Riedmiller. A learned feature descriptor for object recognition in RGB-D data. In *Proceedings of the 2012 International Conference on Robotics and Automation (ICRA'12)*, pages 1298–1303, 2012.

T. Blumensath and M. E. Davies. Gradient pursuits. *IEEE Transactions on Signal Processing*, 56(6):2370–2382, 2008.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

L. Bo, X. Ren, and D. Fox. Unsupervised feature learning for RGB-D based object recognition. In *Proceedings of the 2013 International Symposium on Experimental Robotics (ISER'13)*, pages 387–402, 2013.

L. Bo, X. Ren, and D. Fox. Learning hierarchical sparse features for RGB-(D) object recognition. *International Journal of Robotics Research (IJRR)*, 33(4):581–599, 2014.

R. Bogue. Domestic robots: Has their time finally come? *Industrial Robot: An International Journal*, 44(2):129–136, 2017.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 15th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–186. 2010.

P. R. Cannon, A. E. Hayes, and S. P. Tipper. Sensorimotor fluency influences affect: Evidence from electromyography. *Cognition and Emotion*, 24(4):681–691, 2010.

A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *Computing Research Repository (CoRR)*, abs/1605.07678, 2016.

R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin. Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access*, 6:6505–6519, 2018a.

Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 794–803, 2018b.

R. S. Choras. Image feature extraction techniques and their applications for CBIR and biometrics systems. *International journal of biology and biomedical engineering*, 1 (1):6–16, 2007.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *International Conference on Learning Representations (ICLR'16)*, 2016.

A. Coates and A. Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 921–928, 2011.

A. Coates and A. Y. Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pages 561–580. Springer, 2012.

A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, pages 215–223, 2011.

P. Collet and J. Rennard. Stochastic optimization algorithms. In *Handbook of research on nature-inspired computing for economics and management*, pages 28–44. 2007.

M. Corley, L. J. MacGregor, and D. I. Donaldson. It's the way that you, er, say it: Hesitations in speech affect language comprehension. *Cognition*, 105(3):658–668, 2007.

N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 2016.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

J. G. Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Research*, 20(10):847–856, 1980.

J. G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America (JOSA)*, 2(7):1160–1169, 1985.

H. Daume III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.

L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. L. Seltzer, G. Zweig, X. He, J. D. Williams, et al. Recent advances in deep learning for speech research at microsoft. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, volume 26, page 64, 2013.

A. Depierre, E. Dellandréa, and L. Chen. Jacquard: A large scale dataset for robotic grasp detection. *Computing Research Repository (CoRR)*, abs/1803.11469, 2018.

G. Desjardins, K. Simonyan, R. Pascanu, et al. Natural neural networks. In *Advances in Neural Information Processing Systems (NIPS'15)*, pages 2062–2070, 2015.

R. Detry, C. H. Ek, M. Madry, and D. Kragic. Learning a dictionary of prototypical grasp-predicting parts from grasping experience. In *Proceedings of the 2013 International Conference on Robotics and Automation (ICRA'13)*, pages 601–608, 2013.

I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42(1-2):143–175, 2001.

C. Ding and D. Tao. Robust face recognition via multimodal deep face representation. *IEEE Transactions on Multimedia*, 17(11):2049–2058, 2015.

C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(2):295–307, 2016.

D. L. Donoho. For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(6):797–829, 2006.

D. L. Donoho and I. M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

D. L. Donoho, Y. Tsaig, I. Drori, and J.-L. Starck. Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 58(2):1094–1121, 2012.

L. Duong, T. Cohn, S. Bird, and P. Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 845–850, 2015.

B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

H. C. Ellis. *The transfer of learning.* Macmillan, New York, 1965.

K. Engan, S. O. Aase, and J. Hakon Husoy. Method of optimal directions for frame design. In *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, pages 2443–2446, 1999.

T. Evgeniou and M. Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'04)*, pages 109–117, 2004.

C. Fabian Benitez-Quiroz, R. Srinivasan, and A. M. Martinez. Emotionet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 5562–5570, 2016.

J. Fan and J. Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5): 849–911, 2008.

I. Fogel and D. Sagi. Gabor filters as texture discriminator. *Biological cybernetics*, 61 (2):103–113, 1989.

C. Garcia-Cardona and B. Wohlberg. Convolutional dictionary learning. *Computing Research Repository (CoRR)*, abs/1709.02893, 2017.

C. R. Genovese, J. Jin, L. Wasserman, and Z. Yao. A comparison of the lasso and marginal regression. *Journal of Machine Learning Research (JMLR)*, 13(1):2107–2143, 2012.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, pages 315–323, 2011.

C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelossof. Grasp planning via decomposition trees. In *Proceedings of the 2007 International Conference on Robotics and Automation (ICRA'07)*, pages 4679–4684, 2007.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 1319–1327, 2013.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. 2016.

S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *Proceedings of the 2014 European Conference on Computer Vision (ECCV'14)*, pages 345–360. 2014.

K. Hara, M. Liu, O. Tuzel, and A. Farahmand. Attentional network for visual object detection. *Computing Research Repository (CoRR)*, abs/1702.01478, 2017.

K. Hashimoto, Y. Tsuruoka, R. Socher, et al. A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP'17)*, pages 1923–1933, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV'15)*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016a.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Proceedings of the 2016 European Conference on Computer Vision (ECCV'16)*, pages 630–645, 2016b.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

K. Hitomi. Automation — its concept and a short history. *Technovation*, 14(2):121 – 128, 1994.

S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the 2018 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'18)*, June 2018.

G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, pages 2261–2269, 2017.

T.-Y. Hung, J. Lu, Y.-P. Tan, and S. Gao. Efficient sparsity estimation via marginal-lasso coding. In *Proceedings of the 2014 European Conference on Computer Vision (ECCV'14)*, pages 578–592. Springer, 2014.

A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, pages 448–456, 2015.

A. K. Jain and F. Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern Recognition*, 24(12):1167–1186, 1991.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the 2009 IEEE International Conference on Computer Vision (ICCV'09)*, pages 2146–2153, 2009.

Y. Jiang, S. Moseson, and A. Saxena. Efficient grasping from RGBD images: Learning using a new rectangle representation. In *Proceedings of the 2011 International Conference on Robotics and Automation (ICRA'11)*, pages 3304–3311, 2011.

X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, 2016.

I. Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

B. Jou and S.-F. Chang. Deep cross residual learning for multitask visual recognition. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 998–1007, 2016.

A. Jourabloo and X. Liu. Large-pose face alignment via cnn-based dense 3d model fitting. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 4188–4196, 2016.

A. Karpathy. Software 2.0, July 2018. URL https://medium.com/@karpathy/software-2-0-a64152b37c35.

B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2): 398–409, 2015.

B.-K. Kim. *Internationalizing the Internet: the Co-evolution of Influence and Technology*. Edward Elgar Publishing, 2005.

K. K. Kim. Deep learning face detection and recognition, implemented by pytorch., February 2018. URL https://github.com/kuaikuaikim/DFace.

R. Kiros, R. Salakhutdinov, and R. Zemel. Multimodal neural language models. In *Proceedings of the 31th International Conference on Machine Learning (ICML'14)*, pages 595–603, 2014.

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems (NIPS'17)*, pages 971–980, 2017.

M. Koestinger, P. Wohlhart, P. M. Roth, and H. Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *Proceedings of the 2011 IEEE International Conference on Computer Vision Workshops (ICCV'11 Workshops)*, pages 2144–2151, 2011.

K. Kreutz-Delgado, J. F. Murray, B. D. Rao, K. Engan, T.-W. Lee, and T. J. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2): 349–396, 2003.

A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 1(4), 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS'12)*, pages 1097–1105, 2012.

K. Labusch, E. Barth, and T. Martinetz. Robust and fast learning of sparse codes with stochastic gradient descent. *IEEE Journal of Selected Topics in Signal Processing*, 5 (5):1048–1060, 2011.

K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *Proceedings of the 2011 International Conference on Robotics and Automation (ICRA'11)*, pages 1817–1824, 2011.

Q. V. Le, D. Kamm, A. F. Kara, and A. Y. Ng. Learning to grasp objects with multiple contact points. In *Proceedings of the 2010 International Conference on Robotics and Automation (ICRA'10)*, pages 5062–5069, 2010.

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

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems (NIPS'06)*, pages 801–808, 2006.

I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *International Journal of Robotics Research (IJRR)*, 34(4-5):705–724, 2015.

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

H. Li, Z. Xu, G. Taylor, and T. Goldstein. Visualizing the loss landscape of neural nets. *International Conference on Learning Representations: Workshops (ICLR'18)*, 2018.

M. Lin, Q. Chen, and S. Yan. Network in network. *Computing Research Repository (CoRR)*, abs/1312.4400, 2013.

Y. Lin, M. Song, D. T. P. Quynh, Y. He, and C. Chen. Sparse coding for flexible, robust 3d facial-expression synthesis. *IEEE Computer Graphics and Applications*, 32 (2):76–88, 2012.

C.-L. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Handwritten digit recognition: investigation of normalization and feature extraction techniques. *Pattern Recognition*, 37(2):265–279, 2004.

J. Liu, S. Ji, and J. Ye. Multi-task feature learning via efficient l 2, 1-norm minimization. In *Proceedings of the twenty-fifth Conference on Uncertainty in Artificial Intelligence (UAI'09)*, pages 339–348, 2009.

X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y.-Y. Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL'15)*, pages 912–921, 2015a.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV'15)*, 2015b.

M. Long and J. Wang. Learning multiple tasks with deep relationship networks. *Computing Research Repository (CoRR)*, abs/1506.02117, 2015.

M. Long, G. Ding, J. Wang, J. Sun, Y. Guo, and P. S. Yu. Transfer sparse coding for robust image representation. In *Proceedings of the 2013 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'13)*, pages 407–414, 2013.

Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *Proceedings of the 2017 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'17)*, pages 5334–5343, 2017.

A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, pages 689–696, 2009.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research (JMLR)*, 11(Jan):19–60, 2010.

J. Mairal, F. Bach, J. Ponce, et al. Sparse modeling for image and vision processing. *Foundations and Trends® in Computer Graphics and Vision*, 8(2-3):85–283, 2014.

J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proceedings of the 2010 International Conference on Robotics and Automation (ICRA'10)*, pages 2308–2315, 2010.

S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

S. Marĉelja. Mathematical description of the responses of simple cortical cells. *Journal of the Optical Society of America (JOSA)*, 70(11):1297–1300, 1980.

D. A. Mély and T. Serre. Towards a theory of computation in the visual cortex. In *Computational and Cognitive Neuroscience of Vision*, pages 59–84. Springer, 2017.

A. T. Miller and P. K. Allen. Graspit!: A versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122, 2004.

I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 3994–4003, 2016.

T. M. Mitchell. The need for biases in learning generalizations. Technical report, 1980.

N. Murata. A statistical study of on-line learning. In *On-line Learning in Neural Networks*, pages 63–92, 1999.

K. P. Murphy. Naive bayes classifiers. *University of British Columbia*, 18, 2006.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

D. Needell and R. Vershynin. Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit. *Foundations of Computational Mathematics*, 9(3):317–334, 2009.

M. S. Nixon and A. S. Aguado. *Feature extraction & image processing for computer vision*. Academic Press, 2012.

N. Novemsky, R. Dhar, N. Schwarz, and I. Simonson. Preference fluency in choice. *Journal of Marketing Research*, 44(3):347–356, 2007.

B. A. Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

D. M. Oppenheimer. Consequences of erudite vernacular utilized irrespective of necessity: Problems with using long words needlessly. *Applied Cognitive Psychology*, 20 (2):139–156, 2006.

S. Ovchinnikov. Max-min representation of piecewise linear functions. *Contributions to Algebra and Geometry*, 43(1):297–302, 2002.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(10):1345–1359, 2010.

R. Pascanu, G. Montúfar, and Y. Bengio. On the number of response regions of deep feedforward networks with piecewise linear activations. *International Conference on Learning Representations (ICLR'14)*, 2014.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems: Workshops (NIPS'17)*, 2017.

Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 40–44. IEEE, 1993.

J. Pearl et al. Causal inference in statistics: An overview. *Statistics Surveys*, 3:96–146, 2009.

R. Pelossof, A. Miller, P. Allen, and T. Jebara. An SVM learning approach to robotic grasping. In *Proceedings of the 2004 International Conference on Robotics and Automation (ICRA'04)*, volume 4, pages 3512–3518, 2004.

L. Pinto and A. Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *Proceedings of the 2017 International Conference on Robotics and Automation (ICRA'17)*, pages 2161–2168, 2017.

PIXNIO. Raccoon procyon lotor, 2018. URL https://pixnio.com/fauna-animals/raccoons/raccoon-procyon-lotor.

P. Ramachandran, B. Zoph, and Q. V. Le. Swish: a self-gated activation function. *Computing Research Repository (CoRR)*, abs/1710.05941, 2017.

R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. In *Proceedings of the 2015 International Conference on Robotics and Automation (ICRA'15)*, pages 1316–1322, 2015.

M. B. Reed, R. Wang, A. M. Shillington, J. D. Clapp, and J. E. Lange. The relationship between alcohol use and cigarette smoking in a sample of undergraduate college students. *Addictive Behaviors*, 32(3):449–464, 2007.

S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS'15)*, pages 91–99, 2015.

H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer, 1985.

C. Robert. Machine learning, a probabilistic perspective, 2014.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

M. Rouse. Artificial intelligence as a service (aiaas), July 2018. URL https://searchenterpriseai.techtarget.com/definition/Artificial-Intelligence-as-a-Service-AIaaS.

M. Rubin, S. Paolini, and R. J. Crisp. A processing fluency explanation of bias against migrants. *Journal of Experimental Social Psychology*, 46(1):21–28, 2010.

S. Ruder. An overview of multi-task learning in deep neural networks. *Computing Research Repository (CoRR)*, abs/1706.05098, 2017.

S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard. Learning what to share between loosely related tasks. *Computing Research Repository (CoRR)*, abs/1705.08142, 2017.

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

R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10)*, pages 2155–2162, 2010.

A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 1089–1096, 2011.

A. Saxena, J. Driemeyer, J. Kearns, and A. Y. Ng. Robotic grasping of novel objects. In *Advances in Neural Information Processing Systems (NIPS'06)*, pages 1209–1216, 2006.

A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research (IJRR)*, 27(2):157–173, 2008.

S. Scardapane, S. Van Vaerenbergh, S. Totaro, and A. Uncini. Kafnets: kernel-based non-parametric activation functions for neural networks. *Computing Research Repository (CoRR)*, abs/1707.04035, 2017.

M. Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab, 2005. URL `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`.

H. L. Seal. Studies in the history of probability and statistics. xv the historical development of the gauss linear model. *Biometrika*, 54(1-2):1–24, 1967.

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations (ICLR'14)*, 2014.

A. Shah, E. Kadam, H. Shah, S. Shinde, and S. Shingade. Deep residual networks with exponential linear unit. In *Proceedings of the Third International Symposium on Computer Vision and the Internet (VisionNet'16)*, pages 59–65. ACM, 2016.

S. H. Shah and I. Yaqoob. A survey: Internet of Things (IOT) technologies, applications and challenges. In *Proceedings of the 6th IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, pages 381–385. IEEE, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Computing Research Repository (CoRR)*, abs/1409.1556, 2014.

R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3D object classification. In *Advances in Neural Information Processing Systems (NIPS'12)*, pages 665–673, 2012.

A. Søgaard and Y. Goldberg. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 231–235, 2016.

H. Song and N. Schwarz. If it's difficult to pronounce, it must be risky: Fluency, familiarity, and risk perception. *Psychological Science*, 20(2):135–138, 2009.

J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *International Conference on Learning Representations: Workshops (ICLR'15)*, 2015.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

Standford. CS231n Convolutional neural networks for visual recognition, 2015. http://cs231n.github.io/neural-networks-1.

B. L. Sturm and M. G. Christensen. Comparison of orthogonal matching pursuit implementations. In *Proceedings of the 20th European Signal Processing Conference (EUSIPCO'12)*, pages 220–224, 2012.

C. Sun, Y. Yu, H. Liu, and J. Gu. Robotic grasp detection using extreme learning machine. In *Proceedings of the 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO'15)*, pages 1115–1120. IEEE, 2015.

Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the 2013 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'13)*, pages 3476–3483, 2013.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pages 1139–1147, 2013.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the 2015 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'15)*, pages 1–9, 2015.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the 2014 IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition (CVPR'14), pages 1701–1708, 2014.

S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

Y. Tian, P. Luo, X. Wang, and X. Tang. Pedestrian detection aided by deep learning semantic tasks. In *Proceedings of the 2015 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'15)*, pages 5079–5087, 2015.

G. H. Tijmen Tieleman. Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

I. Tosic and P. Frossard. Dictionary learning. *Signal Processing Magazine, IEEE*, 28 (2):27–38, 2011.

J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Simultaneous sparse approximation via greedy pursuit. In *Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05)*, volume 5, pages v–721, 2005.

L. Trottier, B. Chaib-draa, and P. Giguere. Temporal feature selection for noisy speech recognition. In *Proceedings of the 2015 Canadian Conference on Artificial Intelligence (Canadian AI'15)*, pages 155–166, 2015a.

L. Trottier, P. Giguère, and B. Chaib-Draa. Feature selection for robust automatic speech recognition: a temporal offset approach. *International Journal of Speech Technology (IJST)*, 18(3):395–404, 2015b.

L. Trottier, P. Giguère, and B. Chaib-draa. Multi-task learning by deep collaboration and application in facial landmark detection. *Computing Research Repository (CoRR)*, abs/1711.00111, 2017a.

L. Trottier, P. Giguère, and B. Chaib-draa. Convolutional residual network for grasp localization. In *Proceedings of the 2017 Conference on Computer and Robot Vision (CRV'17)*, 2017b.

L. Trottier, P. Giguère, and B. Chaib-draa. Parametric exponential linear unit for deep convolutional neural networks. In *Proceedings of the 16th IEEE International Conference On Machine Learning And Applications (ICMLA'17)*, 2017c.

L. Trottier, P. Giguère, and B. Chaib-draa. Sparse dictionary learning for identifying grasp locations. In *Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV'17)*, pages 871–879, March 2017d.

M. Varma and A. Zisserman. A statistical approach to material classification using image patch exemplars. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(11):2032–2047, 2009.

A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems (NIPS'16)*, pages 550–558, 2016.

Z. Wang, Z. Li, B. Wang, and H. Liu. Robot grasp detection using multimodal deep convolutional neural networks. *Advances in Mechanical Engineering*, 8(9): 1687814016668077, 2016.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

J. Xu, S. Denman, S. Sridharan, C. Fookes, and R. Rana. Dynamic texture reconstruction from sparse codes for unusual event detection in crowded scenes. In *Proceedings of the 2011 Joint ACM Workshop on Modeling and Representing Events (JMRE'11)*, pages 25–30, 2011.

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, pages 2048–2057, 2015.

Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.

J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09)*, pages 1794–1801, 2009.

S. Yang, P. Luo, C.-C. Loy, and X. Tang. Wider face: A face detection benchmark. In *Proceedings of the 2016 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 5525–5533, 2016.

Y. Yang and T. M. Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *Computing Research Repository (CoRR)*, abs/1605.06391, 2016a.

Y. Yang and T. M. Hospedales. Trace norm regularised deep multi-task learning. *Computing Research Repository (CoRR)*, abs/1606.04038, 2016b.

J. Yim, H. Jung, B. Yoo, C. Choi, D. Park, and J. Kim. Rotating your face using multi-task deep neural network. In *Proceedings of the 2015 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'15)*, pages 676–684, 2015.

X. Yin and X. Liu. Multi-task convolutional neural network for pose-invariant face recognition. *IEEE Transactions on Image Processing*, 2017.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the 2014 European Conference on Computer Vision (ECCV'14)*, pages 818–833, 2014.

C. Zhang and Z. Zhang. Improving multiview face detection with multi-task deep convolutional neural networks. In *Proceedings of the 2014 IEEE Winter Conference on Applications of Computer Vision (WACV'14)*, pages 1036–1041, 2014.

K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10): 1499–1503, 2016.

M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(8):1819–1837, 2014.

T. Zhang, B. Ghanem, S. Liu, and N. Ahuja. Robust visual tracking via multi-task sparse learning. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'12)*, pages 2042–2049, 2012.

Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *Proceedings of the 2014 European Conference on Computer Vision (ECCV'14)*, pages 94–108, 2014.

Z. Zhang, Y. Xu, J. Yang, X. Li, and D. Zhang. A survey of sparse representation: algorithms and applications. *IEEE access*, 3:490–530, 2015.