University of Liège
Faculty of Applied Sciences
Department of Electrical Engineering & Computer Science

PhD dissertation

CONTRIBUTIONS TO DEEP TRANSFER LEARNING

FROM SUPERVISED TO REINFORCEMENT LEARNING

by Matthia Sabatelli

Advisor: Prof. Pierre Geurts
December 2021

JURY MEMBERS

Matthia

Dedicated to the loving memory of my beloved father
Francesco Sabatelli ...
1960 – 2011

... and to that of my dear friend Marco Alexander Wiering.
1971 – 2021

ABSTRACT

Throughout our lifetime we constantly need to deal with unforeseen events, which sometimes can be so overwhelming to look insurmountable. A common strategy that humans as well as animals have learned to adopt throughout millions of years of evolution, is to start tackling novel, unseen situations by re-using knowledge that in the past resulted in successfull solutions. Being able of recognizing patterns across similar settings, as well as the capacity of re-using and potentially adapting an already established skill set, is a crucial component in human's and animal's intelligence. This capacity comes with the name of Transfer Learning.

The field of Artificial Intelligence (AI) aims to create computer programs that are able of mimicking at least to a certain exent the properties underlying natural intelligence. It follows that among such properties, there is also that of being capable of learning how to solve new tasks whilst exploiting some previously acquired knowledge. Within the mathematical and algorithmic AI toolbox, Convolutional Neural Networks (CNNs) are nowadays by far among the most successfull techniques when it comes to machine learning problems involving high-dimensional and spatially organized inputs. In this dissertation we focus on studying their transfer learning properties, and investigate whether such models can get transferred and trained across a large variety of domains and tasks.

In the quest of better characterizing the transfer learning potential of CNNs, we focus on two of the most common machine learning paradigms: supervised learning and reinforcement learning. After a first part (Part I) devoted to presenting all the necessary machine learning background, we will move to Part II, where the transfer learning properties of CNNs will be studied from a supervised learning perspective. Here we will focus on several computer vision tasks that range from image classification to object detection, which will be tackled by regular CNNs as well as by pruned models. Next, in part III, we will shift our transfer learning analysis to the reinforcement learning scenario. Here we will first start by introducing a novel family of deep reinforcement learning algorithms, and then move towards studying their transfer learning properties alongside that of several other popular model-free reinforcement learning algorithms.

Our transfer learning experiments allow us to identify the benefits, as well as some of the possible drawbacks that can come from adapting transfer learning strategies, while at the same time shading some light on how convolutional neural networks work.

You're never gonna grow if you don't grow now
You're never gonna know if you don't find out
You're never going back never turning around
You're never gonna go if you don't go now

ACKNOWLEDGEMENTS

Thaaank you

CONTENTS

SYMBOLS AND NOTATION

$\mathcal{X}$	input space
$\mathcal{Y}$	output space
$P(X, Y)$	joint probability distribution
$\mathcal{F}$	set of all possible functions
ℓ	loss
$\mathcal{L}$	learning set
$R(f)$	expected risk
$\hat{R}$	empirical risk
f^*	optimal function
R_B	Bayes risk
f_B	Bayes model
σ	sigmoid function
$\mathbf{w}$	weight vector
$\mathbf{W}$	weight matrix
θ	neural network parameters
$\mathscr{L}$	loss function
η	learning rate
ρ	momentum
$\circledast$	cross convolution operator
$\mathbf{o}$	feature map
$\mathcal{M}$	Markov Decision Process
$\mathcal{S}$	state space
$\mathcal{A}$	action space
$\mathcal{P}$	transition function
t	time-step
$\Re$	reward function
γ	discount factor
π	policy
$\tau = \langle s_t, a_t, r_t, s_{t+1} \rangle$	trajectory
G	goal
$V^\pi(s)$	state-value function
$Q^\pi(s, a)$	state-action value function
$A^\pi(s, a)$	advantage function
π^*	optimal policy
$V^*(s)$	optimal state-value function
$Q^*(s, a)$	optimal state-action value function

δ_t	temporal-difference error
y_t	temporal-difference target
ϵ	epislon-greedy exploration parameter
$e(t)$	eligibility traces for state s
D	experience replay buffer
$h(s, a; \theta)$	generic function approximator
$\mathcal{D}$	domain
$\mathcal{T}$	task
$\mathcal{D}_S$	source domain
$\mathcal{D}_T$	target domain
$\mathcal{T}_S$	source task
$\mathcal{T}_T$	target task
f	decison function
$f_T(\cdot)$	target predictive function
$\mathcal{K}$	knowledge
$\mathcal{K}_S$	source knowledge
$\mathcal{K}_T$	target knowledge
θ_S	source neural network parameters
θ_T	target neural network parameters
$\mathcal{X}_S$	source input space
$\mathcal{X}_T$	target input space
$\mathcal{Y}_S$	source output space
$\mathcal{Y}_T$	target output space
N_T	number of samples in target dataset
Q_T	number of classes in target dataset
$\mathscr{X}$	feature vector
θ_i	ImageNet pre-trained parameters
θ_r	Rijksmuseum pre-trained parameters
I_t	number of instruments in MINERVA
m	mask
k	late resetting epochs
$f(x; m \odot \theta_0)$	winning ticket
$f(x; m \odot \theta_r)$	random ticket
$f(x; m \odot \theta_k)$	winning ticket obtained through late resetting
$\mathbf{T}$	tensor

$\mathscr{D}$	He-Uniform weight distribution
$V(s;\Phi)$	state-value network
$V(s;\Phi^-)$	state-value target network
$Q(s,a;\theta)$	state-action value network
$Q(s,a;\theta^-)$	state-action value target network
N	capacity of the memory buffer
$\mathcal{M}_S$	source Markov Decision Process
$\mathcal{M}_T$	target Markov Decision Process
$\mathscr{R}$	area ratio score

1 INTRODUCTION

Ever since the creation of the first computers, humans started to wonder whether such complex machines could one day be able to think. Already in the 20th century, computer scientist Alan Turing, who by many is considered to be the main progenitor of computer science, designed the *Imitation Game*, an experiment devoted to test whether a machine can exhibit intelligent behaviors. While purely theoretical, as well as philosophical, his work opened the door to many questions that throughout the years would have defined the field of Artificial Intelligence (AI), a multidisciplinary field that aims to create computer programs that are able to mimic, at least in part, the cognitive abilities underlying human intelligence. In the attempt of answering Turing's everlasting question *Can machines think?*, part of the AI community started to consider an equally challenging and foundamental question: *Can machines learn?*. This question is nowadays being tackled by researchers working at the intersection of computer science, probability, statistics and even information theory and psychology, which all fall under the research field that is denoted as machine learning.

The question *Can machines learn?* might only at first sight appear to be simple and straightforward to answer, as in reality it actually forces us to define two very important concepts. First, while it is true that the term machine has to relate to computer programs, it is equally true that it gives little insight about the computational nature of such programs, which in practice can come in very different flavours as they are typically built on top of a wide range of mathematical models. Second, it also requires us to consider what it means for a computer program to *learn* and how such an ability, which by many cognitive scientists is considered to be one of the main intellectual feats underlying intelligence, is related to computer science, a discipline that at first sight might have little in common with research fields that study the human brain.

1.1 MACHINE LEARNING

According to Mitchell et al. [157] a computer program, is said to learn if it manages to improve its performance on a certain task through experience. Examples of potential tasks might be the recognition of digits in images [130], mastering a certain boardgame [226], or even the ability of predicting the outcome of a clinical trial [291]. While tasks can come in numerous flavours and can differ across eachother in terms of complexity, the way these are usually tackled by a learning

algorithm can be divided into three different paradigms. While two of these three learning scenarios will be covered in depth in the first part of this dissertation we still briefly describe them hereafter.

1. Supervised Learning: this instance of machine learning is characterized by problems where there is an input space $\mathcal{X}$ and an output space $\mathcal{Y}$ and the goal is to learn a mapping f from $\mathcal{X}$ to $\mathcal{Y}$. The computer program can learn this mapping thanks to some input-output pairs that are i.i.d. drawn from the joint probability distribution $P(X, Y)$ and that can be observed throughout the learning process. Broadly speaking, the more the program observes these input-output pairs, the more experienced it gets and therefore the more accurate its mapping function becomes.

2. Unsupervised Learning: in this scenario the learning algorithm does not have access to any output values drawn from $P(X, Y)$ and therefore cannot rely on them throughout the learning process. As a result the aforementioned mapping function f cannot be learned. Instead, unsupervised learning algorithms aim at discovering occurring patterns within the input samples they observe.

3. Reinforcement Learning: is a branch of machine learning where an agent has to learn how to interact with its surroundings, typically defined as the environment. Thanks to an interactive learning procedure, known as the agent-environment loop, the agent's goal is that of learning an optimal policy π^*, which allows it to maximize a certain reward signal over time. Differently from the two learning paradigms mentioned above, reinforcement learning is a much more dynamic and uncertain learning scenario, which builds on top of ideas stemming from a large variety of disciplines ranging from mathematical optimization to cognitive psychology.

Several learning algorithms able of dealing with the aforementioned learning scenarios exist, however, the last years have witnessed the developement of many successfull applications that are based on artificial neural networks. Among the different types of possible artificial neural networks, strong potential has been exhibited in all three machine learning areas by Convolutional Neural Networks (CNNs), a class of techniques that is particularly well suited for dealing with high-dimensional and spatially organized inputs such as images. Computer programs based on CNNs are to this date the state of the art when it comes to a large variety of machine learning tasks. In fact, CNNs are nowadays able of successfully dealing with supervised learning problems that involve the classification of natural images (see Figure 1.1), and even to learn how to play popular arcade videogames without any supervision (see Figure 1.2).

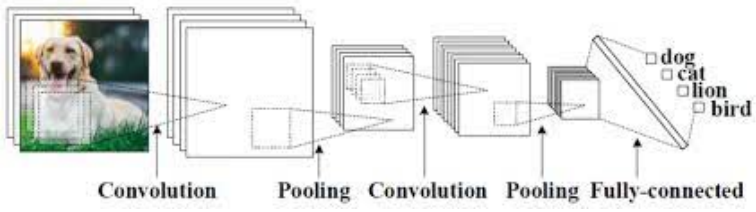


Figure 1.1: An example of a Convolutional Neural Network that gets trained for solving a supervised learning problem, namely correctly classifying the animal depicted in the image as a dog. The different components of this network will be explained in detail in Chapter 2 of this dissertation.

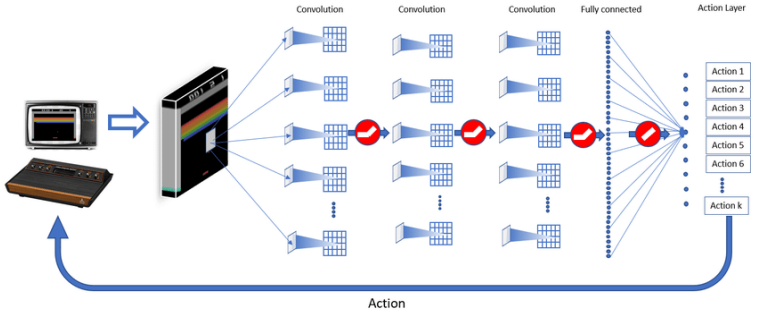


Figure 1.2: An example of a Convolutional Neural Network trained for solving a reinforcement learning problem, namely learning how to play the arcade Atari game of `Breakout`. The way the network achieves this task will be presented in detail in Chapter 3. Image courtesy of Patel et al. [178].

1.2 OBJECTIVES AND RESEARCH QUESTIONS

The typical machine learning training scenario assumes that when the learning process begins, a computer program starts solving a certain task *tabula rasa*. Going back to the examples presented in Fig. 1.1 and Fig. 1.2, this corresponds to a situation where a CNN has either never dealt with the supervised learning problem of natural image classification, or has never learned how to play any Atari videogame before facing the `Breakout` game depicted in the image. While overall CNNs are able of successfully learning how to solve a certain machine learning problem from scratch, there can however be certain situations where learning from scratch can result in sub-optimal performance or even prevent a network from learning at all. Based on this potential limitation, the work presented in this dissertation aims at exploring whether this family of neural networks can get transferred and trained across different tasks, and aims to identify which sort of benefits can arise from tackling several machine learning problems with CNNs that instead of being trained from scratch have already been pre-trained on related tasks in the past. Therefore, driven by the following general research question:

Can convolutional neural networks be transferred and trained across different source and target domains? if so, which target domains could be of interest for investigating their transfer learning properties?

we aim at better charactering the transfer learning properties of CNNs, and investigate whether this family of machine learning models can get transferred both in a supervised learning scenario as well as in a reinforcement learning one. We will help answer this research question by addressing three, arguably more specific, research questions which are:

1. *What Transfer Learning training strategy should be adopted to maximize the performance of pre-trained networks?*
2. *Can Transfer Learning be a valuable tool for better understanding convolutional neural networks?*
3. *Do different machine learning paradigms result in convolutional neural networks with different transfer learning properties?*

1.3 OUTLINE OF THE DISSERTATION

As mentioned earlier, all research questions will be studied both from a supervised learning perspective as well as from a reinforcement learning one, while at the same time leaving a transfer learning study for the unsupervised learning scenario as future work. It naturally follows that after having introduced these learning paradigms in Chapters 2, 3 and 4, the rest of this dissertation will be divided into two more parts: Part II, including Chapters 5, 6 and 7, which will focus on studying the transfer learning potential of CNNs within supervised learning; and Part III, made of Chapters 8, 9 and 10, which will focus on the reinforcement learning scenario instead. More specifically, in Part II we will focus our supervised transfer learning analysis on domains that are not defined within the realm of natural images and that see instead CNNs applied in the areas of digital heritage (Chapters 5, 6 and 7) and digital pathology (Chapter 7). Here we will be considering computer vision tasks ranging from image classification to object detection. In Part III, our transfer learning studies will consider the domain of model-free deep reinforcement learning, a research area that aims to solve optimal control problems with convolutional neural networks that need to serve as value function approximators as well as feature extractors. Here we will see how this task differs from the aforementioned computer vision problems and how it affects the overall transfer learning properties of CNNs.

1.4 PUBLICATIONS

The aforementioned research questions will be answered thanks to the work that has been presented in several peer-reviewed publications. These contributions, together with the role they play throughout this dissertation, are presented in chronological order hereafter:

Main Publications

- Sabatelli et al. [206] *"Deep transfer learning for art classification problems."* Proceedings of the European Conference on Computer Vision (ECCV) Workshops, 2018.
  This publication will be mainly reviewed in Chapter 5, and will be of interest in Chapters 6, 7 and 9.

- Sabatelli et al. [210] *Deep Quality Value (DQV) Learning."* Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop, 2018.
  This publication is of interest in Chapter 8 and 9.

- Sabatelli et al. [208] *"Approximating two value functions instead of one: towards characterizing a new family of Deep Reinforcement Learning algorithms"* Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop, 2019.
  This publication is of interest in Chapter 8.

- Sabatelli et al. [209] *"The deep quality-value family of deep reinforcement learning algorithms"* Proceedings of the International Joint Conference on Neural Networks (IJCNN). IEEE, 2020.
  This is the main publication underlying Chapter 8, which will also be of interest in Chapter 9.

- Sabatelli, Kestemont, and Geurts [207] *"On the transferability of winning tickets in non-natural image datasets.* Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP), 2021.
  This work was also presented at the first edition of the Sparsity in Neural Networks (SNN) Workshop 2021.
  Chapter 7 is based on this publication.

- Sabatelli et al. [203] *"Advances in Digital Music Iconography: Benchmarking the detection of musical instruments in unrestricted, non-photorealistic images from the artistic domain."* DHQ: Digital Humanities Quarterly 15.1 2021.
  Chapter 6 extends this publication.

- Sabatelli and Geurts [205] *"On the Transferability of Deep-Q Networks."* Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop, 2021. Chapter 9 is based on this publication.

Throughout the Ph.D. several other peer-reviewed papers have been published, however these are not directly presented in this thesis as tehy are either the result of external collaborations, or have served for reporting preliminary results of ongoing research:

Additional Publications

- Leroy et al. [133] *"QVMix and QVMix-Max: Extending the Deep Quality-Value Family of Algorithms to Cooperative Multi-Agent Reinforcement Learning."* Proceedings of the AAAI-21 Workshop on Reinforcement Learning in Games, 2021.
- Hammond et al. [82] *"Forest Fire Control with Learning from Demonstration and Reinforcement Learning"*. Proceedings of the International Joint Conference on Neural Networks (IJCNN). IEEE, 2020.
- Banar et al. [12] *"Transfer Learning with Style Transfer between the Photorealistic and Artistic Domain."* IS&T International Symposium on Electronic Imaging. Computer Vision and Image Analysis of Art, 2021.
- Sasso, Sabatelli, and Wiering [214] *"Fractional Transfer Learning for Deep Model-Based Reinforcement Learning."* ArXiv preprint arXiv:2108.06526.

Part I

PRELIMINARIES

2

SUPERVISED LEARNING AND DEEP NEURAL NETWORKS

Outline

In this first chapter, we present Supervised Learning (SL), a branch of machine learning that aims to create statistical models that, given a set of previously collected input-output observations, can predict the value of new unseen output variables. Throughout the chapter, as well as during this dissertation, we will focus on how one can represent such models through artificial neural networks, a family of algorithms that, over the last decade, has gained tremendous popularity within the artificial intelligence community. We start by providing a general overview of SL in Sec. 2.1 where we describe the main ideas behind this machine learning paradigm before characterizing it from a more mathematical perspective in Sec. 2.2. We then move on towards presenting artificial neural networks in Sec. 2.3 where we will describe how these kinds of algorithms can be used for solving SL problems, as well as how these models are trained and designed. We then present convolutional neural networks, a particular type of artificial neural network that is particularly well suited for dealing with SL problems with high-dimensional and spatially organized inputs. We will do this in Sec. 2.4, before ending the chapter with some concluding remarks in Sec. 2.5.

2.1 INTRODUCTION

Today's modern society is surrounded by technological services that aim at exploiting the power of data. It is in fact not a hard task to think of newspapers, or companies, referring to data-related terms like "Big Data" or "Internet of Things" in the context of Artificial Intelligence (AI). While their resulting articles and products are not always scientifically accurate, nor necessarily useful, it comes without a doubt that they nevertheless do focus on one critical component of today's AI toolbox. Data plays indeed a crucial role nowadays in the development of AI services, and many AI-based solutions, ranging from recommendation systems underlying our favorite streaming services to online language translators that allow us to easily translate the most exotic of the languages to our mother tongue, would not be as effective without data. Yet, what does it mean to build an AI system based on data, and why does data play such an important

role? In this chapter we will answer these questions by focusing on supervised learning, a branch of machine learning that aims at developing statistical models that are able of capturing relationships within structured datasets that come in the form of input-output pairs. The general idea underlying supervised learning algorithms is that there is a very precise and defined process governing the generation of data, which if discovered, can result in the development of successful AI-based applications. However, correctly identifying such data generation process can be particularly challenging. We will now see how one can approach this difficult, yet exciting, problem.

2.2 STATISTICAL LEARNING

We start by defining a Supervised Learning (SL) problem with a quadruplet containing the following elements [63, 143]:

- An input space $\mathcal{X}$,
- An output space $\mathcal{Y}$,
- A joint probability distribution $P(X, Y)$.
- A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$

Let us define with $\mathcal{F}$ the set of all functions f that a certain learning algorithm can produce. In SL the main goal is to find a function $f : \mathcal{X} \rightarrow \mathcal{Y} \in \mathcal{F}$ that minimizes the expectation over $P(X, Y)$ of the loss ℓ, based on the predictions made by f and the correct outputs defined in $\mathcal{Y}$.

This expectation is also known as the expected risk, or generalization error, and is defined as:

$$R(f) = \mathbb{E}_{(\mathbf{x},y)\sim P(X,Y)}\big[\ell(y, f(\mathbf{x}))\big], \tag{2.1}$$

where f is built from a limited set of observations that define the SL problem we would like to solve. Such observations constitute the learning set $\mathcal{L}$ which is defined by N pairs of input vectors and output values $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ are i.i.d. drawn from $P(X, Y)$ [68].

As $P(X, Y)$ is unknown and $\mathcal{L}$ is finite, one cannot evaluate the quantity defined in Eq. 2.1. However, one can compute an estimate of it instead. To this end it is common to use part of the learning set $\mathcal{L}$ for constructing a training set $\mathcal{L}_{\text{Train}}$, of size M, that can be used for computing the empirical risk, or training error, as follows:

$$\hat{R}(f, \mathcal{L}_{\text{Train}}) = \frac{1}{M} \sum_{(\mathbf{x}_i, y_i)\in\mathcal{L}_{\text{Train}}} \ell(y_i, f(\mathbf{x}_i)). \tag{2.2}$$

Computing Eq. 2.2 results in an unbiased estimate that can be used for finding a good approximation of the optimal function f^* that minimizes Eq. 2.1. Formally this corresponds to satisfying the equality

$$f^*_{\mathcal{L}_{\text{Train}}} = \arg\min_{f \in \mathcal{F}} \hat{R}(f, \mathcal{L}_{\text{Train}}), \tag{2.3}$$

which is known as the empirical risk minimization principle [266]. As mentioned by Vapnik and Chervonenkis [265], empirical risk minimizers should converge in the limit to optimal models:

$$\lim_{M \to \infty} f^*_{\mathcal{L}_{\text{Train}}} = f^*. \tag{2.4}$$

With these concepts in place, we can summarize the goal of SL as finding a function f that, on average, makes good predictions over $P(X, Y)$. To this end, when explaining the data generating process underlying $P(X, Y)$, f does not have to be too "simple" nor too "complex". In order to assess this, let $\mathcal{Y}^{\mathcal{X}}$ be the set of all functions $f : \mathcal{X} \to \mathcal{Y}$. The minimal expected risk over all these functions is defined as

$$R_B = \min_{f \in \mathcal{Y}^{\mathcal{X}}} R(f), \tag{2.5}$$

and is called the Bayes risk. When minimized, the quantity defined in Eq. 2.5 results in the best possible function f_B, which is called the Bayes model. If the capacity of the hypothesis space $\mathcal{F}$ chosen for finding f is too low, then it follows that $R(f) - R_B$ will be large for any $f \in \mathcal{F}$, including f^* and $f^*_{\mathcal{L}_{\text{Train}}}$. Similarly, if the capacity of $\mathcal{F}$ is too high then although $R(f^*) - R_B$ will be small, $f^*_{\mathcal{L}_{\text{Train}}}$ can fit $\mathcal{L}_{\text{Train}}$ arbitrarily well such that:

$$R(f^*_{\mathcal{L}_{\text{Train}}}) \geq R_B \geq \hat{R}(f^*_{\mathcal{L}_{\text{Train}}}, \mathcal{L}_{\text{Train}}) \geq 0. \tag{2.6}$$

When f is too simple, then it is said to underfit the data, whereas it is said to overfit it when it is too complex. As a result, one wants both the expected risk R and the empirical risk $\hat{R}$ minimizers to be as low as possible. To achieve this we can again evaluate the performance of $f^*_{\mathcal{L}_{\text{Train}}}$ by computing the empirical risk defined in Eq. 2.2 on a separate independent dataset known as the testing set $\mathcal{L}_{\text{Test}}$. Note, however, that this quantity should be used for model evaluation purposes only and not for model selection ones as this would lead to some bias. Model selection is usually done through a separate dataset called the validation set.

So far we have defined the concepts of expected risk and empirical risk with respect to a loss function ℓ. However, we have not yet seen what this loss function looks like in practice. In SL ℓ changes based on the characteristics of Y. This allows us to distinguish between two different SL problems: classification and regression. In the first case $\mathcal{Y}$ comes in the form of a finite set of classes $\{c_1, c_2, ..., c_C\}$, whereas

in the latter case $\mathcal{Y} \in \mathbb{R}$. For classification the arguably most straightforward loss function is the $0-1$ loss defined as

$$\ell(f(\mathbf{x}), y)) = \mathbb{1}(f(\mathbf{x}) \neq y), \tag{2.7}$$

while for regression problems ℓ can for example come in the form of the squared error loss:

$$\ell(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2 \tag{2.8}$$

or in the form of the absolute error loss

$$\ell(f(\mathbf{x}), y) = |y - f(\mathbf{x})|, \tag{2.9}$$

depending on how much one wants to penalize large errors made by f.

While several SL algorithms adopting the empirical risk minimization principle exist, throughout this dissertation, we will only focus on artificial neural networks, a family of techniques that will be reviewed hereafter.

2.3 NEURAL NETWORKS

Artificial neural networks are learning algorithms that have originally been developed with the goal of mimicking the neural interactions within biological systems [157]. Therefore, their primal intent was, to serve as mathematical models that could be used for better understanding biological learning processes. Despite being motivated by a well-grounded research objective, the most successful algorithms that nowadays fall within the category of artificial neural networks have, however, only been developed with the simple objective of resulting in effective empirical risk minimizers. We, therefore, note that most of the neural architectures that will be presented and studied throughout this dissertation mirror actual existing biological processes only to a very limited extent. Nevertheless, as will be explained in the coming section, they do build on top of ideas that are in line with biologically plausible artificial networks.

2.3.1 *Multilayer Perceptrons*

The first mathematical model developed with the intention of mimicking the biological processes underlying the human brain was proposed by Rosenblatt [195]. Inspired by the work of McCulloch and Pitts [150], Rosenblatt developed the perceptron, the simplest form of artificial neural network capable of tackling binary classification problems through supervised learning. Given an input vector $\mathbf{x}$ the perceptron produces the following output:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

which is given by summing up each input x with a certain weight w and a final additional bias term b. The result of this sum is then passed through the sign non-linear activation function which yields output h:

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

The way the perceptron works is visually represented in Fig. 2.1 and can be summarized by the following equation

$$f(\mathbf{x}) = \text{sign}(\sum_i w_i x_i + b), \tag{2.12}$$

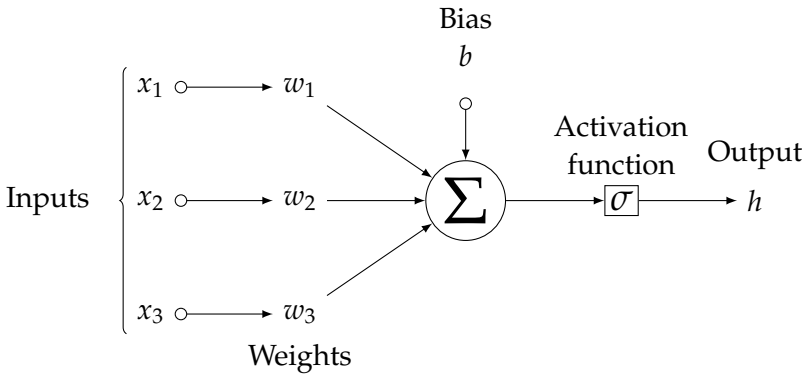


Figure 2.1: A visualization of a perceptron.

which, interestingly, can also be rewritten in terms of tensor operations. This allows us to express the perceptron classification rule as

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\intercal \mathbf{x} + b). \tag{2.13}$$

Eq. 2.13 makes it possible to conveniently visualize the mathematical operations of the perceptron through a computational graph, a directed graph where each node represents a certain mathematical operation. The computational graph of Eq. 2.13 is represented in Fig. 2.2 and can be considered as the main building block of modern artificial neural networks.

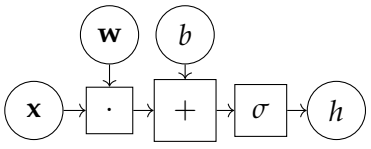


Figure 2.2: The computational graph representing the mathematical operations performed by the perceptron represented in Fig. 2.1 and defined by Eq. 2.13.

Eq. 2.13 summarizes the computations that are performed by one single input where $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$. However, the computation capabilities of such a single unit are very limited and can rarely be adopted to solve complex tasks. To overcome this, one can stack several units in parallel such that they create a layer with q outputs defined as:

$$\mathbf{h} = \sigma(\mathbf{W}^\intercal \mathbf{x} + \mathbf{b}), \tag{2.14}$$

where $\mathbf{h} \in \mathbb{R}^q, \mathbf{x} \in \mathbb{R}^p, \mathbf{W} \in \mathbb{R}^{p \times q}, b \in \mathbb{R}^q$. To increase the flexibility and capabilities of the model even further, one can then compose a sequence of L layers

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{x} \\ \mathbf{h}_1 &= \sigma(\mathbf{W}_1^\intercal \mathbf{h}_0 + \mathbf{b}_1) \\ &... \\ \mathbf{h}_L &= \sigma(\mathbf{W}_L^\intercal \mathbf{h}_{L-1} + \mathbf{b}_L) \end{aligned} \tag{2.15}$$

that define a multilayer perceptron (MLP), also known as feedforward neural network. From now on we will refer to an MLP as $f(\mathbf{x}; \theta)$ where $\theta = \{\mathbf{W}_k, \mathbf{b}_k | k = 1, ..., L\}$.

Now that we defined the mathematical computations that are performed by a feedforward neural network we move on to explaining how one can train these kinds of models to perform empirical risk minimization.

2.3.2 *Stochastic Gradient Descent*

Training a neural network consists in finding parameters θ such that a loss function $\mathscr{L}(\theta)$, also denoted as the objective function, is minimized. Such loss functions are typically expressed as a sum of the losses ℓ_n incurred by each sample n in a training set of size N, and can be expressed in the following form:

$$\mathscr{L}(\theta) = \sum_{n=1}^{N} \ell_n(\theta). \tag{2.16}$$

When neural networks are used, $\mathscr{L}$ has to be differentiable as this allows to minimize it through first-order optimization algorithms. Among such methods, the arguably most straightforward one is gradient descent, which updates the parameters θ proportionally to the negative gradient of $\mathscr{L}$. This is done by applying the following update rule:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \eta_t (\nabla \mathscr{L}(\theta_t))^\intercal \\ &= \theta_t - \eta_t \sum_{n=1}^{N} (\nabla \ell_n(\theta_t))^\intercal, \end{aligned} \tag{2.17}$$

where t is a time counter variable, and $\eta \geq 0$ is the learning rate, sometimes also denoted as the step-size parameter. We can easily observe that computing Eq. 2.17 can become computationally very expensive as it requires evaluating gradients from all individual functions ℓ_n. In fact, this property defines gradient descent as a batch optimization method, which makes it unfortunately unsuitable for dealing with large datasets. A possible solution to this computational burden consists in reducing the amount of computations required by the sum in Eq.2.17 by simply considering a small, random batch of samples of the training set. In extreme cases, one can even just estimate the gradient on one single, randomly chosen training sample, a method called Stochastic Gradient Descent (SGD). While it is true that this approach gives an unbiased estimate of the true gradient, its estimate can also be very noisy, which is the reason why it is preferable to evaluate the gradient for a mini-batch of samples rather than on one single unique sample. A large body of work has investigated the effect that the batch size has on neural network training [110, 111, 189]; however, so far, no exact rule for determining an optimal batch size exists. Yet, provided that enough computational resources are available, large mini-batches are usually preferred as they will result in more accurate estimates of the gradient, and therefore reduce the variance in the parameter update θ_{t+1}.

When the optimization surface is made of valley floors, gradient descent has the limitation of being very slow. To deal with this issue, several works have designed optimization strategies which make gradient based optimization faster and more efficient. The most straightforward improvement to the gradient descent algorithm is the one proposed by Rumelhart, Hinton, and Williams [198] who suggested the use of an additional term in the update rule presented in Eq. 2.17, named momentum. This term, simply keeps track of what happened when the parameters were updated at $t-1$ and determines the next parameter update as a linear combination between the current and previous gradients. This results in the following update rule:

$$\theta_{t+1} = \theta_t - \eta_t(\nabla \ell(\theta_t))^\intercal + \rho \Delta \theta_t \tag{2.18}$$

where

$$\begin{aligned} \Delta \theta_t &= \theta_t - \theta_{t-1} \\ &= \rho \Delta \theta_{t-1} - \eta_{t-1}(\nabla \ell(\theta_{t-1}))^\intercal, \end{aligned} \tag{2.19}$$

and $\rho \in [0, 1]$. Momentum is well-known to overall accelerate the optimization process while also allowing the algorithm to average out noisy estimates of the gradient.

Next to adding a momentum term to improve the performance of gradient descent, another common method that can accelerate its convergence revolves around dynamically adapting the learning rate parameter η. Popular neural network optimizers such as `RMSProp` [252],

`AdaGrad` [52], and the very well-known `Adam` optimizer [113], all adopt this method. While discussing these algorithms in detail is out of the scope of this thesis, we refer the reader to the work of Ruder [197], which provides a nice overview of the most common gradient descent optimization algorithms, and to the work of Schmidt, Schneider, and Hennig [218] who empirically evaluate their performance across different networks and machine learning problems.

Before ending this section, it is worth noting that there are several alternative algorithms besides SGD-like methods that can be used for optimization problems. Among such methods, we mention second-order optimization techniques such as Newton, Quasi-Newton, and the Conjugate gradient methods discussed in [246]. While these algorithms are able to minimize the empirical risk faster and even better than SGD, they do not result in equally good generalization performance. Recall from Sec. 2.2, that in SL, minimizing the expected risk is just as important as minimizing the empirical risk, which is a property that the aforementioned second-order optimization algorithms do not have. This key result, first presented by Bottou and Bousquet [25], is what motivates the use of SGD-like optimizers in deep learning.

2.3.3 *Backpropagation*

From Eq. 2.19 we can note that a crucial role in the optimization process is played by the gradient $\nabla\ell(\theta)$. As we have seen in Sec. 2.3.1 neural networks can be considered as a composition of nested functions $\mathbf{h}_k$ for $k = 1, ..., L$, where each function comes with its own parameters θ_k. Therefore the gradient comes in the form of a vector which contains all the partial derivatives of the loss ℓ with respect to the weights θ that parametrize the neural network:

$$\nabla\ell(\theta) = \Big[\frac{\partial\ell}{\partial\theta_1}(\theta), ..., \frac{\partial\ell}{\partial\theta_L}(\theta)\Big]. \tag{2.20}$$

As the number of functions increases, so does the complexity of the gradient; therefore, an efficient way of calculating it is necessary. The backpropagation algorithm [30, 139, 198] is a special case of a more general technique, called automatic differentiation (see [15] for a general review about the topic), that allows evaluating the gradient of complicated functions numerically and automatically. This is done by exploiting the chain rule, which can be applied recursively on the computation graph that keeps track of all the arithmetic operations that are performed by the network.

To this end, let us define a simplified version of a two hidden layer perceptron f that is parametrized with weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$. When given input data $\mathbf{x}$ the network produces a prediction $\hat{y}$ which results from traversing the computational graph represented in Fig.

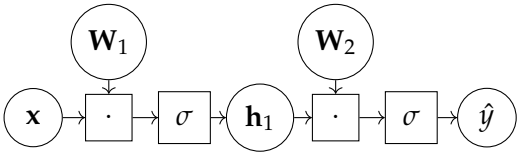


Figure 2.3: The computational graph representing a simplified version of a multi-layer perceptron with one hidden layer. Note that no bias term is added after multiplying $\mathbf{x}$ and $\mathbf{h1}$ by $\mathbf{W_1}$ and $\mathbf{W}_2$ respectively.

2.3. During the traversal, also known as the forward pass, the result of each mathematical operation is stored within its own output variable, denoted $\mathbf{u}_i$ (see Fig. 2.4). Having such an annotated graph it is now

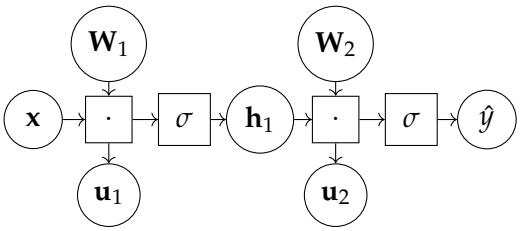


Figure 2.4: The computational graph that results after having performed one forward pass through the network. We can see that the result of each mathematical operation is stored within a new node $\mathbf{u}$ that will be necessary for computing the partial derivatives required to perform stochastic gradient descent.

possible to compute all partial derivatives efficiently by traversing the graph backwards (backward pass [139]), and by applying the chain rule which in its general form states that:

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta_i} = \sum_{k \in \text{parents}(\ell)} \frac{\partial \ell}{\partial u_k} \frac{\partial u_k}{\partial \theta_i}. \tag{2.21}$$

Therefore taking as example $\mathbf{W}_1$, the derivative of the network's output $\hat{y}$ with respect to this weight matrix is given by:

$$\begin{aligned} \frac{\mathrm{d}\hat{y}}{\mathrm{d}\mathbf{W}_1} &= \frac{\partial \hat{y}}{\partial u_2} \frac{\partial u_2}{\partial h_1} \frac{\partial h_1}{\partial u_1} \frac{\partial u_1}{\partial \mathbf{W}_1} \\ &= \frac{\partial \sigma(u_2)}{\partial u_2} \frac{\partial \mathbf{W}_2^\intercal h_1}{\partial h_1} \frac{\partial \sigma(u_1)}{\partial u_1} \frac{\partial \mathbf{W}_1^\intercal \mathbf{x}}{\partial \mathbf{W}_1}. \end{aligned} \tag{2.22}$$

2.3.4 *Loss Functions*

Defining an appropriate loss function is a task that has important practical implications for the design of the neural architecture. As for any other type of machine learning model, the choice of which loss function to minimize depends on the SL task we would like to solve. Fortunately, since neural networks are parametric models, their

loss functions are not too different from those typically used by, e.g., linear models such as logistic regression. The most important concept underlying the loss functions used by neural networks is that of maximum likelihood estimation. As many other parametric models, neural networks implicitly define a distribution $p(Y|\mathbf{x};\theta)$. This is convenient as it makes it possible to exploit the cross-entropy between the training data and the model's predictions. Therefore, no matter whether we are dealing with a classification problem or a regression one, the loss function that a neural network will adopt will always come in the following general form:

$$\mathscr{L}(\theta) = -\mathbb{E}_{(\mathbf{x},y)\sim P(X,Y)} \log p_{\text{model}}(Y|\mathbf{x}). \tag{2.23}$$

Typical loss functions that derive from Eq. 2.23 (see Chapter 5 of [73] for the exact derivations) are the mean squared error (MSE) loss

$$\mathscr{L}(\theta) = \frac{1}{2}\mathbb{E}_{(\mathbf{x},y)\sim P(X,Y)}||y - f(\mathbf{x};\theta)||^2 \tag{2.24}$$

which is used for tackling regression problems, and the categorical cross-entropy loss

$$\mathscr{L}(\theta) = -\mathbb{E}_{(\mathbf{x},y)\sim P(X,Y)} \sum_{i=1}^{C} 1(y=i) \log p_{\text{model}} f_i(\mathbf{x};\theta) \tag{2.25}$$

which is used for multi-class classification problems, where C is the number of classes we would like to classify.

Based on whether Eq. 2.24 or Eq. 2.25 is minimized, the final layer of a neural network comes in different forms. As the goal of a regression problem is to predict a single numerical value, it follows that the final layer simply consists of one individual unit that is necessary for estimating $\mathcal{Y} \in \mathbb{R}$. One single output unit is also used for binary classification problems, where it is combined with the sigmoid activation function

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{2.26}$$

which allows to model a Bernoulli distribution over a binary variable. For classification problems, where $C > 2$, and the goal is to represent the distribution over a discrete variable that can have C possible values, the sigmoid function can be generalized to a softmax function by producing a vector $\mathbf{z}$ for $i = 1, ..., C$ such that:

$$\text{Softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)}. \tag{2.27}$$

We can observe that Eq. 2.27 makes the log probabilities z_i, typically estimated by the second-last layer of a network, positive and sum up to one, therefore successfully modeling a multinoulli distribution.

While the aforementioned output layers are arguably among the most popular ones it is worth noting that several other types of output layers exist [74, 75]. Since throughout this dissertation none of these layers will be used in practice, we will not describe them here and refer the reader to Chapter 6 of [73] for more information about this topic.

2.3.5 *Vanishing Gradients and Activation Functions*

A typical problem of neural networks that come with many hidden layers is vanishing gradients. Recall from Sec. 2.3.3 that in order to perform SGD, we first need to collect all the partial derivatives of the network output with respect to its parameters. As we do this by applying the chain rule, this can have the drawback of making the gradient decrease exponentially with respect to the depth of the network. As a result, deeper layers can become particularly hard to train since no information necessary for updating the respective weights will be contained within the gradient. The most common cause of this problem is the activation function used for introducing non-linearity across the network. For example, let us consider the sigmoid function presented in Eq. 2.26 and its derivative, which comes in the following form:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x}(x) = \sigma(x)(1-\sigma(x)). \tag{2.28}$$

As we can see from the first image of Fig. 2.5, the maximum value of Eq. 2.28 is 0.25. If we then use this value when adopting the chain rule as done in Eq. 2.22, and assume the network comes with a large number of hidden layers, it is easy to see that the gradient $\frac{\mathrm{d}\hat{y}}{\mathrm{d}\mathbf{W}_1}$ will shrink to zero as the number of layers increases. The sigmoid function is not the only activation function which suffers from this phenomenon, which is also not restricted to feedforward neural networks only. In fact as first presented by Hochreiter and Schmidhuber [97], another non-linear activation function that suffers from the vanishing gradient problem is the hyperbolic tangent

$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}. \tag{2.29}$$

As can be seen in the second plot of Fig. 2.5, the tanh is very similar in shape to the sigmoid. This activation function is largely used within Recurrent Neural Networks (RNNs), a particular type of neural network that can be unfolded into very deep MLPs. Its vanishing gradient issue has put the potential of RNNs into question for many years until it was solved thanks to the introduction of the Long Short Term Memory (LSTM) cells [97].

Another solution to the vanishing gradient problem is to use the Rectified Linear Unit (ReLU) activation function (represented in green

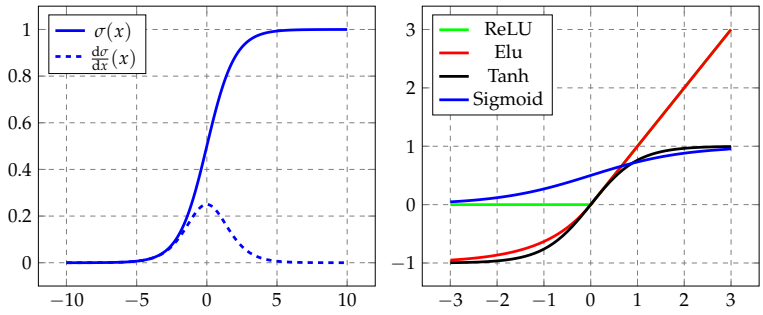


Figure 2.5: In the left plot a visualization of the vanishing gradient problem that can come from using a sigmoid non-linear activation function throughout a network. In the right plot a representation of typical non-linear activation functions within the $[-3, 3]$ range that are currently used by popular neural architectures.

in Fig. 2.5), which is arguably the most popular choice when it comes to the design of deep neural networks. This activation function is simply defined as

$$\text{ReLU}(x) = \max(0, x). \tag{2.30}$$

Its derivative has the appealing property of staying constant to 1 whenever a unit is activated as defined by:

$$\frac{\mathrm{d}}{\mathrm{d}x}\text{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.31}$$

However, a potential drawback of the ReLU is that whenever its input is negative, gradient-based methods could not be used for learning, as the unit will have a value of 0. To overcome this, several activation functions that generalize the ReLU to negative inputs have been proposed within the literature [42, 89, 145], among which we mention the Elu [42] that is visually represented in red in the right plot of Fig. 2.5.

2.4 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a family of artificial neural networks that are particularly well suited for problems involving spatially organized inputs such as (2D) images or (3D) videos. This kind of data, in fact prohibits the use of the multi-layer perceptrons presented in Sec. 2.3.1, as it requires representing images as unstructured vectors, which is a process that, for obvious computational reasons, is not feasible. Furthermore, MLPs present some additional limitations. First and foremost, due to their fully connected structure, they do not involve any sort of parameter sharing across the network. Second, as the output of each unit in a layer is given as input to all

the units in the subsequent layer, the interaction among neurons is also extremely dense. CNNs address these limitations by exploiting sparse weight sharing strategies that result in neural networks that are significantly more memory, computationally and statistically efficient.

2.4.1 *Mathematical Operations*

As their name suggests, the key mathematical operation behind CNNs is that of convolution. In one dimension, a convolution operation is performed over two arguments: an input vector $\mathbf{x} \in \mathbb{R}^W$, and a kernel $\mathbf{u} \in \mathbb{R}^w$. Its output is a new vector of size $W - w + 1$ such that:

$$(\mathbf{x} \circledast \mathbf{u})[i] = \sum_{m=0}^{w-1} x_{m+i} u_m, \tag{2.32}$$

where $\circledast$ technically denotes the cross-correlation operation, namely a convolution operation that does not flip the kernel. The process described in Eq. 2.32 can easily be generalized to multi-dimensional tensors such as images which can in fact be seen as three-dimensional tensors $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$, of width and height W and H respectively, defined over the RGB color domain ($C = 3$). Similarly, one can also define a three-dimensional kernel $\mathbf{u} \in \mathbb{R}^{C \times h \times w}$ whose purpose is to slide over the input tensor $\mathbf{x}$ and which yields a two-dimensional output tensor $\mathbf{o}$ of size $(H - h + 1) \times (W - w + 1)$ that is computed as follows:

$$\begin{aligned} \mathbf{o}_{i,j} &= \mathbf{b}_{i,j} + \sum_{c=0}^{C-1} (\mathbf{x}_c \circledast \mathbf{u}_c)[i,j] \\ &= \mathbf{b}_{i,j} + \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,n+i,m+j} \mathbf{u}_{c,n,m}, \end{aligned} \tag{2.33}$$

where $\mathbf{b}$ ($\in \mathbb{R}^{h \times w}$) and $\mathbf{u}$ are learnable parameters. Within the deep learning literature, $\mathbf{o}$ is also referred to as a feature map [73].

Note that by adopting a convolution approach, one input unit in the network only affects as many output units as defined by the size of the kernel, which improves the computational efficiency of the network greatly. Furthermore, each member of the kernel is used across the entire image, which means that the parameters that define a convolution operation are shared alongside the different locations that are visited by $\mathbf{u}$. The way the kernel interacts with its respective tensor is usually defined by two additional components that both play an important role in the design of convolutional networks. The first of these components is padding which is a technique that adds some extra values around the perimeter of the input tensor $\mathbf{x}$, to preserve the information that is depicted around its corners. Second, there is the concept of strides which defines by how many elements at a time

we wish to slide $\mathbf{u}$ over $\mathbf{x}$. As the goal of CNNs is to downsample the input tensor in a computationally efficient manner, it is usually good practice to have strides larger than one, albeit this comes at the cost of extracting features not exhaustively.

Convolutional networks typically perform several convolutions in parallel, as multiple kernels are used. The output of each convolution is then passed through a non linear activation function such as the ones that we represented in Fig. 2.5. To downsample the resulting feature maps even further, a pooling function is usually adopted. Its idea is to summarize the output of the convolving process at a certain location of the feature map through a summary statistic. This reduces its size while at the same time preserves the presence of the detected features. There are two common pooling operations one can choose from: max-pooling [294], which, given a three dimensional tensor $\mathbf{x} \in \mathbb{R}^{C\times(rh)\times(sw)}$, produces a tensor $\mathbf{o} \in \mathbb{R}^{C\times r\times s}$ by simply keeping the maximum value of a feature map within a certain rectangular neighborhood of size $h \times w$ such that

$$\mathbf{o}_{c,j,i} = \max_{0\leq n<h, m<w} \mathbf{x}_{c,rj+n,si+m}, \tag{2.34}$$

and average pooling, which instead computes the mean of a feature map such that

$$\mathbf{o}_{c,j,i} = \frac{1}{hw} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{c,rj+n,si+m}. \tag{2.35}$$

Besides reducing the size of a feature map, pooling operations also have the important benefit of making the representations learned by the network invariant to small translations. In fact, one could translate the input by a small amount and still obtain the same output after pooling. Note, however, that albeit desirable in most cases, there are situations where adopting pooling strategies should be avoided [23, 204].

2.4.2 *Popular Architectures*

With all these concepts in place we can now define the general structure of a convolutional neural network. These models follow a general pattern, originally described in [129], which is in principle very simple: an input tensor is processed by the aforementioned convolution operation, which is done many times in parallel, as different kernels are typically used. The resulting feature map is then given as input to one of the non-linear activation functions described in Sec. 2.3.5, among which the ReLU is by far the most popular choice, as it allows controlling the vanishing gradient problem. The resulting feature map is then reduced by performing one of the aforementioned pooling operations. This process of convolving + ReLU + pooling is

repeated several times, until the feature map is small enough to be reduced to a feature vector. This feature vector is finally processed by either a multilayer perceptron, or directly by the last output layer of the network, which as described in Sec. 2.3.4, changes with respect to the SL problem we would like to solve. While this general principle has arguably barely changed over the last two decades, it is worth noting that several design choices have been proposed over the years with the aim of creating better performing, and increasingly more efficient models. We will review some of the most important ones hereafter.

Image Classification Networks The first successful application of a convolutional neural network dates back to 1998, when LeCun et al. [129] introduced `LeNet-5`, a 5-layer deep network which achieved state-of-the-art results on the MNIST handwriting recognition benchmark. Despite its success however, convolutional neural networks did not gain much popularity for over ten years. In fact, the largely limited computational resources of the time, prevented them to successfully tackle image classification tasks more complicated than the ones defined by the aforementioned MNIST dataset. Only in 2012, with the introduction of `AlexNet` [119], did convolutional networks start to grab the spotlight within the computer vision community. The work of Krizhevsky, Sutskever, and Hinton [119], resulted in an 8-layer convolutional network, which combined with a 3-layer multilayer perceptron, achieved state-of-the-art results on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a popular computer vision dataset which we will review in more detail in Chapter 4. Among the main contributions of their work we mention the first results reporting the possibility of applying convolutional networks to more complicated computer vision tasks, and the possibility of training these models in a distributed fashion by exploiting, at least partially, the benefits of parallel computing. The advent of better specialized hardware, among which we mention the development of increasingly more powerful Graphical Processing Units (GPUs), together with the successful results obtained by `AlexNet`, convinced the machine learning community to explore the potential benefits of convolutional networks further. The most promising line of work was certainly pioneered by Oxford's University Visual Geometry Group (VGG) which investigated whether deeper networks could yield better performance. Their `VGG16` and `VGG19` models [228], of depth 16 and 19 respectively, showed that this was indeed the case, a design choice which combined with the use of smaller kernels allowed these models to outperform `AlexNet`. Similar results were almost concurrently achieved by Szegedy et al. [243] who introduced `GoogLeNet`, a convolutional network that uses the notion of Inception-Blocks, a specific form of convolutional layer which simultaneously uses ker-

nels of different sizes. Among these kernels we mention the use of 1×1 convolutions which have the appealing benefit of acting as a powerful memory reduction technique. While all these networks are certainly deeper than `LeNet-5`, their number of hidden layers remains on average around a dozen. Despite adopting ReLU activation functions, all the aforementioned networks do still happen to suffer from the vanishing gradient problem. He et al. [90] successfully addressed this limitation by introducing the concept of residual blocks and skipped connections. They propose using the output of one convolutional layer l, not only as input to the immediate subsequent layer $l + 1$, but also to some of the subsequent layers e.g., $l + 2$ and $l + 3$. This simple, yet very effective trick, allowed He et al. [90] to build `ResNets`, convolutional networks consisting of up to 152 layers which significantly outperformed `GoogLeNet`. Huang et al. [101] built on top of their ideas and introduced `DenseNets`, which take the concept of skipped connections to one level further, by designing models where each layer in the network takes as input the feature maps computed by all the predecessor layers. While the models presented so far are arguably the most popular ones, as they outperformed each other over the years when the ILSVRC was still an on-going yearly competition, it should be noted that many more, equivalently successful networks have been proposed over the years. Among such networks we mention `Inception-ResNets`, which combine inception and residual blocks [242] and `MobileNets` [99, 212] and `EfficientNet` [247], which are models that are specifically built for minimizing inference time on devices with limited hardware capabilities.

Beyond Image Classification So far we have only described convolutional architectures that are specifically designed for tackling the machine learning task of image classification; however, many other supervised learning problems involving high dimensional inputs exist. When it comes to the field of computer vision, lots of research attention has been given to the tasks of object detection and object segmentation. It naturally follows that several neural architectures, specifically designed for such problems, have been introduced over the years. When it comes to object detection, namely the task of locating and classifying existing objects in an image [105], convolutional neural networks are typically divided into two different families: models that adopt a region proposals based framework, and models that directly tackle the problem as a combination of classification and regression. The first family of techniques requires a network to go through two distinctive stages before being able to detect objects within images: a fisrt stage that aims at proposing candidate object bounding boxes, and a second, later stage, that classifies them. The most representative model that adopts region proposals is arguably `R-CNN` [70] which, as described by Jiao et al. [105], was among the

very first works to show that convolutional neural networks could successfully be used for object detection tasks, and that this type of architectures could perform better than computer vision approaches based on HOG-like features [105]. One of the main issues of R-CNN, and of most models that are based on region proposals, is their particularly high inference time. As a result, several improvements to this architecture have been introduced over the years, among which we mention Fast R-CNN [194], Faster R-CNN and Mask R-CNN [88], which all managed to significantly reduce inference time as well as the total training time that was originally required by R-CNN. When it comes to networks that do not involve region proposals, the arguably more popular, and by far successful, architecture is YOLO, whose first version was introduced by Redmon et al. [191]. As we will see in Chapter 6, YOLO treats the task of object detection as a regression problem, which overcomes the need of having two separate models that are respectively responsible for object localization and object classification. As no region proposals have to be estimated, YOLO-like networks [102, 191, 192] are typically more time efficient compared to Fast-CNN based methods, although this can sometimes come at the price of overall worse performing detectors [105].

When it comes to object segmentation, namely the task of classifying each pixel in an image according to its semantic label, several deep learning-based approaches also exist. These approaches are however not only limited to the use of convolutional neural networks exclusively [156], and can therefore also include architectures such as RNNs, Generative Adversarial Networks and Graphical Models. As the task of object segmentation will not be considered throughout this dissertation we will not discuss these approaches here, but refer the reader to a recent survey about the topic instead [156].

2.5 CONCLUSION

In this chapter we have introduced supervised learning and seen what it means to build statistical models that can capture the interaction between input-output observations. We have specifically focused on algorithms that come in the form of artificial neural networks as this is the type of learning technique that, to this date, are among the most successful ones, in particular for computer vision tasks. Interestingly, their learning capabilities are not limited to supervised learning problems only, which means that artificial neural networks can also be used for machine learning tasks that do not strictly require building an empirical risk minimizer. Among such problems, we mention the ones modeled in reinforcement learning, a branch of machine learning that throughout this dissertation will receive as much attention as supervised learning. We will present reinforcement learning in the next chapter.

3

REINFORCEMENT LEARNING AND DEEP NEURAL NETWORKS

Outline

This chapter introduces the research field of Reinforcement Learning (RL) and presents how its algorithms can successfully be combined with neural networks. The successful marriage between RL and deep neural networks comes with the name of Deep Reinforcement Learning (DRL) and builds on top of research that dates back to a time when training neural networks was not the common practice it is nowadays. We start by providing a general introduction to the field of RL in Sec. 3.1 where we describe the main objectives of this machine learning paradigm and see how it differs from the supervised learning setting that we have described in the previous chapter. We then present the mathematical framework that underpins the development of RL algorithms in Sec. 3.2, 3.3 and 3.4. In Sec. 3.5 and Sec. 3.6 we describe how one can create RL algorithms and why it is desirable to integrate the resulting algorithms with neural networks. In Sec. 3.7 we describe the field of DRL and introduce some of the most popular techniques that have been proposed over the years. This chapter ends with Sec. 3.8 where we discuss one of the main challenges that currently characterizes DRL and that has served as inspiration for the research that will be presented in Chapters 8 and 9 of this dissertation.

3.1 INTRODUCTION

In Chapter 2, we have described Supervised Learning (SL), a machine learning framework that aims at constructing models which can answer statistical questions about data coming in the form of input-output pairs. When these models are built successfully, it is possible to use them to make predictions about the behavior of new unseen data. Training SL models is a process which from some perspective is very static. Datasets are divided into training, validation, and testing sets, and besides providing a model with a large set of samples drawn from these datasets, there is no real interaction between the learning algorithm and the data that drives the learning process. In Reinforcement Learning (RL), this changes drastically. The goal is not to learn a mapping between a set of fixed input samples and their respective targets, but to train an algorithm that learns how to interact

with an environment. RL is, therefore, a much more dynamic learning paradigm, where the concept of time is omnipresent and is critical for the development of algorithms that not only need to solve a specific problem, but additionally, also have to be able to adapt themselves while training progresses.

In RL, a learning algorithm is usually called the agent, and it can come in numerous forms: it can range from being a self-driving car that needs to learn how to drive; to a recommendation system whose goal is to propose products to users navigating the web. More generally, we define an RL agent as any system that, given a specific situation, has to choose which action to perform. However, there is one more additional component that makes RL the challenging machine learning setting it is. It is not enough for an agent to just learn how to interact with the environment, it is even more desirable for it to learn an interaction which can be defined as "intelligent". Going back to the self-driving car example, an "intelligent" agent would not only be a car that can drive autonomously, but a car that is also able to do this while complying with the driving code. Because of this concept of learning how to make (intelligent) decisions while interacting, the problems tackled by RL algorithms are also reffered to as optimal decision making problems, which are also the target of research fields other than machine learning, such as control theory. Interestingly, both worlds try to solve the same set of problems, one by tackling them through algorithms that are denoted as "intelligent", while the other through the development of algorithms that are "optimal." Throughout this dissertation, we will not make a clear distinction between these two worlds and will assume that algorithms yielding intelligent behaviors also result in optimal behaviors. Nevertheless, we encourage the reader that has finished reading this chapter to assess whether acting optimally necessarily coincides with acting intelligently.

3.2 MARKOV DECISION PROCESSES

Before starting to develop RL algorithms for sequential decision making problems, we formulate the problem within the mathematical framework of Markov Decision Processes (MDPs) [186, 187]. Throughout this dissertation, we will characterize MDPs, and the resulting RL concepts, by using the mathematical notation that was used by Sutton and Barto [239] in their seminal book about RL, although it is worth noting that within the literature, different formulations can be found for expressing the same kind of concepts [20–22, 31].

We start by introducing the following elements:

- A set of possible states $\mathcal{S}$, that can be visited by an agent while it is interacting with the MDP, where $s_t \in \mathcal{S}$ denotes the state being visited at time-step t.

- A set of possible actions $\mathcal{A}$ that are available to the agent when it is in a certain state, where $a_t \in \mathcal{A}(s_t)$ denotes the action that is performed by the agent in state s at time-step t.
- A transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ that defines the probability for an agent to visit state s_{t+1}, based on its current state and the action which will be performed thereafter.
- A reward function $\Re : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ which returns a reward signal r_t when an agent performs action a_t in state s_t and transits to s_{t+1}.
- A discount factor denoted as $\gamma \in [0, 1]$ (explained in Section 3.3).

Based on these concepts a MDP is defined by the following tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \Re, \gamma \rangle$ and is also commonly denoted in the RL literature as the environment. The way the agent interacts with this environment is given by its policy π, defined as a probability distribution over $a \in \mathcal{A}(s)$ for each $s \in \mathcal{S}$:

$$\pi(a|s) = \Pr\{a_t = a | s_t = s\}, \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{3.1}$$

Policies can be deterministic if $\forall s : \pi(a|s) = 1$ for exactly one $a \in \mathcal{A}(s)$ and $\pi(b|s) = 0$ for all $b \in \mathcal{A}(s) \setminus \{a\}$. A policy is also stationary if it does not change over time.

The elements of the MDP allow us to properly model the dynamics of an agent interacting with its environment, an interaction which can be summarized as follows: at each time-step t the environment provides the agent with a certain state s_t, the agent then performs action a_t which results into the reward signal r_t. After performing such action the agent will enter into a new state s_{t+1}. This continuous interaction with the environment is also known as the Reinforcement Learning loop, and can technically be infinite. However, this is never the case in practice, as an agent will eventually visit a state (denoted as terminal) that only transits to itself, which will therefore stop the agent-environment interaction. We visually represent the Reinforcement Learning loop in Fig. 3.1.

Each interaction of the agent with the environment is defined as an episode, which consists of one or several trajectories τ that come in the form of the following sequence:

$$\langle (s_t, a_t, r_t, s_{t+1}) \rangle, t = 0, \ldots, T - 1 \tag{3.2}$$

where T is a random variable representing the length of the episode.

A key property of the environment is that it fulfills the Markov property which is defined as follows:

Definition 1 *A discrete stochastic process is Markovian if the conditional distribution of the next state of the process only depends from the current state of the process.*

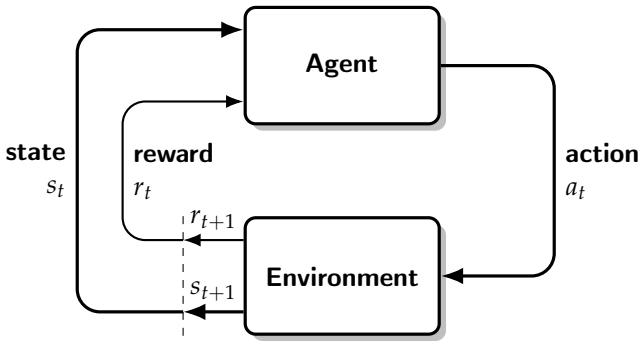


Figure 3.1: A visual representation of how an agent interacts with an environment as modeled by a Markov decision process. Figure inspired by page 48 of the Sutton and Barto [239] textbook.

This implies that the only information that is necessary for predicting to which state an agent will step next are s_t and a_t, a concept which can be expressed formally as:

$$p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = p(s_{t+1}|s_t, a_t). \tag{3.3}$$

Interestingly, the same property is also assumed for the reward that the agent will get, meaning that the reward that an agent obtains is only determined by its previous action, and not by the history of all previously taken actions, as defined by:

$$p(r_t|s_t, a_t, \ldots, s_1, a_1) = p(r_t|s_t, a_t). \tag{3.4}$$

3.3 GOALS AND RETURNS

So far, we have defined all the elements that model an agent's interaction with an environment while introducing some of its fundamental properties. However, we do not yet know what the purpose of this interaction is. In RL, an agent's goal is defined with respect to the reward signal r_t that is returned by the reward function $\Re$ and is very straightforward: maximizing the total amount of reward it receives while interacting with the environment. In the simplest case, we can define this as:

$$G_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_T. \tag{3.5}$$

While simple and intuitive, this formulation has one major drawback: it treats each reward signal equally as it does not distinguish rewards that are obtained in the near future, r_t, from the ones that will be obtained in the more distant future, r_{T-1}. To deal with this issue, we need an additional concept known as discounting, and that is governed by the discount rate parameter γ, also known as the discount factor. γ allows us to weight the different reward signals based on how close or distant in the future these rewards are received by the

agent. By introducing γ in Eq. 3.5 we can now define the expected discounted return as:

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \end{aligned} \tag{3.6}$$

The role of γ can be interpreted as follows: a reward obtained k time steps in the future is only worth γ^{k-1} times what it would be worth if received immediately. It is easy to see how different γ values can result in different agent's behaviors. If $\gamma = 0$ an agent will only take into account immediate rewards, therefore aiming to maximize r_{t+1} only and resulting into having a "myopic" behavior. If γ approaches 1 the agent will become more "far-sighted", it will take future rewards into account more strongly and will therefore increase its chances of accessing rewards that will result into a higher cumulative return. Please note that by defining $\gamma < 1$, we can make the infinite sum presented in Eq. 3.6 finite as long as the rewards r_k are bounded.

While the role of γ is often taken for granted within the RL literature, it is worth noting that as mentioned by Hessel et al. [94] and Schmidhuber [217], γ is an artificial concept that is not present in fields such as traditional control theory or engineering. This is because γ corresponds to a concept that does not exist in the real world and that in practice distorts the actual value of r_t in an exponentially shrinking fashion. Even if it is considered standard practice to include a discount factor in the development of RL algorithms, it is worth noting that making γ part of the RL framework corresponds to including a form of "inductive bias" within the resulting algorithms. It is common knowledge that low discount factors result in poor performance and that it is therefore beneficial to set γ as close to 1, yet choosing an appropriate γ parameter can be more challenging than expected, especially when RL algorithms are combined with function approximators. For example, Wiering and Van Hasselt [277] show that different algorithms prefer different discount factors, while François-Lavet, Fonteneau, and Ernst [58] show the benefits of initially starting with a low discount factor which gradually gets increased while training progresses. Finally, Van Seijen, Fatemi, and Tavakoli [262] introduce a method that allows the use of low discount factors for approximate RL algorithms while at the same time highlighting that the common perception of the role of γ might need revision from the RL community. We discuss an alternative perspective to solving RL problems that does not involve the role of γ in Appendix C.

3.4 VALUE FUNCTIONS

We are now ready to introduce the arguably most important concept underlying many RL algorithms: the concept of value. We can define

the value of a state s, as well as the value of a specific policy π or of a particular action a, anyhow, independently from what we are considering, the notion of value is always directly linked to the concept of expected discounted return defined in Eq. 3.6. Given an MDP and a policy π, we can determine the value of a state s as a function $V^\pi : \mathcal{S} \to \mathbb{R}$ that measures the expected return that the agent will receive when starting in s and following π thereafter.

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, \pi\right]. \tag{3.7}$$

$V^\pi(s)$ is also known as the state-value function and intuitively tells us how good or how bad it is for an agent to be in a certain state. While this function is only conditioned on the state that is being visited by the agent, we can also condition it on the actions that the agent takes. By doing so we will quantify how good or bad it is for the agent to take a certain action a in a certain state. This function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ comes with the name of state-action value function and is defined as follows:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t = s, a_t = a, \pi\right]. \tag{3.8}$$

Both value functions are very powerful as they allow to characterize an agent's behavior by quantitatively assessing its interaction with the environment. They can be seen as the agent's knowledge and represent its desirability of being in a specific state. As we will see in the coming sections, accurately modeling these value functions is one of RL's major goals.

A key property of $V^\pi(s)$ and $Q^\pi(s,a)$ is that both value functions satisfy a consistency condition that allows us to define both functions recursively. For example let us consider the state-value function $V^\pi(s)$ presented in Eq. 3.7, we can rewrite it as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \big| s_t = s, \pi\big] \\ &= \mathbb{E}\big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \big| s_t = s, \pi\big] \\ &= \mathbb{E}\big[r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \dots) \big| s_t = s, \pi\big] \\ &= \mathbb{E}\big[r_t + \gamma V^\pi(s_{t+1}) \big| s_t = s, \pi\big] \\ &= \sum_a \pi(a|s) \sum_{s_{t+1}} p(s_{t+1}|s,a)\big[\Re(s_t, a, s_{t+1}) + \gamma V^\pi(s_{t+1})\big]. \end{aligned} \tag{3.9}$$

Similar steps can be followed when considering $Q^\pi(s,a)$ which can then be recursively defined as:

$$Q^\pi(s,a) = \sum_{s_{t+1}} p(s_{t+1}|s,a)\big(\Re(s_t,a,s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) Q^\pi(s_{t+1}, a_{t+1})\big). \tag{3.10}$$

When it comes to sequential decision making, we are interested in maximizing each state value or each state-action pair value, since by doing so, we will be finding a policy π that is optimal. The optimal policy π^* is a policy that realizes the optimal expected return defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \text{ for all } s \in \mathcal{S} \tag{3.11}$$

and the optimal Q value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}. \tag{3.12}$$

When we recursively define both optimal value functions as we did for Eq. 3.9 we obtain:

$$V^*(s_t) = \max_{a} \sum_{s_{t+1}} p(s_{t+1}|s_t,a) \Big[\Re(s_t,a,s_{t+1}) + \gamma V^*(s_{t+1}) \Big] \tag{3.13}$$

for the optimal state-value function, and

$$Q^*(s_t,a_t) = \sum_{s_{t+1}} p(s_{t+1}|s_t,a_t) \Big[\Re(s_t,a_t,s_{t+1}) + \gamma \max_{a} Q^*(s_{t+1},a) \Big], \tag{3.14}$$

for the optimal state-action value function. Equations 3.13 and 3.14 are well known to correspond to the Bellman optimality equations [18].

If the optimal Q function is learned, it becomes a straightforward task to derive an optimal policy since one only needs to select the action which has the highest value in each state as defined by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \text{ for all } s \in \mathcal{S}. \tag{3.15}$$

It is also worth noting that the Q function and the V function satisfy the following equality

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a) \text{ for all } s \in \mathcal{S}. \tag{3.16}$$

As we will later see throughout this thesis this equality is particularly important for the development of many RL algorithms.

3.5 LEARNING VALUE FUNCTIONS

The V function and the Q function play a crucial role in the development of optimal decision making algorithms, and over the years, several methods have been introduced to learn them. While all these algorithms' ultimate goal is to yield an optimal policy, there exist cases for which learning these value functions is easier than others.

The complexity of learning a value function depends on how many MDP components are known to the agent. If the agent has access to all five of the components of the MDP that we introduced in Sec. 3.2 and the state and action spaces are finite, then these algorithms are part of a collection of methods that comes with the name of Dynamic Programming (DP). DP algorithms such as value-iteration, policy-iteration and variants [19, 272] learn an optimal value function, or optimal policy, by exploiting the fact that the transition function $\mathcal{P}$, and the reward function $\Re$ of the MDP are known. While DP methods can be considered as the progenitors of many RL algorithms, we will not discuss them here since throughout this thesis, we will be interested in scenarios for which $\mathcal{P}$ and $\Re$ are unknown. Specifically, we will introduce novel methods that aim to learn an optimal value function without requiring to learn an approximation of the transition and reward functions ($\widehat{\mathcal{P}}$ and $\widehat{\Re}$) neither, therefore placing all contributions of this dissertation within the model-free RL literature.

3.5.1 *Monte Carlo Methods*

The first family of methods that can learn optimal value functions when no complete knowledge of the environment is available comes with the name of Monte Carlo (MC) methods. MC algorithms only require RL trajectories to discover an optimal policy and achieve this by sampling and averaging the rewards obtained while the agent is interacting with the environment. While MC methods can be used both for learning $V^*(s)$ as well as for learning $Q^*(s,a)$, in this section, we only present how one can learn the state-value function. MC algorithms' key idea relies on computing the actual sum of discounted rewards that an agent obtains once an episode finishes. This corresponds to computing the quantity defined in Eq. 3.6. Once this value is computed, it can be used for updating the current value of each state with the following update rule:

$$V(s_t) := V(s_t) + \alpha\big[G_t - V(s_t)\big] \tag{3.17}$$

where $\alpha \in [0,1]$ is the learning rate controlling how much we want to change the value estimate of a state based on G_t. As a practical example let us consider the MDP represented in Fig. 3.2. Let us assume that the starting state of the environment is s_0 while the terminal state (the state that interrupts the Reinforcement Learning loop) of the environment is s_2, and that the agent follows a policy π that results into the following state visits: s_0, s_1, s_0, s_2. The rewards associated to each visited state are therefore $-1, +2$ and $+3$ respectively. If we set the discount factor to 0.99 we know that the real discounted return that is obtained at the end of the agent-environment interaction when starting in state s_0 is $\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} = -1 + \gamma 2 + \gamma^2 3 \approx 3.92$. If we now assume that the value of s_0 has never been updated before and that is

therefore 0, and that we set $\alpha = 0.5$, the result of one MC update for s_0 will be ≈ 1.96.

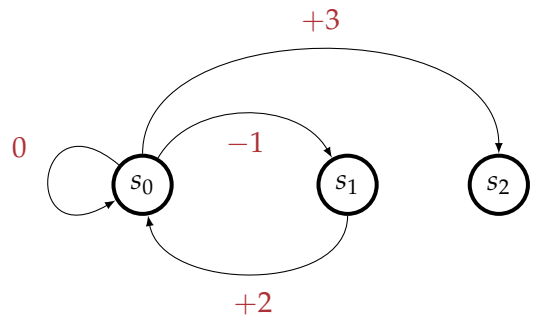


Figure 3.2: A visual representation of a simple MDP. For each state transition the corresponding reward is presented in red.

When dealing with MC learning, it can be possible that a certain state is visited more than once before a terminal state is reached. In the MDP represented in Fig. 3.2 this is the case for s_0. If that happens, one must decide when to update $V(s)$ and which value to use as G_t, as different state visits result in different G_t values. There are two typical ways to deal with this: either update $V(s)$ only once, or one can update $V(s)$ each time the state is visited by simply using as G_t the average of all the different discounted returns. While several successful applications of MC methods exist [104, 126, 140], a well known issue from this family of algorithms is that they suffer from highly biased updates. In fact, one needs to compute the sum presented in Eq. 3.5 over all visited states, resulting in returns with considerable variance. It is easy to see how this can become an issue, especially when the length of the episodes increases. In fact, the larger the episode's length, the more significant the variance of the updates. Furthermore, an additional drawback of MC methods is that one must wait until the agent visits a terminal state before being able to perform an update that is based on Eq. 3.17, a drawback that is addressed by the methods that are presented hereafter.

3.5.2 *Temporal Difference Learning*

Temporal Difference (TD) Learning [237, 241] is a learning paradigm that allows overcoming the issues mentioned above that characterize MC learning based methods. The key idea of TD-Learning is to update the value of each state with respect to a single MC update, therefore overcoming the hurdle of having to wait for the end of an episode before being able to update the value of a state. Just as MC methods TD-Learning algorithms also learn an optimal value function based on the experience that the agent collects. However, these algorithms base their updates only on the value of a single, consecu-

tive state rather than on the real discounted return that is dependent on the entire sequence of visited states. Updating the value of a state with respect to the value of its successor state only is a technique that comes with the name of bootstrapping, and is a very effective design choice that reduces the variance in the updates. Bootstrapping can be used to learn the V function and the Q function and is at the core of the most popular model-free RL algorithms. The first and simplest form of TD-Learning was introduced by Sutton [237] for learning the state-value function with an algorithm that updates the value of a state based on the following learning rule:

$$V(s_t) := V(s_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]. \tag{3.18}$$

We can now clearly see that differently from what happens in the MC update presented in Eq. 3.17 the update of a state now only depends on the reward and the value of the next state. This quantity is denoted as the TD-error δ_t and is defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{3.19}$$

where $r_t + \gamma V(s_{t+1})$ is also known as the TD-target. If we again consider the simple MDP represented in Fig. 3.2 and assume that the value of each state of the process is set to 0 while the discount factor γ is this time set to 0.5 and α is again 0.5, a TD update for $V(s_0)$ based on a policy resulting in state s_2 will result into the new value estimate of 1.5. TD-Learning is a very effective strategy for building algorithms that can learn in an online, fully incremental fashion as one only needs to wait for a single time-step before updating the considered value function. Due to its striking simplicity, TD-Learning has been widely adopted by RL practitioners developing algorithms for learning the Q function. We will present some of the most important algorithms hereafter.

Q-Learning Introduced by Watkins and Dayan [271] is arguably the most popular model-free RL algorithm. It works by keeping track of an estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \Re$ and updates each visited state-action pair with the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t)\big]. \tag{3.20}$$

The key component of Q-Learning's update rule is the max operator, which characterizes its TD-error and that is necessary for constructing the TD-target. Since there are as many Q values as there are actions available to the agent, one must choose which Q value to use as a reference when updating the value of the state-action pair that the agent is currently visiting. The max operator simply chooses the state-action pair with the largest Q value, a simple design choice

that has the appealing property of making Q-Learning converge to $Q^*(s,a)$ with probability 1 as long as all state-action pairs are visited infinitely often. Interestingly, this guarantee holds even if the agent follows a random policy. The max operator also defines Q-Learning as an off-policy learning algorithm, since the Q values chosen for the construction of the TD-target might not correspond to the ones that are associated with the state that the agent will visit after having updated its Q function.

SARSA Also known as "online Q-Learning" [199] can be seen as the most straightforward extension of the TD-Learning method presented in Eq. 3.18, and similarly to Q-Learning is an algorithm that aims at learning the state-action value function Q. The key idea of SARSA is to update a state-action value with respect to the Q value that is associated to the state that the agent will visit after a certain action is performed. Therefore, SARSA does not use the max operator within its TD-error and constructs TD-targets that represent the policy that the agent is following, a characteristic that defines SARSA as an on-policy RL algorithm. The way SARSA learns the Q function is given by the following update rule

$$Q(s_t,a_t) := Q(s_t,a_t) + \alpha\big[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\big], \quad (3.21)$$

where we can clearly see how the algorithm uses all the elements of the quintuple of events $(s_t,a_t,r_t,s_{t+1},a_{t+1})$ a property that gives rise to the name *sarsa*. Not using the max operator in Eq. 3.21 results in an algorithm that, differently from Q-Learning, does not directly learn the optimal Q function anymore, but rather learns to estimate $Q^\pi(s,a)$. This has the drawback of not guaranteeing convergence to $Q^*(s,a)$ for any random policy anymore. To overcome this, SARSA needs an exploration policy that is greedy in the limit of infinite exploration [230]. This can be achieved with the popular ϵ-greedy selection policy which defines the action that the agent takes as:

$$a_t = \begin{cases} \arg\max_{a\in\mathcal{A}} Q(s_t,a) & \text{with probability } 1-\epsilon \\ a \sim \mathcal{U}(\mathcal{A}) & \text{with probability } \epsilon \end{cases} \quad (3.22)$$

where ϵ is a hyperparameter that changes while training progresses. During early training iterations, its value is close to 1, while it approaches 0 by the end of training. This allows the agent to take actions that are representative of a large set of policies when the learned Q function does not yet correspond to $Q^*(s,a)$, while it will favor greedy actions at the end of training. This is a simple, yet effective strategy that deals with the exploration-exploitation dilemma. It is however worth noting that its use is not limited to on-policy RL algorithms only. Furthermore, the method presented in Eq. 3.22 represents only

one possible way of balancing exploration and exploitation, and although it is arguably the most popular of such methods, it is not the only existing one. We refer the reader to chapter 5 of [274] for a thorough analysis of different exploration algorithms.

Double Q-Learning In some environments, Q-Learning is known to perform poorly. This poor performance stems from the fact that the algorithm largely overestimates some state-action values due to the max operator in its TD-error [251]. The max operator serves for constructing an approximation of the maximum expected action-value of a state, which, as discussed by Van Hasselt [259], is a technique that results in positively biased estimates [231, 234]. In some RL problems, this can significantly influence the learning process, which has led the RL community to develop a set of solutions that try to mitigate this bias [131, 132, 181, 295]. Among the different solutions, Double Q-Learning [259] is probably the most popular one. Its main idea is to keep track of two different state-action value functions, Q_1 and Q_2, which get alternatively used for selecting which action to perform. When one of the two Q functions determines the action that maximizes the state-action value of the next state, the remaining value function is used for evaluating this estimate. This can be achieved with the following rule:

$$Q_1(s_t, a_t) := Q_1(s_t, a_t) + \alpha \left[r_t + \gamma \, Q_2(s_{t+1}, a^*) - Q_1(s_t, a_t) \right], \quad (3.23)$$

where $a^* = \arg\max_{a \in \mathcal{A}} Q_1(s_{t+1}, a)$. Note that at each time step, only one of the two Q functions gets updated. While training progresses, the choice of which Q function to update is determined randomly. In the case it is Q_2, the update rule is identical to the one presented in Eq. 3.23 with the only difference being that the role of the two Q functions is swapped. Double Q-Learning converges to the optimal state-action value function with probability 1 under the same conditions as Q-Learning. Van Hasselt [259] shows that using two separate Q functions significantly mitigates the overestimation bias. Yet, this comes at the price of an algorithm that is twice more expensive in terms of memory requirements. It is also worth noting that although Double Q-Learning does not overestimate the state-action values, it might instead underestimate them, which in some environments can still yield poor performance.

QV(λ)-Learning First introduced by Wiering [276] and further developed by Wiering and Van Hasselt [277] is an on-policy RL algorithm which differently from the previously introduced methods keeps track of an estimate of the state-value function $V : \mathcal{S} \to \mathbb{R}$ alongside the usual estimate of the state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Since the goal is to jointly learn two value functions, QV(λ)-Learning requires two separate update rules. The V function

is learned via the same form of TD-Learning that we introduced in Eq. 3.18, with the only difference being the addition of the eligibility traces $e_t(s)$ at the end of the update rule (an RL technique that we will review in Chapter 8). QV(λ)-Learning, therefore, learns the V function with the following update rule:

$$V(s) := V(s) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]e_t(s). \tag{3.24}$$

Since as discussed earlier only learning the V function is not sufficient for deriving an optimal policy one needs to learn the Q function as well. In QV(λ)-Learning this is done as follows:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)\big]. \tag{3.25}$$

An attractive property of the algorithm is that it uses the same TD-target ($r_t + \gamma V(s_{t+1})$) for defining the two different TD-errors that are required for learning the state-value and the state-action value functions. Among the main insights that motivate learning two value functions over one, Wiering [276] mentions the possibility that the V function, since it does not depend on the agent's actions, might converge faster than the Q function. As described earlier, the V function only depends on the state space of the MDP, which by definition is smaller than the state-action space. For a more in-depth and formal presentation of the conditions that show the benefits of jointly learning the V function alongside the Q function, we refer the reader to chapter 5 of [85].

3.6 FUNCTION APPROXIMATORS

If it is true that model-free RL algorithms are very powerful methods for learning an optimal policy when parts of the MDP are unknown, it is also true that all the algorithms mentioned above suffer from the curse of dimensionality. Model-free algorithms are typically implemented in a tabular fashion, meaning that the state values, or state-action values, are stored within tables of sizes $|\mathcal{S}|$ and $|\mathcal{S} \times \mathcal{A}|$ respectively. Albeit straightforward and easy to implement, such an approach presents severe limitations. The first major drawback of the tabular representation approach is that it does not scale well with respect to the MDP complexity. If the environment state and action spaces become very large, storing a table quickly becomes unfeasible in terms of storage space. Furthermore, tabular representations are also unable to deal with continuous states. A natural solution to this problem could consist in discretizing the state space; however, this approach still results in the aforementioned storage space issues when done thoroughly. Therefore, if one wants to use RL techniques, even when the state space of the MDP is large, a better solution is needed. This solution is based on parametrized function approximation. In this context, the goal is not to learn the exact value function anymore

but to rather replace its tabular representation with a parametrized function. This function parameters can then be adjusted based on the RL algorithms that we introduced in Sec. 3.5.2.

3.6.1 *Linear Functions*

The most straightforward type of function approximator one can use is a linear function. Given a state-action tuple that gets represented as a feature vector $\mathbf{x}(s) = [x_1(s), x_2(s), ..., x_q(s)] \in \mathbb{R}^q$, and a function parametrized by a vector of parameters $\boldsymbol{\theta}^a \in \mathbb{R}^q$ for each action $a \in \mathcal{A}$, as shown in [275], we can redefine the value of a state-action pair as:

$$Q(s,a) = \sum_i \theta_i^a x_i(s). \tag{3.26}$$

Given a trajectory $\langle s_t, a_t, r_t, s_{t+1} \rangle$, the Q-Learning algorithm presented in Eq. 3.20 can now be used for updating the parameters θ_i^a for all i with the following update rule:

$$\theta_i^{a_t} := \theta_i^{a_t} + \alpha(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t))x_i(s_t). \tag{3.27}$$

We can observe that this update rule modifies the parameter vectors $\boldsymbol{\theta}^a$ by minimizing the mean squared error loss between a given state-action tuple and Q-Learning's TD-target since

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{2}\big(y_t - Q(s_t, a_t)\big)^2 \text{ with } y_t = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) \\ \frac{\partial \mathcal{L}}{\partial \theta_{i,a_t}} &= -\big(y_t - Q(s_t, a_t)\big)x_i(s_t) \\ \theta_i^{a_t} &:= \theta_i^{a_t} + \alpha(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_t) - Q(s_t, a_t))x_i(s_t). \end{aligned} \tag{3.28}$$

Similar steps can be used for adapting all of the RL algorithms that we introduced in the previous section. As a representative example for the on-policy learning case let us consider the SARSA algorithm. One can learn an approximation of the Q function by updating the parameters of a linear function as follows:

$$\theta_i^{a_t} := \theta_i^{a_t} + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))x_i(s_t). \tag{3.29}$$

Linear functions can yield successful results [122, 150, 175] as they can indeed deal with the aforementioned curse of dimensionality problem. However, non-linear functions are usually preferred since their representational power is even larger than the one of linear methods. Throughout this dissertation, we are interested in non-linear functions that come in the form of deep neural networks. As we have seen in the previous chapter, neural networks such as e.g convolutional networks are able to learn very rich representations from their

inputs. However, this also makes these kinds of models particularly challenging to train in an RL context. We will now describe how one can successfully deal with some of the challenges that characterize the use of deep neural networks in RL by presenting some of the most important algorithms that have been introduced over the years.

3.7 DEEP REINFORCEMENT LEARNING

Before looking into how RL algorithms should be integrated within deep neural networks, it is important to mention that RL techniques have been successfully combined with (less powerful) neural networks for over three decades. In fact, the field known as Connectionist Reinforcement Learning (CRL) resulted in the very first algorithms that managed to outperform human experts on specific tasks. Among the multiple possible examples of this family of techniques, we mention the TD-Gammon program introduced by Tesauro [250]. TD-Gammon successfully learns an approximation of the popular Backgammon boardgame's evaluation function through the same TD-Learning methods that we presented in Sec. 3.5.2. Tesauro's program achieved a level of play comparable to the one of the top human Backgammon players of its time and is even nowadays considered one of the most important RL breakthroughs. For a more detailed presentation about the successful applications of CRL algorithms, we refer the reader to [32].

While certainly successful for a certain set of problems (see for example chapter 1 of [202]), CRL techniques also present severe limitations. Since they only use multi-layer perceptrons as function approximators, these algorithms cannot be used for tackling problems where the state representation of the MDP is highly dimensional. To overcome this, more complicated and powerful networks are required. Deep Reinforcement Learning (DRL) [10, 59, 134] is a research field that combines RL algorithms with deeper and more complex neural architectures. In value based model-free DRL we are interested in learning an approximation of an optimal value function with a deep neural network that comes with parameters θ

$$V(s;\theta) \approx V^*(s) \qquad (3.30) \qquad Q(s,a;\theta) \approx Q^*(s$$

and that usually comes in the form of a convolutional neural network. We now describe some of the algorithms which have contributed to the development of DRL the most.

Deep Q-Learning (DQN) Just like Q-Learning is arguably the most important tabular model-free RL algorithm, so is DQN when it comes to DRL. First introduced by Mnih et al. [160] and then made

popular by the work presented in [161] this algorithm can certainly be considered as the very first successful example of a neural network that is able to learn an approximation of the optimal state-action value function just from high sensory inputs (in this case images). As the name suggests, Deep Q-Learning (DQN)[1] is based upon the Q-Learning algorithm and aims at learning an approximation of the optimal state-action value function Q. This is done by reshaping Q-Learning's update rule, presented in Eq. 3.20, into a differentiable loss function that can be used for training a convolutional network. This is achieved through the following objective function:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \Big[\big(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \big)^2 \Big]. \quad (3.32)$$

We can start by observing that the general principles that characterize the algorithm are the same ones that made it possible to generalize Q-Learning to the use of linear function approximators, however, differently from when a linear function is used, the mapping between input and feature spaces is now naturally not preserved anymore. Similarly to what we presented in Sec. 3.6.1, we can see from Eq. 3.32 that learning $Q(s, a, \theta)$ is again achieved by minimizing the squared error loss between the $Q(s_t, a_t; \theta)$ estimates and the off-policy TD-target

$$y_t^{DQN} = r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-). \quad (3.33)$$

Despite this similarity, DQN requires some additional algorithmic design choices, without which it would turn out to be almost impossible to successfully train a neural network with Eq. 3.32. These additions, which significantly make DQN differ from the algorithm presented in Eq. 3.27 are the following:

- Experience Replay: a memory buffer, D, represented as a queue which stores RL trajectories of the form $\langle s_t, a_t, r_t, s_{t+1}\rangle$. Once this memory buffer is filled with a large set of these quadruples, DQN uniformly samples batches of trajectories for training its network. This makes it possible to exploit past trajectories multiple times by reusing them while training, which makes the overall algorithm more sample efficient. Furthermore, using a memory buffer also improves the stability of the training procedure. Recall that each trajectory is representative of a certain episode. By repeatedly randomly sampling a different τ from the memory buffer, a resulting mini-batch of trajectories will

1 In DQN the 'N" in the acronym stays for 'Network" and replaces what could have been the, arguably more intuitive, 'L" of Learning.

be representative of different episodes and of different policies. As a consequence, the correlation between trajectories within a mini-batch will be small. Although made popular by the DQN algorithm, using an experience replay buffer for tackling sequential decision making problems was already presented by Lin [137].

- Target Network: We can observe from Eq. 3.33 that the TD-target used by DQN for bootstrapping is not computed by the Q network that is being optimized (θ), but rather from a second separate network that is parametrized with θ^-. This second network has the same structure as the main Q network, but its weights do not change each time RL experiences are sampled from D. On the contrary, its weights are temporally frozen and only periodically get updated with the parameters of the main network θ as defined by an appropriate hyperparameter. Note that this is a design choice that is not motivated by the TD-Learning paradigm that we presented in Sec. 3.5.2, where we have seen that TD-Learning based methods learn in a fully online fashion by updating their value estimates based on their own future estimates. With a target-network, although the θ network still learns via the methods of temporal differences, it now requires an auxiliary, external model if it wants to successfully learn $\approx Q^*(s, a; \theta)$. Several works have studied the target network's role to understand why this design choice appears to be necessary for DRL. Yet, the DRL community does not fully understand the role of θ^-. For more about this topic, we refer the reader to [112, 185].

With all these concepts in place we can show that given a training iteration i, differentiating this objective function with respect to θ gives the following gradient:

$$\nabla_{\theta_i}\mathcal{L}(\theta_i) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\Big[(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-_{i-1}) - Q(s_t, a_t; \theta_i))\nabla_{\theta_i} Q(s_t, a_t; \theta_i)\Big]. \quad (3.34)$$

The DQN algorithm showcased its entire potential in [161] where Mnih and colleagues developed a convolutional neural network that trained with Eq. 3.32 learned how to successfully play most of the `Atari` games that are part of the popular Atari Arcade Learning Environment (ALE) [17], a well-known platform that even nowadays serves as a benchmark for testing the performance of DRL algorithms. In the ALE, a DRL algorithm has to learn how to play 57 different emulations of `Atari` games which are specifically designed within a simulator (see Fig. 3.3 for a visualization of some of the games that are part of the ALE suite). Remarkably, DQN not only learned how

to play most of the games of the platform but also achieved a final performance that, on most games, was superior to the one of human expert players. What is even more remarkable is that this was achieved by providing as inputs to the network the images representing the game only, therefore making the model learn just from its own experience in a pure model-free RL fashion. Since then, DQN has been successfully used for a large variety of applications ranging from healthcare [190, 255], robotics [109] and natural language processing [86, 167] to even particle physics [141, 211]. However, despite all these remarkable applications, the algorithm still comes with some drawbacks, some of which are addressed by the algorithms presented below.

Figure 3.3: A visual representation of some of the `Atari` games that are part of the Arcade Learning Environment (ALE) [17]. From left to right `Breakout`, `Seaquest`, `Qbert`, `Pong`, `MsPacman`, `KungFu-Master`, `Enduro` and `Space Invaders`. Most of these games will be of interest in Chapter 8 and Chapter 9.

Double Deep Q-Learning (DDQN) Van Hasselt, Guez, and Silver [261] showed that the DQN algorithm suffers from the same issue that also characterizes the Q-Learning algorithm: the overestimation bias of the Q function. They show that DQN is prone to learn overestimated Q-values because the same values are used both for selecting an action ($\max_{a \in \mathcal{A}}$) as well as for evaluating it ($Q(s_{t+1}, a; \theta^-)$). This becomes clearer when re-writing DQN's TD-target presented in Eq. 3.33 as:

$$y_t^{DQN} = r_t + \gamma \, Q(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta); \theta^-). \tag{3.35}$$

As a result, DQN tends to approximate the expected maximum value of a state, instead of its maximum expected value. As presented in Sec. 3.5.2, in the tabular case this can be solved by keeping track of two separate Q functions, and by randomly preferring one Q function over the other when it comes to selecting which action to execute.

DDQN generalizes this idea and untangles the action selection process from its evaluation by taking advantage of the previously introduced target network θ^-. DDQN's target stays the same as in DQN with the main difference being that the selection of an action, given by the online Q-network θ, and the evaluation of the resulting policy, given by θ^-, can now result into smaller overestimations simply by symmetrically updating the two sets of weights (θ and θ^-) which can easily be achieved by regularly switching their roles during training. While not always significantly impacting the performance of DQN (one can still act optimally even if some actions are associated to unrealistically high $Q(s,a)$ estimates), there are also cases for which the overestimation bias of the Q function significantly slows down the training process, and even prevents the DQN algorithm from improving its policy over time at all. We will come back to this issue in Chapter 8.

Prioritized Experience Replay (PER) We have seen that next to the target network θ^- an equally important role within the DQN algorithm is played by the experience replay memory buffer D. Schaul et al. [216] showed that the efficiency of how Deep Q-Networks use this buffer could be improved. Their claim stems from the fact that, as shown in Eq. 3.32, the RL trajectories that get sampled from the memory buffer when constructing a mini-batch of trajectories are sampled uniformly ($\sim U(D)$). This approach has the main drawback that it considers each τ stored in the buffer as equally important and representative for training. However, it is easy to imagine learning situations where some trajectories are more valuable than others. For example, at the beginning of training, most of the trajectories contained within D will be representative of early agent-environment interactions. It is, therefore, safe to assume that the network will learn the $Q(s,a)$ estimates representative of these early dynamics much faster than it will learn the Q values that are associated with trajectories occurring more rarely. The idea of PER is to use only highly informative trajectories when it comes to building the mini-batches that are used for training the network. The importance of different trajectories is given by their respective TD-error. PER ensures that the probability of sampling trajectories is proportional to their respective TD-errors: the higher the TD-error, the larger the probability for a specific τ to be sampled. In practice, given a trajectory τ, the probability of sampling it is given by the following equation:

$$P(\tau) = \frac{p_\tau^\alpha}{\sum_k p_k^\alpha} \tag{3.36}$$

where p_τ is $|\delta_\tau + \epsilon|$ with ϵ being a small positive number ensuring that the probability of sampling a trajectory remains positive even in the edge case where the TD-error is 0. Although simple and intuitive, implementing a PER buffer is not that straightforward and still

presents some algorithmic caveats that need to be taken into account. Yet, if done correctly it dramatically improves the sample efficiency of Deep Q-Networks [167].

Dueling Networks While the DDQN algorithm directly tackles a fundamental algorithmic bias that characterizes the way DQN learns the Q function, and PER addresses the inefficiency of its memory buffer, the contribution presented by Wang et al. [270] is of slightly different nature. Their work consists of a novel type of neural architecture called the Dueling Network. This is a contribution that resembles more the kind of progress that is made by the supervised learning community, which, as we discussed in the previous chapter, has put a lot of effort into developing novel neural architectures for tackling computer vision tasks. Nevertheless, Wang's work is a perfect example that showcases how in DRL, carefully designing the function approximator is just as important as properly defining its objective function. A Dueling network is a network that, after performing a series of convolutions, instead of directly outputting the state-action values for a specific state as DQN and DDQN do, adds some intermediate computations. The idea is to estimate the value of a state and the advantages for each action before outputting the final Q values. The state values are computed based on Eq. 3.7, while the advantage function A is simply the difference between the Q function and the V function:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{3.37}$$

To successfully estimate state values, advantages, and state-action values, the network requires a specific architecture consisting of three separate streams. Each stream is responsible for estimating one of the three value functions and is initialized with its own parameters $\theta^{(\cdot)}$. The final Q function of the model is then obtained by combining what is learned by each stream as follows:

$$\begin{aligned} Q(s,a;\theta^{(1)},\theta^{(2)},\theta^{(3)}) = V\big(s;\theta^{(1)},\theta^{(3)}\big) + \\ \big(A(s,a;\theta^{(1)},\theta^{(2)}) - \max_{a_{t+1}\in\mathcal{A}} A(s,a_{t+1};\theta^{(1)},\theta^{(2)})\big). \end{aligned} \tag{3.38}$$

Building a network with different task-specific streams is a design choice that resembles the way models are built when tackling multi-task classification tasks. However, note that the output of each stream that either estimates the state values or the advantage function gets aggregated in a final layer that estimates the state-action values. Training a Dueling Network is done by minimizing the same objective function that is also minimized by DQN and DDQN. The idea of taking into account the V function when learning the Q function is a concept which will come back, in a different flavor, in Chapter 8 and Chapter 9.

Policy Gradient Methods So far we have only considered algorithms that are part of the action-value family of methods, which, as explained in Sec. 3.5 are techniques that derive an optimal policy from a learned Q function. Yet, there is a collection of algorithms that is able to learn a policy directly, and that therefore bypasses the requirement of having to consult state-action values when it comes to action selection. These methods, which have contributed to the development of DRL just as much as action-value methods, come with the name of Policy Gradients. They directly parametrize a policy at each time-step as $\pi(a|s;\theta) = \Pr\{a_t = a|s_t = s;\theta_t = \theta\}$ and seek to optimize the parameters θ such that the performance of the policy is maximized. Note that this is drastically different from all the methods which we have seen so far, where the aim, in fact, was to minimize the TD-error through gradient descent optimization. Since policy gradients aim to maximize their performance, they learn via gradient ascent and therefore update their parameters as follows

$$\theta \leftarrow \theta + \alpha \nabla \xi(\theta). \tag{3.39}$$

Here α is again the learning rate, and ξ is a measure that quantifies the performance of the policy. Similarly to action-value based methods, one can parametrize a policy either with a linear function or with a deep neural network. When the action space is discrete, it is common practice to parametrize π through the same exponential softmax distribution, which we have seen in the previous chapter when presenting neural networks trained for classification tasks. Therefore we have

$$\pi(a|s;\theta) = \frac{e^{h(s,a;\theta)}}{\sum_b e^{h(s,b;\theta)}} \tag{3.40}$$

where $h(s,a;\theta)$ is any function approximator. By doing so, policy gradient methods naturally deal with the exploration-exploitation trade-off since they simply assign different probabilities to different actions. This also allows these methods to approach a deterministic policy (which cannot be achieved when using ϵ-greedy action selection) and makes these algorithms arguably easier to train since learning a policy could be easier than learning state-action returns. The fundamental result which allows optimizing any differentiable policy is the policy gradient theorem [240]. Sutton et al. [240] show that the gradient of π does not depend on the gradient of the state distribution and that it can therefore be expressed as follows:

$$\nabla \xi(\theta) = \sum_s \mu(s) \sum_a Q^\pi(s,a) \nabla \pi(a|s;\theta) \tag{3.41}$$

where $\mu(s)$ is the stationary on-policy distribution under π. By expressing the gradient as such, it is now possible to optimize π even when the state distribution of the environment is unknown (as discussed in Sec. 3.5). Policy gradient methods can be used for learning

a policy through the same kind of techniques which in Sec. 3.5 were used for learning value functions. Within the Monte Carlo setting the arguably most important of such algorithms is REINFORCE [279], while Actor-Critic algorithms [65, 78, 135, 158, 219–221, 269] learn a policy with the additional help of a bootstrapped value function (typically V or A).

Rainbow From the examples above, it is clear that much progress has been achieved by the DRL community in creating algorithms capable of learning faster and better. Explaining all the individual value-based contributions that have made DRL the popular research field it is nowadays [91], is beyond the scope of this chapter. Yet, if there is one algorithm that encapsulates most of the progress that the DRL community has achieved over the last years, that is Rainbow [95]. Rainbow is a single, almighty agent that integrates most of the important breakthroughs that DRL researchers have introduced over the last decade within the same algorithm, ranging from the previously mentioned DDQN algorithm and PER system to more recent techniques such as distributional DRL [16], multi-step learning, distributed training [158] and noisy networks [57] that allow for better exploration.

3.8 THE DEADLY TRIAD OF DEEP REINFORCEMENT LEARNING

We now end this chapter by presenting one of the main limitations that currently characterizes the field of DRL and that has inspired part of the research that is presented in this dissertation.

The combination of RL with function approximation can result in unstable training and algorithms prone to diverge while learning. As a representative example of what it means for an RL algorithm to diverge, let us consider the following MDP firstly introduced by Tsitsiklis and Van Roy [256]. The MDP consists of three states s_0, s_1 and the terminal state s_2. Each state is described by a single scalar feature ψ such that $\psi(s_0) = 1$, $\psi(s_1) = 2$, and $\psi(s_2) = 3$. The estimated state-value of each state is therefore given by $V(s) = \psi(s) \cdot w$, where w is the single weight we would like to update. The MDP is represented in Fig 3.4. Each state transition is associated with a reward

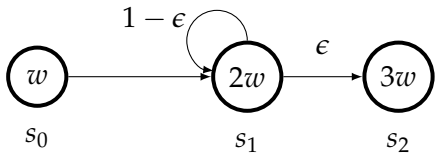


Figure 3.4: A visual representation of the MDP proposed by Tsitsiklis and Van Roy [256] that shows how RL algorithms combined with function approximators can diverge.

of 0, which means that the optimal weight value for having perfect value predictions is $w^* = 0$. Let us now assume that we are updating the state-value function based on an on-policy learning scheme as discussed in Sec. 3.5.2. We then know that each time we are updating the state-value function for s_0, the value of s_1 will, in expectation, also be updated multiple times. This however, changes if we are following an off-policy learning scheme since each time we update the state-value function for s_0, we do not necessarily update the value of s_1 anymore. As shown by Van Hasselt et al. [260] if we would now update w based on Eq. 3.18 we would have to modify w as $\Delta w \propto r_t + \gamma(V(s_{t+1}) - V(s_t))$. Which results into $0 + \gamma 2w - w = \gamma 2w - w = (2\gamma - 1)w$. If we then set the discount factor $\gamma > 0.5$ as is common practice, we can see that we have $2\gamma > 1$, which will make any weight $w \neq 0$ be updated away from the desired value of 0. Sutton and Barto [239] show that the cause of this type of divergence occurs when RL algorithms are combined with three concepts which we have already encountered in this chapter. These concepts are: bootstrapping (Sec. 3.5.2), off-policy learning (Sec. 3.5.2) and function approximation (Sec. 3.6). This combination is known as the 'Deadly Triad" of DRL, and it is well known that if all of these three elements are combined within the same algorithm, divergence can appear. Divergence results in algorithms that are extremely slow to train, or even in agents that are not able to improve their policy over time at all. Throughout this dissertation we will tackle the problem of the Deadly Triad by first introducing novel algorithms that are less prone to diverge (Chapter 8), while in a second approach by investigating whether the particularly long training times that characterize such algorithms can be reduced by adapting transfer learning strategies (Chapter 9).

4

TRANSFER LEARNING

Outline

We now present Transfer Learning (TL), a machine learning methodology that aims to create algorithms capable of retaining and reusing previously learned knowledge when getting trained on new, unseen problems. Most of the contributions presented within this dissertation are motivated by TL. Therefore, we now introduce the readers to this specific learning paradigm to provide them with all the preliminary knowledge necessary to fully understand the research presented in the coming chapters. We start with a gentle introduction to TL in Sec. 4.1 where we describe the main concepts underlying TL and explain why it is desirable to have machine learning models that are transferable. We then show in Sec. 4.2 some practical, high-level examples that visually represent the benefits that can come from adopting TL strategies in machine learning. We will then provide more rigorous mathematical definitions in Sec. 4.3 where we will characterize TL both for the supervised learning setting as well as for the reinforcement learning one. In Sec. 4.4 we thoroughly review how TL has been studied by the deep learning community in both settings. We end this chapter with Sec. 4.5, where we describe how all the machine learning concepts encountered throughout the first part of this dissertation will play a role in the coming chapters.

4.1 INTRODUCTION

Imagine being an Italian Ph.D. student living abroad who just went for dinner with his French housemate. You had the chance of eating the best, most Italian pasta the two of you have ever eaten in almost ten years spent abroad. The pasta was so good that, one week later, you cannot see yourself as having anything else for dinner. Unfortunately, when your housemate suggests going back to the restaurant, you have to politely decline the invitation, as your favorite Italian restaurant, next to serving delicious food, is also extraordinarily expensive. This prevents you from going there at least until the end of the month, which is when your gentle supervisor will kindly pay you despite you not having published a single paper in over a year. However over all the years you spent abroad, seeking some proper Italian food that was supposedly going to make you feel less homesick, you learned how to cook plenty of pasta yourself. The outcome

of your cooking sessions is certainly not on par with those of your favorite restaurant, but over the years, resulted in some pretty tasty dishes nevertheless. Therefore you decide to cook dinner for yourself and your housemate by replicating the pasta you had as dinner one week ago. While cooking, you try to remember all of the knowledge you acquired over the years while learning how to cook abroad. You remember how to salt the water properly, how long to keep the pasta in the boiling water, and how to season the sauce with the right mix of spices. In the end, the resulting meal, albeit not perfect, will turn out to be a much better dish than the one you will be having one week later, when, this time, it will be your housemate's turn to replicate the pasta for dinner. The reason your pasta ended up being tastier than your housemate's is not that your housemate is French, but rather because he turned out to never having cooked any pasta before. Therefore, he had no previous knowledge to attain from while cooking, while you could instead re-use a set of previously acquired cooking skills. This intuitive but straightforward life episode perfectly summarizes the concept of Transfer Learning, the main topic of interest of this dissertation. However, throughout this work, we will not be focusing on any sort of cooking abilities, nor on any Ph.D. student other than the one authoring this document. We will instead center our attention on situations where it is particularly desirable to have machine learning models, particularly neural networks, that are transferable.

4.1.1 *Transfer Learning in Machine Learning*

There are many reasons that can motivate why it should be worth studying machine learning models from a Transfer Learning (TL) perspective. In this dissertation we have mainly been driven by the following rationale. A first typical scenario where it is helpful to have transferable models is characterized by the lack of training data. This is a problem that mostly affects supervised learning and is typical in situations where collecting and annotating data is an expensive and laborious task (we will see one of such situations in Chapter 7.) As we have seen in Chapter 2, the ultimate goal of supervised learning is that of training a machine learning model that minimizes the empirical risk alongside the expected risk. Unfortunately, when training data is limited it is well known that this goal becomes much harder to achieve, as the resulting learning algorithms become more prone to overfitting and successfully training neural networks can therefore become problematic [4]. TL is also particularly desirable when there are limited computational resources available, which might not allow machine learning practitioners to train new models each time a novel machine learning task is encountered. This can be the case both when tackling supervised learning problems as well as rein-

forcement learning ones, although we believe that among the two learning paradigms, the resources that are required by reinforcement learning are by far more prone to becoming out of reach for many researchers (for a position paper about this topic, we refer the reader to the work of Obando-Ceron and Castro [170]). Lastly, we also believe that studying the degree of transferability of machine learning models has potential additional benefits that go beyond the aforementioned practical advantages. In fact, we argue that TL offers an effective tool for better understanding the machine learning models that are currently being used. By analyzing whether such models can be adapted to novel unseen tasks, it is possible to gain insights about their generalization capabilities, as well as their fundamental inner workings.

4.2 TRANSFER LEARNING IN PRACTICE

Before mathematically defining TL, we continue building some intuition by visually representing how one can observe the benefits of TL in practice. We assume that we would like to solve a certain task and that we can choose between two different models: a model that has never dealt with any task before, and a second model, which is identical to the first one with the major difference being that it has already seen a similar task in the past. Because of their different nature, we refer to the first model as a model trained from scratch and to the second model as a pre-trained model. Ideally, as mentioned in the previous section, we would like the latter model to perform better on the considered task than the first one. Yet, how can we assess if one model is better than the other one? As initially presented by Langley [123], and later generalized by Lazaric [125], we would like the performance of a pre-trained model to result in three possible improvements. If at least one of these improvements is observed while training, we can then consider the pre-trained model to be a better alternative than the scratch model. These improvements are the following:

- Learning Speed Improvements: in this scenario, the performance between a pre-trained model and a model trained from scratch is identical by the time training is finished; however, when this kind of improvement appears, we observe that the pre-trained model converges faster than the model trained from scratch. An example of this TL improvement is presented in the first plot of Fig. 4.1 [1]. The goal is to train a model such that by the end of training, its performance reaches a value of 200. We can clearly see that both models manage to converge to this desired per-

1 This example has been created after pre-training a DQN agent [161] on the `Cartpole-v1` environment of the OpenAI gym [28] and transferring and fine-tuning its parameters to the `Cartpole-v0` environment.

formance value, but that the pre-trained model manages to converge already after ≈ 50 training iterations, whereas the model trained from scratch requires more than 200 training iterations to perform similarly. Also, note that the performance of both models at the beginning of training is identical, as they both start from an initial performance of ≈ 20.

- Jumpstart Improvements: similar to the previous case, also in this scenario, there are no significant differences between the performance of a pre-trained model and the performance of a scratch model by the end of training. However, this changes when we consider the very first training iteration. If jumpstart improvements appear, we can usually observe that when both models start their training process, the performance of the pre-trained model is much closer to the one that will be obtained by the end of training than the one of the scratch model. We visually report an example of this scenario in the second plot of Fig. 4.1 [2]. In this case, the goal is to train a model such that by the end of training its performance will be of ≈ -100. We can clearly see that by the end of training both models are able to achieve this goal successfully, but that at the very first training iteration, the performance of the pre-trained model is significantly closer to the desired final performance (≈ -250) than the one of the scratch model (≈ -450).

- Asymptotic Improvements: when this TL improvement appears, the final performance of a pre-trained model is significantly higher than the one of a model trained from scratch. It is worth noting that similarly to what happens when learning speed improvements are observed, also in this case, the performance of both models is identical when training begins, and that this TL improvement only presents itself after several training iterations. A visualization of this TL improvement can be observed in the last plot of Fig. 4.1 [3] where we can observe that for the first ≈ 20 training iterations, there are no differences in terms of performance between a pre-trained model and a model trained from scratch. However, the more training iterations are performed, the more the pre-trained model starts outperforming the model trained from scratch, reaching a final performance that is almost twice as good by the 100th training iteration.

2 This example has been created after pre-training a DQN agent [161] on the `Acrobot-v1` environment of the OpenAI gym [28] and transferring and fine-tuning its parameters to a modified version of the task where we have manually increased the size of the joints of the pendulum.

3 This is a pure artificial example solely created with the purpose of visualizing how asymptotic improvements can present themselves in practice.

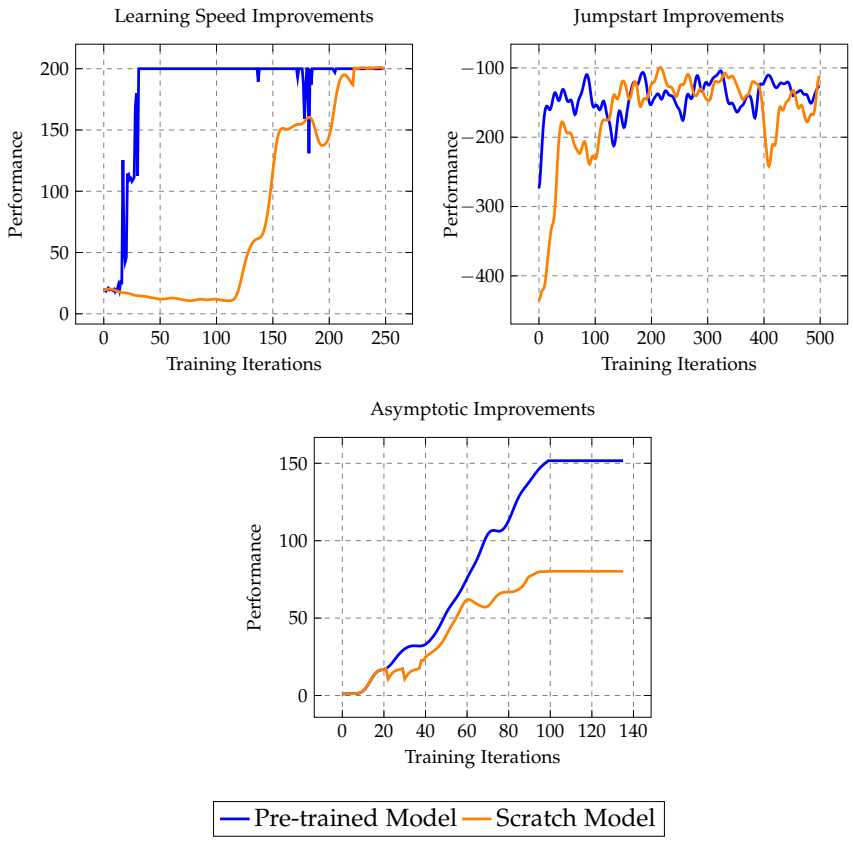


Figure 4.1: A visualization of the three possible desired outcomes that can come from adopting Transfer Learning strategies as initially defined by Langley [123] and later by Lazaric [125].

While the improvements presented in Fig. 4.1 are all highly desirable, it is worth noting that some of them can be preferable to others. In fact, the potential benefits of TL highly depend on the problem at hand. For example, let us consider a training situation where the main goal is to minimize the overall training time of a model. In this particular case, jumpstart and learning speed improvements are more desirable than asymptotic improvements since the latter improvement might not result in a model that converges to a potentially desirable solution faster. On the other hand, if the main objective is that of training a model which performs as best as possible, then evidently, asymptotic improvements are preferred. It is also worth noting that the examples presented in Fig. 4.1 are not mutually exclusive and that, in practice, the benefits of TL can present themselves as a combination of improvements rather than in the form of a single, isolated improvement.

Now that we have introduced the key ideas behind TL and presented why adopting TL training strategies is beneficial, we move to formally characterizing this machine learning paradigm both from a supervised learning perspective as from a reinforcement learning one.

4.3 MATHEMATICAL DEFINITIONS

As is common throughout this thesis, we start by focusing on the supervised learning case.

4.3.1 *Supervised Learning*

The first two definitions that we provide are the ones of domain and task. The first one is defined as follows:

Definition 2 *A domain $\mathcal{D}$ is the combination between an input space $\mathcal{X}$ and a marginal distribution $P(X)$, $\mathcal{D} = \{\mathcal{X}, P(X)\}$.*

Examples of domains can be natural images, time-series data, biomedical markers, etc. In supervised learning, we know that each domain is associated with its respective task, representing the problem we would like to solve. A task is defined as follows:

Definition 3 *A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a conditional output distribution $P(Y|X)$, $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. A decision function $f : \mathcal{X} \rightarrow \mathcal{Y}$ can only be learned by sampling data from $\mathcal{X} \times \mathcal{Y}$ according to $P(X)$ which comes in the form of $(\{x_i, y_i\})$ pairs.*

Examples of possible decision functions f can be classifiers that categorize natural images in their respective classes or regressors that can predict future values in a time series. The key concept underlying TL is that, differently from the common supervised learning scenario, where the only available information to a model is one domain and one task, there is now access to at least two domains. The one that corresponds to the task we would like to solve called the target-domain $\mathcal{D}_T$, and an additional, possibly related, domain that comes with the name of the source-domain $\mathcal{D}_S$. With all these concepts in place, we can now define Transfer Learning as:

Definition 4 *Given one, or more, observations corresponding to $m^s \in \mathbb{N}^+$ source domain(s) and tasks(s) (i.e., $\{(\mathcal{D}_{S_i}, \mathcal{T}_{S_i})|i = 1 \cdots, m^s\}$) and some additional observation(s) about $m^T \in \mathbb{N}^+$ target domain(s) and task(s) (i.e. $\{(\mathcal{D}_{T_j}, \mathcal{T}_{T_j})|j = 1, \cdots, m^T\}$), transfer learning utilizes the knowledge implied in the source domains to improve the performance of the learned decision functions $f_j^T (j = 1, \cdots, m^T)$ on the target domain(s).*

As pointed out by Pan and Yang [173] the condition that the source and the target domains might be different $\mathcal{D}_S \neq \mathcal{D}_T$ implies that either their respective input spaces are different as well $(\mathcal{X}_S \neq \mathcal{X}_T)$, or that their corresponding marginal distributions are different $(P_s(X) \neq P_T(X))$. Similarly, if the tasks are different instead $\mathcal{T}_S \neq \mathcal{T}_T$, then either one of the output spaces, or the conditional distributions has to be different ($\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$).

Based on the consistency between the source and target input spaces $\mathcal{X}$, and the respective output spaces $\mathcal{Y}$, one can categorize TL into three following settings.

Inductive Transfer Learning This TL scenario is characterized by the fact that the target task $\mathcal{T}_T$ is different from the source task $\mathcal{T}_S$, while the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$ might, or might not be similar. As originally presented in [173] we define inductive transfer learning as:

Definition 5 *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, inductive transfer learning aims to help to improve the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{T}_S \neq \mathcal{T}_T$.*

A key requirement of this type of TL is that some labeled data in the target domain is necessary for inducing the objective predictive function $f_T(\cdot)$. To build some more intuition about this kind of TL, let us assume that we would like to train a model on our target task $\mathcal{T}_T$, which corresponds to recognizing what kind of Japanese letter is depicted in an image. Instead of training a model only on a dataset of letters, we instead start from a model that has already been trained to recognize digits, which will therefore be our source task $\mathcal{T}_S$. Note that in this case, the source task and the target tasks are evidently different $\mathcal{T}_S \neq \mathcal{T}_T$ (classifying digits vs. classifying letters), but that the input space of the source and target domains is the same $\mathcal{X}_S = \mathcal{X}_T$, since it corresponds to black and white images as represented in Fig. 4.2. Please note that in this example, as we use a model that is pre-trained on images representing digits, we assume that we have had access to some labeled data in the source domain in the past. However, the definition of inductive transfer learning does not require labeled data within the source domain to be strictly necessary.

Transductive Transfer Learning Also known as Domain Adaptation [9], this type of TL is characterized by the fact that the source and target tasks are the same $\mathcal{T}_S = \mathcal{T}_T$, but their respective domains are different $\mathcal{D}_S \neq \mathcal{D}_T$. It is also possible that the feature spaces between domains are the same but, if that is the case, then the marginal probability distributions are different $P(X_S) \neq P(X_T)$. In its original formulation, Arnold, Nallapati, and Cohen [9] assumed that all unlabeled data in the target domain is available at training time. However, we hereafter report the definition of Pan and Yang [173] who instead relax this condition and require only a subset of unlabeled target data to be seen at training time.

Definition 6 *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$, and a learning task $\mathcal{T}_T$, transductive transfer learning aims to help to improve the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$.*

Figure 4.2: A visualization of two datasets that can be used for performing inductive transfer learning. On the left we represent instances of the popular MNIST dataset [130], while on the right instances of the Kuzushiji-MNIST dataset [41]. We see that both datasets share the same domain $\mathcal{D}_S = \mathcal{D}_T$ (black and white images), but are associated to different tasks $\mathcal{T}_S \neq \mathcal{T}_T$ (classification of digits vs classification of Japanese letters).

As an example of transductive transfer learning let us assume that we would like to train a model to classify which type of clothing is depicted in an image. This time, differently from the inductive transfer learning case, the images in our target domain $\mathcal{D}_T$ are not black and white anymore, but rather colored images; therefore, defined over the RGB domain (see the right image of Fig. 4.3). We now assume that we have access to a pre-trained model which has already been trained to classify the same type of clothes, with the main difference being that the images constituting the source domain $\mathcal{D}_S$ were black and white images (see the left image of Fig. 4.3). If we now train this pre-trained model on our colored dataset, we see that our setting fits the transductive transfer learning scenario: our considered domains are different $\mathcal{D}_S \neq \mathcal{D}_T$ (colored vs. black and white images), but their respective tasks are the same $\mathcal{T}_S = \mathcal{T}_T$, since a model is always trained to classify types of clothing.

Although this section, as well as the second part of this dissertation, primarily focuses on supervised learning, we hereafter still report for the sake of completeness a definition of unsupervised transfer learning, and see how this kind of TL connects to the types of TL that we have analyzed so far.

Unsupervised Transfer Learning Arguably considered to be the most challenging and the least explored type of TL, unsupervised transfer learning is characterized by the total absence at training time of labeled data in both the source domain and the target domain. As mentioned by Pan and Yang [173], very little research work has so far explored this TL paradigm, with the only existing works exploring typical unsupervised learning topics such as clustering [43, 106, 188]

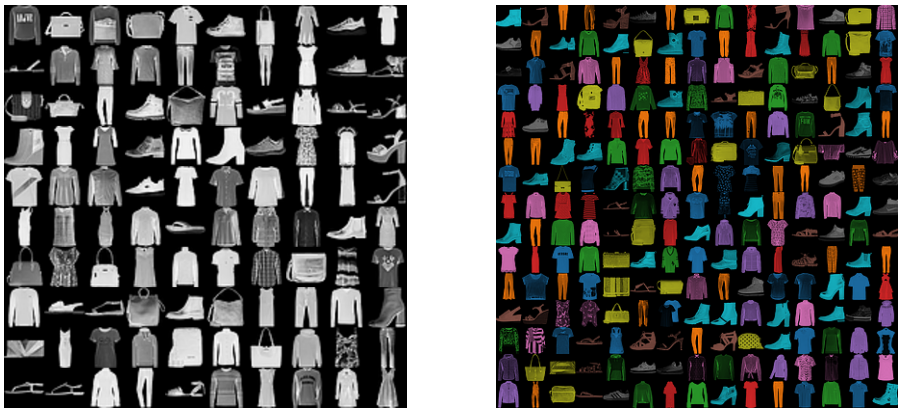


Figure 4.3: Two datasets that are representative of transductive transfer learning. On the left we show images coming from the Fashion-MNIST dataset [284], while on the right we report instances of the same dataset that are colored. In this case tasks among datasets are shared $\mathcal{T}_S = \mathcal{T}_T$ (classification of clothes), but the respective images come from different domains $\mathcal{D}_S \neq \mathcal{D}_T$ (black and white vs RGB images).

and dimensionality reduction [268, 296, 297]. Unsupervised transfer learning is defined as follows:

Definition 7 *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$, and a learning task $\mathcal{T}_T$, unsupervised transfer learning aims to help to improve the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ by using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{T}_S \neq \mathcal{T}_T$ and $\mathcal{Y}_S$ and $\mathcal{Y}_T$ are not observable.*

Based on this definition, we can note that unsupervised transfer learning is more similar to inductive transfer learning than to transductive transfer learning as we again assume that the source and the target tasks are different $\mathcal{T}_S \neq \mathcal{T}_T$.

4.3.2 *Reinforcement Learning*

When it comes to the Reinforcement Learning (RL) setup, the TL definition mentioned above slightly changes and becomes arguably less general. Recall from Chapter 3 that in RL, the main goal is that of training an agent such that it becomes able to interact with its environment, a problem that is modeled with Markov Decision Processes (MDP). It follows that in the RL context, the previously introduced concept of domain $\mathcal{D}$ (which could come in numerous flavors) now comes in the arguably more strictly form of an MDP $\mathcal{M}$. Just like domains, MDPs can either be representative of a source task, $\mathcal{M}_S$, or of a target task $\mathcal{M}_T$, with the latter case corresponding to the main RL problem we are interested in solving. The previously introduced predictive function $f(\cdot)$ now corresponds to the task of learning an

optimal policy π^* for $\mathcal{M}_T$. Based on these concepts, we give the following definition of TL for reinforcement learning that is adapted from the one proposed by Zhu, Lin, and Zhou [299].

Definition 8 *Given a source MDP $\mathcal{M}_S$ and a target MDP $\mathcal{M}_D$, transfer learning in reinforcement learning aims to learn an optimal policy π^* for $\mathcal{M}_D$ by exploiting some prior knowledge related to $\mathcal{M}_S$, denoted as $\mathcal{K}_S$, together with the knowledge that underlies $\mathcal{M}_T$, $\mathcal{K}_T$ such that:*

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\mu_0^t, a\sim\pi}\left[Q^{\pi}_{\mathcal{M}}(s,a)\right], \tag{4.1}$$

where $\pi = \zeta(\mathcal{K}_S \sim \mathcal{M}_S, \mathcal{K}_T \sim \mathcal{M}_T) : \mathcal{S}^t \to \mathcal{A}^t$ is a function mapping from the states to actions for $\mathcal{M}_T$ learned thanks to both $\mathcal{K}_S$ and $\mathcal{K}_T$.

Note that differently from the supervised learning case, we are now making explicit use of the concept of knowledge $\mathcal{K}$, which is what we would like to retain when moving from a source MDP $\mathcal{M}_S$ to a target MDP $\mathcal{M}_T$. We do this because RL is a machine learning paradigm that is arguably, more complex than supervised learning. A complexity that stems from the fact that in RL there are concepts such as e.g., rewards and policies, which are, by definition, not present in the supervised learning setup. As a result, $\mathcal{K}$ can come in forms that it cannot take in SL, and correctly identifying which kind of knowledge to transfer between $\mathcal{M}_S$ and $\mathcal{M}_T$ is just as important as developing a method that successfully transfers this knowledge in the first place. As mentioned by Lazaric [125] $\mathcal{K}$ can come in the following forms.

Transfer of Instances In this scenario $\mathcal{K}$ corresponds to RL trajectories coming in the form $\langle s_t, a_t, r_t, s_{t+1}\rangle$ and that have been collected on one, or possibly multiple, source MDPs $\mathcal{M}_S$. Such trajectories can then be used both in a model-based RL setting, as done by Taylor, Jong, and Stone [248], or for speeding up the process of learning a value function as described in [127] and [124]. Ideally, transferring RL trajectories should result in highly sample efficient algorithms, although it is worth noting that this property, albeit desirable, can constrain the source task $\mathcal{M}_S$ and the target task $\mathcal{M}_T$ to have similar transition models and reward functions. This instance of TL is usually used within the batch RL setup, where gathering experience samples for $\mathcal{M}_T$ can be particularly expensive or time-consuming, which is a constraint that usually does not hold for $\mathcal{M}_S$. The typical challenge then consists of correctly identifying which of the samples coming from $\mathcal{M}_S$ are the most informative ones for solving $\mathcal{M}_T$, as for example, studied by Tirinzoni et al. [254].

Transfer of Parameters As we have seen in Chapter 3 it is often desirable to integrate RL algorithms with parametric function approximators. The main goal is to then train these parametric functions to

learn an approximation of an optimal value function or policy. When parameter transfer is performed, the main idea is to start solving the problem modeled by the target task $\mathcal{M}_T$ with a function that, instead of being initialized with random parameters, is initialized with parameters that have been learned on a certain source task $\mathcal{M}_S$. Examples of knowledge $\mathcal{K}$ that fit this description are the parameters θ of a pre-trained Deep Q-Network, or the parameters that model an Actor-Critic agent.

While both representations are arguably the most popular ones, it is important to mention that as described by Tirinzoni, Sanchez, and Restelli [253] there are alternative possible ways of representing $\mathcal{K}$. Among such ways $\mathcal{K}$ can come in the form of e.g., features [13, 151], rewards [116, 215] and options [229].

4.4 DEEP TRANSFER LEARNING

We now present the field of 'Deep Transfer Learning" (DTL), a machine learning paradigm that aims at performing TL when a source model comes in the form of a pre-trained convolutional neural network. We start by describing how one can exploit the availability of a pre-trained network when training a model on the desired target task and then present a thorough literature review that describes the most successful applications that the DTL community has so far achieved.

4.4.1 *General Framework*

Let us assume we would like to train a neural network on a regression task. Instead of initializing its weights randomly, we initialize it with weights that have already been optimized on a certain source task. We refer to the parameters of this pre-trained model as θ_S, where S stands again for 'source". When training the network on the target task, the main challenge revolves around deciding up to what extent the parameters θ_S should be modified through stochastic gradient descent. In practice, this corresponds to deciding how much of the knowledge contained in θ_S should be retained when training the pre-trained network on $\mathcal{T}_T$. Typically, one can choose between three different approaches:

- Off-the-Shelf Extraction: in this setting, the parameters θ_S of the pre-trained model are not changed when the network gets trained on the target task. Instead, the only weights that get optimized are the ones that are responsible for the final predictions of the model. In the regression example mentioned above, these weights would correspond to the ones parametrizing the network's final output neuron. Similarly, if we would be dealing with a classification problem, we would only train the weights

that constitute the final softmax layer of the model, as this is the part of the network that, as described in Chapter 2, is responsible for outputting the predicted output classes. Since this approach does not involve any backpropagation operations, it is particularly desirable when computational resources are limited. In fact, one only needs to compute the forward pass in order to get the features from the pre-trained model. However, this approach also comes with the major limitation of not allowing the network to adapt to the target task as it assumes that all the knowledge that is required for solving $\mathcal{T}_T$ is already contained within θ_S. In the context of convolutional neural networks, this corresponds to a model that has already learned all the features that are necessary for solving $\mathcal{T}_T$ when getting trained on $\mathcal{T}_S$.

- Fine-Tuning: when this approach is adopted all the parameters θ_S that have been learned on the source task get optimized when training the network on the target task. Evidently, this training strategy is computationally more expensive than the previous one, as it involves all the training steps that characterize neural networks discussed in Chapter 2. Despite being computationally more demanding, fine-tuning a network has the significant benefit of allowing a model to become target task-specific. In fact, whilst training, part of the knowledge contained within θ_S can be 'forgotten", which will result in a new set of parameters, θ_T, that can perform better on $\mathcal{T}_T$ than θ_S. From a practical perspective, however, it is important to train the model in such a way that the knowledge contained within θ_S does not get 'forgotten" too quickly, while at the same time ensuring that the network stays flexible enough for successfully learning the target task. One possible way of achieving this is by using small learning rate values for training.

- Intermediate Approaches: the aforementioned approaches do either not modify the source weights θ_s parametrizing the convolutional layers at all, or instead allow the network to completely change them. However, some intermediate appraoches are also possible. As convolutional networks typically come with a large number of layers, one possible way of exploiting the respective source weights θ_s could be done by fine-tuning only a specific sub-set of layers, while keeping the remaining ones frozen. As a result only one part of the model will involve the backpropagation algorithm, whereas the remaining parts will simply act as feature extractors. Furhtermore it is also worth noting that an off-the-shelf feature extraction approach can technically be performed after any convolutional layer. As features can be obtained after any convolutional layer, some interme-

diate approaches rely on extracting them without performing a forward pass throughout the entire network. While such approaches can certainly be valuable [164, 214], throughout this thesis they will not be considered.

While all approaches come with pros and cons, the second option is usually preferred if enough computational resources are available. In fact, as we will see in the coming section, the community seems to agree that it often results into better final performance. When DTL strategies are adopted it is usually good practice to compare the performance of a pre-trained model with the performance that is obtained by a model that is initialized randomly, and that gets therefore trained from scratch. Throughout this thesis we will constantly characterize the benefits of adapting TL strategies from this perspective, therefore, to add even more clarity to the concepts presented in this section, we also visually represent them in Fig. 4.4.

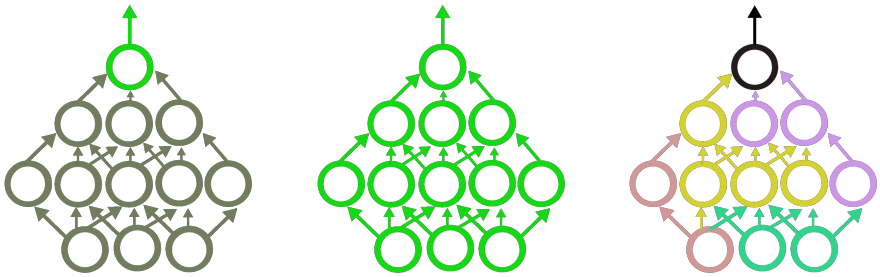


Figure 4.4: A simplified visual representation of the different deep transfer learning training paradigms that are considered throughout this thesis. The first plot represents a model that comes as pre-trained on a source task but which will not update most of its weights during the training stage (represented in gray): the only trainable parameters of this network are the ones that parametrize the final layer of the model and that are represented in green. In the second plot we visually represent a model that is parametrized with weights that have been learned on a certain source task, and that get "unfrozen" when the network gets trained on the target task, therefore defining the model as fully trainable. Lastly we visualize a model that does not come as pre-trained on any source task, and that is therefore initialized with random weights instead (represented by the various colours). As already mentioned throughout this chapter, the main goal of TL is to obtain a model that if trained with the first two approaches results into a final performance that is better than the one that would be obtained by the last type of model.

4.4.2 *Literature Review*

We now review how the deep transfer learning community has over the years studied the transfer learning properties of convolutional neural networks. Specifically, we focus on four different perspectives.

Convolutional Neural Networks as Feature Extractors As soon as convolutional neural networks started to perform well on popular Computer Vision (CV) benchmarks, research investigating whether these networks could be transferred and reused for novel tasks started to bloom. The first work exploring this direction was that of Yosinski et al. [286], who observed that the early layers of deep neural networks trained on natural images, learn features which are general and, therefore, independent from the CV task used for training. Such generalization property can hence be exploited by initializing a convolutional neural network with transferred features instead of randomly, which is a strategy that results in models less prone to overfitting.

Alongside Yosinski et al. [286], Donahue et al. [49] investigated whether features extracted from a convolutional neural network trained for image classification could also be used for tackling CV tasks such as scene recognition and domain adaptation. Donahue et al. [49] showed that this was indeed the case and publicly released the pre-trained model under the name of `DeCAF` intending to stimulate the CV community to investigate further the extent to which the features learned by this network were transferable to novel tasks. Almost concurrently, similar conclusions about the transferability of pre-trained convolutional networks were drawn by Oquab et al. [171], who showed the benefits of using a pre-trained convolutional model as feature extractor when dealing with object and action localization problems, and by Zeiler and Fergus [289] who first showed that on many problems it was more beneficial to simply train the final classification layer of a pre-trained network, than to train a randomly initialized model from scratch. While definitely promising, all these works restricted their experimental analysis to a relatively small set of CV problems, and it was only with the seminal work of Sharif Razavian et al. [224] that the deep learning community realized how powerful pre-trained networks could be. Sharif Razavian et al. [224] used the off the shelf (OTS) features of the pre-trained `OverFeat` network [223] for tackling numerous challenging CV problems and consistently reported a final performance that was superior to the one of the state of the art algorithms of the time. Next to providing a wealth of empirical evidence supporting the use of off-the-shelf features, their work also established the first training protocol for combining high dimensional OTS features with linear classifiers, such as SVMs, and dimensionality reduction techniques such as principal component analysis.

It did not take long before the scientific community started to investigate whether off-the-shelf features could also be used for problems outside the typical CV benchmarks, and therefore fully realized the potential of this TL approach. Among the very first practical applications, we mention the work of Van Ginneken et al. [258] who used the previously mentioned `DeCAF` model for (successfully) tackling the challenging medical task of pulmonary nodule detection. Along the same line of research, equally good results were obtained within the medical domain by Hernandez-Diaz, Alonso-Fernandez, and Bigun [92] who tackled the problem of periocular recognition, and by Nguyen et al. [169] who considered the similar task of iris recognition.

Further successful applications of OTS classification, which go beyond the medical domain, are the ones reported by Sharma et al. [225], who considered the handwriting recognition task of word spotting, and the one of Wolfshaar, Karaaba, and Wiering [281], who studied the task of gender classification. While all these researches solely relied on an OTS feature extraction approach when addressing a CV problem, it is also worth noting that OTS features can be used in combination with more traditional CV feature engineering techniques such as SIFT [144] and HOG [44]. This research direction has been successfully explored by, e.g., Wang, Qiao, and Tang [267] who examined the task of human action understanding, and by Zhong, Sullivan, and Li [293] who addressed the problem of face localization.

On the Benefits of Fine-Tuning Modern deep learning frameworks such as `TensorFlow` [1] and `PyTorch` [177], provide high level and easy to use APIs that make it possible to create and train deep learning models even without a necessarily strong machine learning background. Among the main reasons that have made deep learning so accessible there is the fact that the aforementioned deep learning libraries provide easy access to models that have already been trained on a large variety of CV tasks [54, 138, 200]. As a result, using pre-trained neural networks has become increasingly easy, which is among the reasons that allowed the deep learning community to explore whether fine-tuning pre-trained models could result in better performance than simply using them as OTS feature extractors. It is easy to see how this research question is of high practical interest and why it has therefore been heavily explored by practitioners working at the intersection of machine learning and fields such as medicine [96, 245]. Among the first works exploring whether it is beneficial to fine-tune pre-trained models instead of using them as simple feature extractors, there is the one of Tajbakhsh et al. [245]. The authors consistently show that fine-tuning a pre-trained network outperforms the OTS feature extraction approach when it comes to four distinct medical imaging tasks and that, similarly to what was predicted by Zeiler

and Fergus [289], pre-trained networks outperform models that are trained from scratch. A similar conclusion has also been reached by Mormont, Geurts, and Marée [164], who analyzed the same research question under the lens of image classification problems coming from the digital pathology field. They also show that fine-tuning yields better performance than OTS feature extraction, but they do not answer whether this TL strategy works better than training a network from scratch. The question of whether to fine-tune or not to fine-tune a neural network has also been explored outside of the CV domain. Among the different works, we mention the one of Peters, Ruder, and Smith [183], who address this question from a Natural Language Processing (NLP) perspective. In line with what has been observed by the CV community, they also highlight the significant benefits that can come from fine-tuning popular NLP models such as ELMo [182] and BERT [46] as long as the source task is carefully chosen. By now, studies investigating the benefits coming from fine-tuning pre-trained models are countless and range over a large variety of domains that do not necessarily strictly involve CV problems [3, 26, 45, 48, 67, 98, 100, 115, 288].

On the Role of Imagenet as Source Task Throughout this chapter, we have constantly referred to the concept of source domain $\mathcal{D}_S$ and source task $\mathcal{T}_S$, two key elements without which the entire field of TL would not even exist. Albeit in the previous paragraphs we have mentioned the task of image classification as source task $\mathcal{T}_S$ that can be used for pre-training convolutional networks, we have not explicitly described what this task consists of in practice. When adopting TL strategies for CV problems, the most common and, by far successful, approach is that of relying on models that have been trained for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [200]. The ILSRVC dataset, more commonly referred to as the ImageNet dataset, is a collection of over one million natural images that are categorized among one thousand different classes. Until 2017 it was primarily considered to be the most complex and challenging problem of all CV and is among the main reasons which have encouraged the deep learning community to develop most of the neural architectures that we described in Chapter 2. Due to the large number of samples constituting the dataset and the complexity of the tackled task, networks trained for the ILSVRC challenge are regularly used as pre-trained networks even when the target task $\mathcal{T}_T$ does not involve the classification of natural images. Intuitively, the reasons behind this choice are very straightforward: on the one hand, it is safe to assume that some of the features that are learned by a network that receives as input more than one million images will, at least in part, correspond to the features that the same network would

have to learn when getting trained from scratch on the desired target task.

On the other hand, it is also unlikely that the target task $\mathcal{T}_T$ will be more complex than the source task $\mathcal{T}_S$ since it is not common to deal with classification problems that involve more than 1000 classes. In this sense, as pointed out by Mensink et al. [153], it is reasonable to assume that if a network performs well on $\mathcal{T}_S$, it should also perform well on $\mathcal{T}_T$, as the latter task is essentially easier than the former. Despite these intuitive explanations, a large body of work has studied why it is beneficial to transfer ImageNet pre-trained models and the factors of influence of this dataset for TL. This question was first tackled by Huh, Agrawal, and Efros [103] who studied, among other questions, how many examples and classes of the ImageNet dataset should be used for successfully pre-training a model. Perhaps surprisingly, they found that already half of the ImageNet data yielded a well-performing pre-trained network and that among the different 1000 classes, it was already enough to pre-train a network on a subset of 127 of them. Kornblith, Shlens, and Le [118] empirically studied the benefits of using ImageNet pre-trained models for 12 different classification problems and found that the better a model performs on ImageNet, the better it transfers to new unseen tasks. A similar study, which, however, yielded slightly different results, is the one of He, Girshick, and Dollár [87] who showed that there might not be significant differences in terms of final performance between using an ImageNet pre-trained network and a randomly initialized one, but that the first ones consistently converge faster than the latter. Finally, the recent work of Mensink et al. [153] shows that ImageNet pre-trained models always outperform models that are trained from scratch, but that this dataset might not necessarily always be the best possible source task for pre-training. It is worth noting that all the examples mentioned earlier revolve around the field of supervised learning within CV, it naturally follows that different pre-training strategies, and therefore different source domains, need to be considered for other fields (see e.g. NLP.). Since discussing these techniques would go beyond the main scope of this dissertation, we do not present them here. However, we refer the reader to [29, 46, 155, 196] for a discussion of pre-training strategies outside the CV and supervised learning domain.

The Deep Reinforcement Learning Setting While within the supervised learning setting, the body of work studying the transfer learning properties of neural networks is substantial, the same cannot be said when it comes to reinforcement learning. Although, as presented in Sec. 4.3.2 there exist many different approaches for performing TL in RL, the integration of such techniques with convolutional neural networks is much rarer. Perhaps the work that studies

the TL properties of Deep Q-Networks in a flavor that is the closest to the TL approaches used in supervised learning presented in Fig. 4.4 is that of Farebrother, Machado, and Bowling [55]. The authors study whether convolutional neural networks that are trained with the DQN algorithm [161] are capable of learning features that are robust enough to allow the algorithm to generalize across different tasks. The results, obtained on four different Atari 2600 games do not provide a clear answer to this question: when Deep Q-Networks come as pre-trained on a particular game and simply get fine-tuned on a new different game, the authors do not observe any of the benefits that this TL approach typically yields in the supervised learning context. However, if networks get pre-trained in combination with typical supervised learning regularization techniques such as dropout [232] and l_2 regularization, then the authors observe that fine-tuning these models results in a final performance that is better than the one of models trained from scratch. While indeed encouraging, these results were only obtained on a minimal set of RL environments, and it is unclear whether these conclusions would hold if a more extensive set of benchmarks, and algorithms, would be tested. A similar study, which arguably presents the same limitations, is the one performed by Tyo and Lipton [257]. On the same line with Farebrother, Machado, and Bowling [55], the authors study the effect of fine-tuning different pre-trained Rainbow agents [95] that use different weight initialization strategies. Results reported on three different Atari games show that fine-tuning is beneficial only for one single game but do not explain the TL properties of pre-trained Deep Q-Networks any further. A more thorough and successful study is the one presented by Parisotto, Ba, and Salakhutdinov [174], where the authors also investigate the effect of fine-tuning pre-trained DRL agents and show that this TL strategy can result in significant benefits. Their study, however, presents some significant differences when compared to the previous works. First, the DRL algorithm under scrutiny is a policy gradient algorithm which, as discussed in Chapter 3, is part of a family of techniques that is significantly different from the family that DQN and Rainbow are part of. Second, their proposed Actor-Mimic algorithm does not come as pre-trained on a single Atari game anymore but is pre-trained in a multi-task learning setting instead. The algorithm, therefore, deals with different source-tasks $\mathcal{T}_S$ during the pre-training stage, which is a strategy that arguably can result in algorithms that are more robust and suitable for TL [114].

4.5 RELEVANCE FOR THIS DISSERTATION

With all the concepts above and the ones presented in Chapters 2 and 3, we are now ready to end this first part of this dissertation

by describing how all the aforementioned content will play a role throughout the rest of this thesis.

TAKEAWAY OF PART I

First, we start by noting that all the quantitative results that will be presented from now on will be defined with respect to the three transfer learning benefits that we described in Sec. 4.2. No matter which kind of task we will be training neural networks for, we will always seek to identify at least one of the three possible transfer learning benefits. The only chapter where we will not do this is Chapter 8, which instead, will serve for introducing some novel deep reinforcement learning algorithms that will be studied from a transfer learning perspective only in Chapter 9.

When it comes to the research performed within the supervised learning setting (Chapters 5, 6 and 7), it is important to note that we will only consider the inductive transfer learning scenario. More specifically, we will always study the extent to which neural networks pre-trained on natural images can generalize to non-natural datasets. The content of these datasets, and therefore the considered target tasks $\mathcal{T}_T$, will change from chapter to chapter, and so will the source task $\mathcal{T}_S$ that will be used for pre-training. For the research involving reinforcement learning, we will instead only consider the TL setting that studies the transferability of parameters (Chapter 9) presented in Sec. 4.3.2. As a result, no matter whether we will tackle supervised learning problems or reinforcement learning ones, the experimental protocol that we will adopt will always follow the TL training strategies that we have described in Sec. 4.4.1 and presented in Fig. 4.4. As a result, the respective studies will all contribute to the development of the field that studies the transferability of neural networks that we have reviewed in Sec. 4.4.

We will now present our research investigating the TL properties of convolutional neural networks trained for supervised learning problems.

Part II

TRANSFER LEARNING FOR DEEP SUPERVISED LEARNING

5

ON THE TRANSFERABILITY OF CONVOLUTIONAL NETWORKS

Contributions and Outline

We now present a first study that characterizes the Transfer Learning (TL) properties of convolutional neural networks that come as pre-trained on the ImageNet dataset. We thoroughly investigate whether popular neural networks that have obtained state of the art results on this benchmark of natural images can perform equally well once used on non-natural datasets. Specifically, we explore whether it is possible to tackle three different target tasks $\mathcal{T}_T$ that come in the form of art classification problems. We study the effects of different TL training approaches (off-the-shelf classification vs. fine-tuning), and explore whether it is possible to improve the TL performance of the considered pre-trained networks by allowing these models to have access to source domains $\mathcal{D}_S$ other than the ImageNet dataset exclusively. The chapter is structured as follows: we start by providing the reader with some background information in Sec. 5.1. In Sec. 5.2 we present a brief theoretical reminder of the field of TL, a description of the datasets that we have used, and the methodological details about the experiments that we have performed. In Sec. 5.3 we present and discuss our results. A summary of the main contributions of this chapter is finally presented in Sec. 5.4.

This chapter is adapted from the publication Sabatelli et al. [206].

5.1 A FIRST EMPIRICAL STUDY

In the first part of this thesis, we have seen that convolutional neural networks have become a crucial component in today's machine learning toolbox. Thanks to their ability to automatically learn relevant features, these neural networks can successfully be used for tackling both supervised learning problems, as well as reinforcement learning ones. Specifically, as reviewed in Chapter 4, thanks to the field of deep transfer learning it is now possible to find a broad range of successful applications that see the usage of convolutional networks even outside the domain these networks were originally developed for, namely natural images [117]. Despite many successfull examples, there still are some domains for which their applicability, and therefore potential transfer learning properties, have not been explored.

A promising research field in this sense is that of Digital Heritage [176]. Due to a growing and rapid process of digitization, museums have started to digitize large parts of their cultural heritage collections, leading to the creation of several digital open datasets [6, 154]. The images constituting these datasets are mostly matched with descriptive metadata, which, as presented by Mensink and Van Gemert [154], can be used for defining a set of challenging machine learning tasks. However, the image samples in these datasets are very different in terms of quantity, size, and resolution from the images that typically constitute popular computer vision benchmarks; therefore, the computer vision potential of convolutional networks in this domain is largely unknown. In this chapter, we address this research question and present a first, thorough, empirical study that explores the potential that convolutional neural networks have to offer when transferred to the artistic domain. The next section moves towards providing the reader with a brief formal reminder of TL. We then introduce the three classification problems that are considered in our study, together with a brief description of the datasets. Finally, we present the neural architectures that we have used for the experiments.

5.2 METHODOLOGY

5.2.1 *Transfer Learning*

As seen in Chapter 2, a supervised learning (SL) problem can be identified by three elements: an input space $\mathcal{X}_T$, an output space $\mathcal{Y}_T$, and a probability distribution $P_T(x, y)$ defined over $\mathcal{X}_T \times \mathcal{Y}_T$ (where T stands for 'target', as this is the main problem we would like to solve). The goal of SL is then to build a function $f : \mathcal{X}_T \to \mathcal{Y}_T$ that minimizes the expectation over $P_T(x, y)$ of a given loss function ℓ assessing the predictions made by f:

$$E_{(x,y)\sim P_t(x,y)}\{\ell(y, f(x))\}, \tag{5.1}$$

where the only information available to build this function is a learning sample of input-output pairs $LS_T = \{(x_i, y_i)|i = 1, \ldots, N_T\}$ drawn independently from $P_T(x, y)$. As introduced in Chapter 4, in the general transfer learning setting, one assumes that an additional dataset LS_S, called the source data, is available that corresponds to a different, but related, SL problem. More formally, the source SL problem is assumed to be defined through a triplet $(\mathcal{X}_S, \mathcal{Y}_S, P_S(x, y))$, where at least either $\mathcal{X}_S \neq \mathcal{X}_T$, $\mathcal{Y}_S \neq \mathcal{Y}_T$, or $P_S \neq P_T$. The goal of TL is then to exploit the source data LS_S together with the target data LS_T to potentially find a better model f in terms of the expected loss (5.1) than when only LS_T is used for training this model. We have seen that depending on the availability of labels in the target and source data and on how the source and target problems differ, one can distinguish

different TL settings (see Sec. 4.4.1 of Chapter 4). In what follows, we assume that labels are available in both the source and target data and that the input spaces $\mathcal{X}_T$ and $\mathcal{X}_S$, that both correspond to color images, match. However, output spaces and joint distributions will differ between the source and target problems, as they will correspond to different classification problems (ImageNet object recognition versus art classification tasks). We, therefore, consider the inductive transfer learning setup and assume that information between the source and target problems is exchanged in the form of a neural network that comes as pre-trained on the source data.

5.2.2 *Datasets and Target Tasks $\mathcal{T}_T$*

For our experiments, we use two datasets that come from two different heritage collections. The first one contains the largest number of samples, comes from the Rijksmuseum in Amsterdam, and corresponds to the first version of the dataset released in 2014 under the name "The Rijsmuseum Challenge" [154]. Our second 'Antwerp' dataset is much smaller. This dataset presents a random sample that is available as open data from a larger heritage repository: DAMS (Digital Asset Management System)[1]. This repository can be searched manually via the web interface or queried via a Linked Open Data API. It aggregates the digital collections of the foremost GLAM institutions (Galleries, Libraries, Archives, Museums) in the city of Antwerp in Belgium. Thus, this dataset presents a varied and representative sample of the sort of heritage data nowadays being collected at the level of individual cities across the globe. While it is much smaller, its coverage of cultural production is similar to that of the Rijksmuseum dataset and presents an ideal testing ground for the transfer learning task under scrutiny here. Both image datasets come with metadata encoded in the Dublin Core metadata standard [273]. We selected three well-understood classification problems:

- Material classification: which consists in identifying the material the different heritage objects are made of (e.g., paper, gold, porcelain, ...);
- Type classification: in which the neural networks have to classify in which artistic category the samples fall into (e.g., print, sculpture, drawing, ...);
- Artist classification: where the main goal is to match each sample of the dataset with its creator.

As the goal is to tackle these classification problems through TL, we will refer to them as $\mathcal{T}_T$ ①, $\mathcal{T}_T$ ② and $\mathcal{T}_T$ ③ respectively. As reported in Table 5.1 we can see that the Rijksmuseum collection is the

1 https://dams.antwerpen.be/

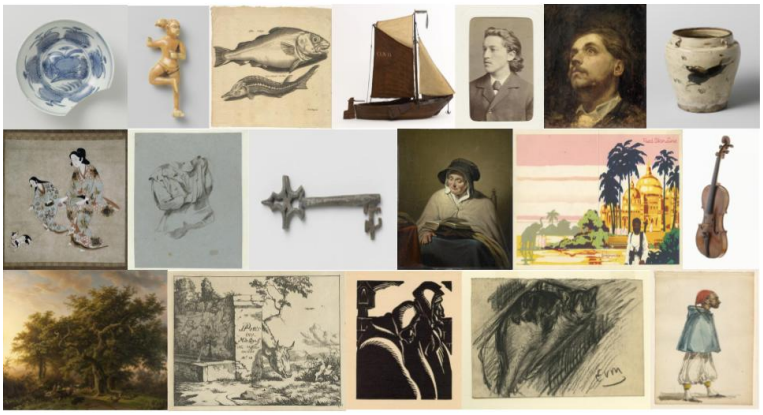


Figure 5.1: A visualization of the images that are used for our experiments. It is possible to see how the samples range from images representing plates made of porcelain to violins, and from Japanese artworks to a more simple picture of a key.

dataset with the largest amount of samples per target task (N_t) and the highest amount of labels to classify (Q_t). Furthermore, it is also worth noting that there was no metadata available when it comes to the first classification task for the Antwerp dataset (as marked by the $-$ symbol), and that there are some common labels between the two heritage collections when it comes to the classification of types ($\mathcal{T}_T$ ②). A visualization reporting some of the images that are present in both datasets is shown in Fig. 5.1.

Table 5.1: An overview of the two datasets that are used in our experiments. For each heritage collection we report with N_t the amount of samples constituting the datasets and with Q_t the number of labels. Lastly, we also report if there are common labels between the two heritage collections.

$\mathcal{T}_T$	Dataset	N_t	Q_t	% of overlap
Material ①	Rijksmuseum	110,668	206	∅
	Antwerp	–	–	
Type ②	Rijksmuseum	112,012	1,054	
	Antwerp	23,797	920	≈ 15%
Artist ③	Rijksmuseum	82,018	1,196	∅
	Antwerp	18,656	903	

We use 80% of the datasets for training while the remaining $2 \times 10\%$ is used for validation and testing respectively. Furthermore, we ensure that only classes which occur at least once in all the splits are used for our experiments. Naturally, in order to keep all comparisons fair between neural architectures and different TL approaches, all experiments have been performed on the exact same data splits.

5.2.3 *Convolutional Networks and Training Approaches*

For our experiments, we use four pre-trained convolutional networks that were reviewed in Chapter 2 and that have all obtained state-of-the-art results on the ImageNet classification challenge. These neural architectures are VGG19 [228], Inception-V3 [244], Xception [38] and ResNet50 [285]. We use the implementations of the networks that are provided by the `Keras` Deep Learning library [39] together with their appropriate `Tensorflow` weights [1] that come from the `Keras` official repository as well. Since all architectures have been built in order to deal with the ImageNet dataset, we replace the final classification layer of each network with a new one. This final layer simply consists of a new softmax output, with as many neurons as there are classes to classify, which follows a 2D global average pooling operation. We rely on this dimensionality reduction step because we do not add any fully connected layers between the last convolution layer and the softmax output. Hence, in this way, we are able to obtain a feature vector, $\mathscr{X}$, out of the rectified activation feature maps of the network that can be properly classified. Since all experiments are treated as a multi-class classification problem, all networks minimize the categorical cross-entropy loss function. We investigate the potential of the TL approaches that were reviewed in Chapter 4, which, as a reminder, are: the off-the-shelf classification approach, which only trains the final softmax classifier on $\mathscr{X}$ retrieved after performing one forward pass of the image through the network and the fine-tuning approach, which differs from the previous one by the fact that together with the final softmax output the entire network is trained as well. From now on, we refer to the networks trained with the off-the-shelf classification approach as θ_i^-, while to the fine-tuned networks simply as θ_i, where i stands for the source task $\mathcal{T}_S$ these models have been trained on, namely the ImageNet (i) dataset. In order to maximize the performance of all models, we follow some of the recommendations presented by Masters and Luschi [149] and train the networks with a relatively small batch size of 32 samples. We do not perform any data augmentation operations besides a standard pixel normalization to the $[0, 1]$ range and a re-scaling operation that resizes the images to the input size that the different models require. Regarding the stochastic optimization procedures of the different classifiers, we use two different optimizers, that after preliminary experiments, turned out to be the best-performing ones. For the off-the-shelf approach we use the RMSprop optimizer [252] which has been initialized with its default hyperparameters (learning rate = 0.001, a momentum value $\rho = 0.9$ and $\epsilon = 1e - 08$). On the other hand, when we fine-tune the models, we use the standard Stochastic Gradient Descent (SGD) algorithm with the same learning rate, 0.001, and a Nesterov Momentum value [168] set to 0.9. Training has been controlled by the early stop-

ping method [34] which interrupted training as soon as the validation loss did not decrease for 7 epochs in a row. The model which is then used on the testing set is the one that obtained the smallest validation loss while training.

5.3 RESULTS

Our experimental results are divided into two sections, depending on which kind of dataset has been used. We first report the results that we have obtained when using architectures that were pre-trained on the ImageNet dataset only and aimed to tackle the three classification tasks of the Rijksmuseum dataset that were presented in Section 5.2.2. We report these results in Section 5.3.1 where we explore the benefits of using the ImageNet dataset as source domain $\mathcal{D}_S$ only, and how well such pre-trained models generalize when it comes to the artistic target domain. We then present the results from classifying the Antwerp dataset, using models that are both pre-trained on the ImageNet dataset and on the Rijksmuseum collection in Section 5.3.3. We investigate whether these neural architectures, which have already been trained to tackle art classification problems before, perform better than those trained on the ImageNet dataset only. All results show comparisons between the off-the-shelf classification approach and the fine-tuning scenario. In addition to that, in order to establish the potential benefits that TL from ImageNet has over training a model from scratch, we also report the results that have been obtained when training a network with weights that have been initially sampled from a He-Uniform distribution [89]. Since we take advantage of the work presented by Bidoia et al. [23] we use the Inception-V3 architecture. We refer to it in all figures as Scratch-V3 and always visualize it with a solid orange line. Fig. 5.2 and Fig. 5.3 report the performance in terms of accuracy (%) that the models have obtained on the validation sets. While the performance that the neural architectures have obtained on the final testing set are reported in Tables 5.2 and 5.3.

5.3.1 *From Natural to Non Natural Images*

The first results that we report have been obtained on $\mathcal{T}_T$ ①, namely the material classification task. We believe that this can be considered as the easiest classification task within the ones that we have introduced in Section 5.2.2 for two main reasons: first, the number of possible classes the networks have to deal with is more than five times smaller when compared to the other two challenges; second, we also believe that this classification task is, within limits, the most similar one when compared to the original ImageNet challenge. Hence, the features that might be useful to classify the different natural images on the latter classification testbed might not be too dissimilar from

the ones needed to properly recognize the material that the different samples of the Rijksmuseum collection are made of. If this were the case, we would expect a very similar performance between the off-the-shelf classification approach and the fine-tuning one. Comparing the learning curves of the two classification strategies in Fig. 5.2, we observe that the fine-tuning approach leads to significant improvements when compared to the off-the-shelf one for three architectures out of the four tested ones. Note, however, that, in support of our hypothesis, the off-the-shelf approach can still reach high accuracy values on this problem and is also competitive with the model trained from scratch, with the crucial difference being that these models result in faster training as jumpstart improvements can be observed. This suggests that features extracted from networks pretrained on ImageNet are relevant for the target task $\mathcal{T}_T$ of material classification.

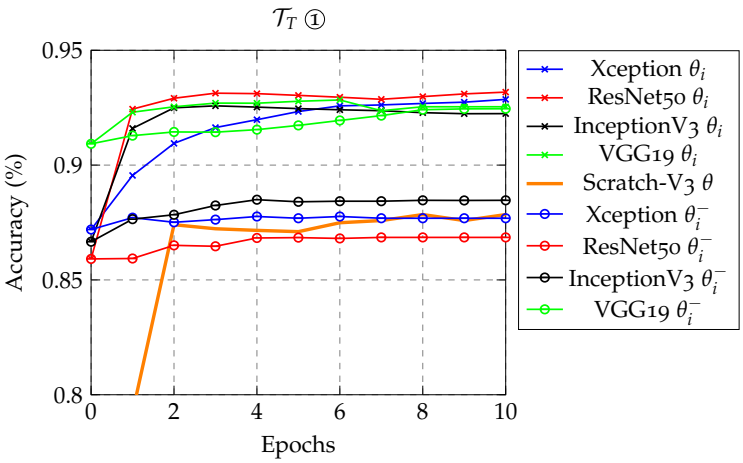


Figure 5.2: Comparison between the fine-tuning approach (θ_i) versus the off-the-shelf one (θ_i^-) when classifying the material of the heritage objects of the Rijksmuseum dataset. We can observe that for three out of four neural architectures the first approach leads to significant improvements when compared to the latter one. Furthermore, we can also observe that training a randomly initialized model from scratch (solid orange line) leads to worse results than fine-tuning a network that comes as pre-trained on the ImageNet dataset.

We can also observe that the ResNet50 architecture is the architecture that, when fine-tuned, performs overall best compared to the other three models. This happens despite it being the network that initially performed worse as a simple feature extractor in the off-the-shelf experiments. As reported in Table 5.2 we can see that this kind of behavior reflects itself on the separated testing set as well, where it obtained the highest testing set accuracy when fine-tuned (92.95%), and the lowest one when the off-the-shelf approach was used (86.81%). It is worth noting that the performance between the different neural architectures do not strongly differ between each other once they are

fine-tuned, with all models performing around $\approx$ 92% on the final testing set. Furthermore, special attention needs to be given to the VGG19 architecture, which does not seem to benefit from the fine-tuning approach as much as the other architectures do. In fact, its off-the-shelf performance on the testing set (92.12%) is very similar to its fine-tuned one (92.23%). This suggests that this neural architecture is the only one that, in this task, and when pre-trained on ImageNet, can successfully be used as a simple feature extractor without relying on complete retraining.

When analyzing the performance of the different neural architectures on $\mathcal{T}_T$ ② (type classification) and $\mathcal{T}_T$ ③ (artist classification), respectively the left and right plots reported in Fig. 5.3, we observe that on these problems the fine-tuning strategy leads to even more significant improvements when compared to what we observed in the previous experiment. The results obtained on the second task show again that the ResNet50 architecture is the architecture that leads to the worse results if the off-the-shelf approach is used (its testing set accuracy is as low as 71.23%), and similarly to what has been observed before, it then becomes the best performing model when fine-tuned, with a final accuracy of 91.30%. Differently from what has been observed in the previous experiment, the VGG19 architecture, despite being the network performing best when used as off-the-shelf feature extractor, this time performs significantly worse when it is fine-tuned, which highlights the benefits of this latter training approach. Similar to what has been observed before, our results are again not significantly favoring any fine-tuned neural architecture, with all final accuracies being around $\approx$ 91%.

If the so far considered target tasks have highlighted the significant benefits of the fine-tuning approach over the off-the-shelf one, it is also important to note that the latter approach is still able to yield satisfying results. In fact, a final accuracy of 92.12% has been obtained when using the VGG19 architecture for tackling $\mathcal{T}_T$ ①, and the same architecture reached a classification rate of 77.33% on $\mathcal{T}_T$ ②. Despite the latter accuracy being very far in terms of performance from the one obtained when fine-tuning the network (90.27%), these results still show that models pre-trained on ImageNet do learn particular features that can also be used for classifying the material and the type of heritage objects. In fact, jumpstart improvements were observed in Fig. 5.2 as well as in the left plot of Fig. 5.3.

When considering the third target task, we can however observe that these conclusions partially change: the Xception, ResNet50, and Inception-V3 architectures all perform extremely poorly if not fine-tuned, with the latter two models not reaching a 10% classification rate. Better results are obtained when using the VGG19 architecture, which reaches a final accuracy of 38.11%. Most importantly, the performance of each model is again significantly improved when the

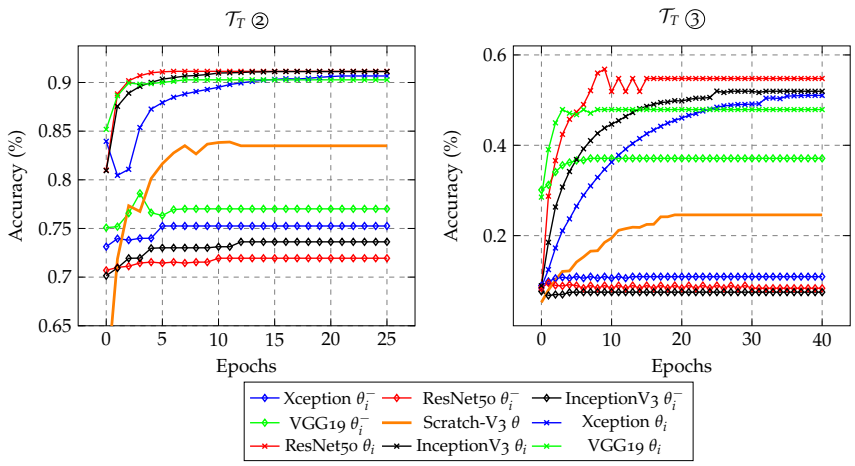


Figure 5.3: A similar analysis as the one which has been reported in Figure 5.2 but for the second and third classification tasks (left and right figures respectively). The results show again the significant benefits that fine-tuning (reported by the dashed line plots) has when compared to the off-the-shelf approach (reported by the dash-dotted lines) and how this latter strategy miserably underperforms when it comes to artist classification. Furthermore we again see the benefits that using a pre-trained model has over training the architecture from scratch (solid orange line).

networks are fine-tuned. As already observed in the previous experiments, ResNet50 outperforms the other architectures on the validation set. However, on the test set (see Table 5.2), the overall best performing network is Inception-V3 (with a final accuracy of 51.73%), which suggests that ResNet50 suffered from overfitting. It is important to state two major important points about this set of experiments. The first one relates to the final classification accuracy that is obtained by all models, and that at first sight might seem disappointing. While it is true that these classification rates are significantly lower when compared to the ones obtained in the previous two experiments, it is important to highlight how a large set of artists present in the dataset are associated to a minimal amount of samples. This reflects a lack of appropriate training data, which does not allow the models to learn all the necessary features for successfully dealing with this particular classification challenge. In order to do so, we believe that more training data is required. Moreover, it is worth pointing out that despite performing very poorly when used as off-the-shelf feature extractors, ImageNet pre-trained models do still perform better once they are fine-tuned than a model that is trained from scratch, as asymptotic improvements were observed in all our experiments. This suggests that these networks do learn potentially representative features when it comes to the classification of artists, but in order to correctly classify them, fine-tuning is required.

Table 5.2: An overview of the results obtained by the different models on the testing set when classifying the heritage objects of the Rijksmuseum. The overall best performing architecture is reported in a green cell, while the second best performing one is reported in a yellow one. The additional columns "Params" and "$\mathscr{X}$" report the amount of parameters the networks have to learn and the size of the feature vector that is used as input for the softmax classifier.

$\mathcal{T}_T$	model	off the shelf	fine tuning	Params	$\mathscr{X}$
①	Xception	87.69%	92.13%	21K	2048
①	InceptionV3	88.24%	92.10%	22K	2048
①	ResNet50	86.81%	92.95%	24K	2048
①	VGG19	92.12%	92.23%	20K	512
②	Xception	74.80%	90.67%	23K	2048
②	InceptionV3	72.96%	91.03%	24K	2048
②	ResNet50	71.23%	91.30%	25K	2048
②	VGG19	77.33%	90.27%	20K	512
③	Xception	10.92%	51.43%	23K	2048
③	InceptionV3	.07%	51.73%	24K	2048
③	ResNet50	.08%	46.13%	26K	2048
③	VGG19	38.11%	44.98%	20K	512

5.3.2 *Discussion*

In the previous section, we have investigated whether four different architectures pre-trained on the ImageNet dataset can be successfully used to address three art classification problems. We have observed that this is particularly the case when it comes to classifying the material and the type, where in fact, an off-the-shelf classification approach already yielded satisfactory results. However, most importantly, we have also shown that the performance of all models can be significantly improved when the networks are fine-tuned and that an ImageNet initialization is beneficial when compared to training a randomly initialized network from scratch. Furthermore, we have also shown that ImageNet pre-trained models can still perform extremely poorly when they are used as simple feature extractors (as demonstrated by the experiments reported on $\mathcal{T}_T$ ③). In the next section, we explore the performance of fine-tuned models trained to tackle two of the already seen target tasks on a different heritage collection. For this problem, we will again compare the off-the-shelf approach with the fine-tuning one.

5.3.3 *From One Target Domain $\mathcal{D}_T$ to Another*

Table 5.3 compares the results that we obtained on the Antwerp dataset when using ImageNet pre-trained models (θ_i) versus the same architectures that were fine-tuned on the Rijksmuseum dataset (θ_r). While looking at the performance of the different neural architectures, two interesting results can be highlighted. First, models which have been fine-tuned on the Rijksmuseum dataset outperform the ones pre-trained on ImageNet on both target tasks $\mathcal{T}_T$. This happens to be the case both when the networks are used as simple feature extractors and when they are fine-tuned. On $\mathcal{T}_T$ ②, this result is not surprising since, as discussed in Section 5.2.2, the types corresponding to the heritage objects of the two collections partially overlap. This is, however, more surprising when it comes to the artist classification tasks $\mathcal{T}_T$ ③ as there is no overlap at all between the artists of the Rijksmuseum and the ones from the Antwerp dataset. A second interesting result, which is consistent with the results presented in the previous section, revolves around the observation that it is always beneficial to fine-tune the networks over just using them as off-the-shelf feature extractors. Once the models get fine-tuned on the Antwerp dataset, these networks, which have also been fine-tuned on the additional source domain of the Rijksmuseum dataset, outperform the architectures that were pre-trained on ImageNet only. This happened to be the case for both target tasks $\mathcal{T}_T$, and for all considered architectures, as reported in Table 5.3. This demonstrates how beneficial it is for the models to have been trained on a similar source task and how this can lead to significant improvements both when the networks are used as feature extractors as when they are fine-tuned.

Table 5.3: The results obtained on the classification experiments performed on the Antwerp dataset with models that have been initially pre-trained on ImageNet (θ_i) and the same architectures which have been fine tuned on the Rijksmuseum dataset (θ_r). Our results show that the latter pre-trained networks yield better results both if used as off the shelf feature extractors and if fine tuned.

$\mathcal{T}_T$	model	θ_i + off the shelf	θ_r + off the shelf	θ_i + fine tuning	θ_r + fine tuning
②	Xception	42.01%	62.92%	69.74%	72.03%
②	InceptionV3	43.90%	57.65%	70.58%	71.88%
②	ResNet50	41.59%	64.95%	76.50%	78.15%
②	VGG19	38.36%	60.10%	70.37%	71.21%
③	Xception	48.52%	54.81%	58.15%	58.47%
③	InceptionV3	21.29%	53.41%	56.68%	57.84%
③	ResNet50	22.39%	31.38%	62.57%	69.01%
③	VGG19	49.90%	53.52%	54.90%	60.01%

5.3.4 *Selective Attention*

The benefits of the fine-tuning approach over the off-the-shelf one are clear from our previous experiments. Nevertheless, we do not have any insights yet about what exactly allows fine-tuned models to outperform the Imagenet only pre-trained architectures. In order to provide an answer to that, we investigate which pixels of each input image contribute the most to the final classification predictions of the networks. We do this by using the "VisualBackProp" algorithm presented by [24], which is able to identify which feature maps of the networks are the most informative ones with respect to their final prediction. Once these feature maps are identified, they get backpropagated to the original input image and visualized as a saliency map according to their weights. The higher the activation of the filters, the brighter the set of pixels covered by these filters are represented.

The results that we have obtained provide interesting insights into how fine-tuned models develop novel selective attention mechanisms over the images, which are very different from those that characterize the ImageNet only pre-trained networks. We report the existence of these mechanisms in Fig. 5.4 where we visualize the different saliency maps between a model pre-trained on ImageNet and the same neural architecture, which has been fine-tuned on the Rijksmuseum collection. In Fig. 5.4 we visualize which sets of pixels allow the fine-tuned model to successfully classify an artist of the Rijksmuseum collection that the same architecture was not able to recognize initially. It is possible to notice that the saliency maps of the latter architecture either correspond to what is more similar to a natural image, as represented by the central image of the first row of plots, or even to what appear to be non-informative pixels at all, as shown by the second image in the second row. However, when considering the fine-tuned model, we clearly observe that these saliency maps change. In this case, the network attends towards the set of pixels representing people at the bottom, suggesting that this allows the model to recognize the artist of the considered artwork appropriately.

These observations can be related to parallel insights in authorship attribution research [53], an established task from Natural Language Processing that is highly similar in nature to artist recognition. In this field, preference is typically given to high-frequency function words (articles, prepositions, particles etc.) over content words (nouns, adjectives, verbs, etc.), because the former are generally considered to be less strongly related to the specific content or topic of a work. As such, function words or stop words lend themselves more easily to attribution across different topics and genres. In art history, strikingly similar views have been expressed by the well-known scholar Giovanni Morelli (1816-1891), who published seminal studies in the field of artist recognition [283]. In Morelli's view too, the attribution

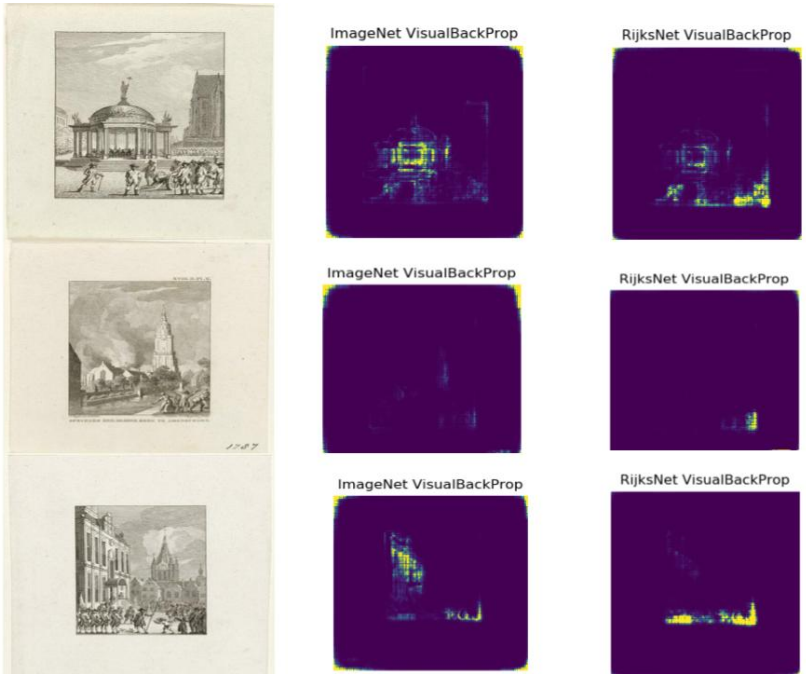


Figure 5.4: A visualization of the saliency maps that are obtained when trying to classify an artist of the Rijksmuseum collection (first row of images) with either an ImageNet pre-trained model that is used as simple feature extractor (second row of images), or with the same kind of model which gets fine-tuned (third row of images). We can observe that after getting fine-tuned the network develops novel selective attention mechanisms that allow it to shift its attention from e.g., the buildings depicted in the paintings to the people represented at the bottom.

of a painting could not happen on the basis of the specific content or composition of a painting, because these items were too strongly influenced by the topic of a painting or the wishes of a patron. Instead, Morelli proposed to base attributions to so-called *Grundformen* or small, seemingly insignificant details that occur frequently in all paintings and typically show clear traces of an artist's individual style, such as ears, hands or feat, a painting's function words, so to speak. The saliency maps above reveal a similar shift in attention when the ImageNet weights are adapted on the Rijksmuseum data: instead of focusing on higher-level content features, the network shifts its attention to lower layers in the network, seemingly focusing on insignificant details, that nevertheless appear crucial to perform artist attribution.

5.4 CONCLUSION

This chapter provides the first insights into the potential that TL can offer for art classification. We have investigated the behavior of convo-

lutional networks which have been pre-trained initially on a very different classification task and shown how their performance can be improved when a fine-tuneing training approach is adopted. Moreover, we have observed that such neural architectures perform better than if they are trained from scratch and that during the fine-tuning stage, they develop new saliency maps that can provide insights about what makes these models outperform the ones that are pre-trained on the ImageNet dataset only. Such saliency maps reflect themselves in the development of new features, which can then be successfully used by the models when classifying heritage objects from different heritage collections. It turns out that the Rijksmuseum fine-tuned models are a better alternative to the same kind of architectures that are pre-trained on ImageNet only and we hope that they will serve the CV community that will deal with similar machine learning problems in the future.

6 NOVEL DATASETS FOR TRANSFER LEARNING

Outline

In this chapter, we continue studying the transfer learning properties of convolutional neural networks trained on non-natural image distributions. To facilitate this process, we present MINERVA, a novel dataset that can be used both for object classification as for object detection. We report thorough experiments that highlight the challenges that can arise from using MINERVA as a computer vision testbed, while at the same time, we further characterize the benefits that can come from adopting transfer learning training strategies. The structure of this chapter is the following: in Sec. 6.1 we describe some of the limitations that currently define the field of computer vision and that have served as inspiration for the development of our newly introduced dataset. In Sec. 6.2 we present MINERVA, we thoroughly explain how its images have been collected and annotated, and how the resulting splits have served for the experiments that are presented in Sec. 6.3. We then report and discuss the results of our experiments in Sec. 6.4 and Sec. 6.5 respectively, before ending the chapter by identifying possible avenues for future work in Sec. 6.6.

This chapter is an extended version of the publication Sabatelli et al. [203].

6.1 CHALLENGES OF MODERN COMPUTER VISION

If it is true that the results presented in the previous chapter show that it is possible to transfer pre-trained convolutional neural networks to non-natural image datasets successfully, it is equally true that some limitations might still need to be addressed. Above all, the need to fine-tune the networks instead of simply using them as feature extractors. As demonstrated by the experiments performed on the third classification problem of the Rijksmuseum dataset, it is clear that pre-trained models might only learn features that are relevant for their respective source task $\mathcal{T}_S$ (ImageNet), which therefore might result in unsatisfying performance when an off-the-shelf training strategy is used. While this is a result that does not come as a surprise, as it would be unreasonable to expect pre-trained networks to act as universal feature extractors, this limitation can still have some important practical implications since it can prevent the deployment of

computer vision systems outside the domain of natural images. As a practical example, let us consider the first image presented in Fig. 6.1 and the computer vision task of object detection. We tackle this task with an object detector that is pre-trained on natural images only and that, therefore, has never seen any images coming from a domain other than the source domain $\mathcal{D}_S$. From the model's performance, it is clear that only one out of the two predictions made by the network is appropriate, as it fails in detecting the musical instrument depicted in the image by wrongly classifying it as a 'frisbee". While certainly reasonable and fully justifiable, this kind of performance is the result of some limitations that currently characterize modern Computer Vision (CV), which we summarize as follows:

- Photorealism and Data Scarcity: it is well known that modern CV strongly gravitates towards photorealistic material since most of the datasets that are used in the field are representative of digitized, or born-digital, versions of photographs. Nevertheless, datasets like MNIST, CIFAR-10/100 and the already mentioned ImageNet play a crucial role in today's rapid development of the field, as they are constantly used as benchmarks by the community. While certainly suitable for defining different challenging CV tasks, it is worth noting that these datasets are also only partially representative of the physical world, as they do not actively attempt to distort the reality they depict. Unfortunately, datasets going beyond the photorealistic domain are either much rarer, or are not as popular as their photorealistic counterparts, a limitation that results into pre-trained models that fail in performing well when used outside from the natural world (see again first image of Fig. 6.1).

- Modern Training Classes: the performance depicted in the first image of Fig. 6.1 can largely be attributed to the fact that the model used for detecting the objects in the image has never been explicitly trained on images of musical instruments. As a result its predictions can only tend to be representative of the classes that have governed the training process. While this behavior has only to be expected, it can still serve as a surrogate for highlighting an important limitation of modern object detection datasets: datasets are not as diversified and heterogeneous as one might expect. As an example let us consider the popular Pascal-Voc [54] and MS-COCO [138] datasets. The first one tackles the detection of 20 classes, out of which more than a third constitute different kinds of transportation systems, such as 'trains", 'boats",'motorcycles" and 'cars". The latter, albeit more complex, mostly represents objects that are representative of the highly technological world we currently live in, with classes such as 'microwave", 'laptop" and 'remote con-

trol". In practice, this results in models that gravitate towards detecting objects in an image that are modern, a behavior that hurts not-technological classes such as the 'person" one, which should, but unfortunately is not, be detected in the second image of Fig. 6.1.

- Model Robustness: the aforementioned limitation also results into models that learn features that are hardly general enough for successfully tackling different representations of the same class. As an example, let us consider the last image of Fig. 6.1: we can see that a model pre-trained on MS-COCO successfully detects the persons represented in the paintings only as long as their pose corresponds to a pose that can easily be found in the images depicting persons in photorealistic datasets. As soon as a person is depicted in a pose different from the one that usually characterizes a person in a photorealistic dataset (sitting or standing), then a pre-trained network mistakenly detects it as an animal.

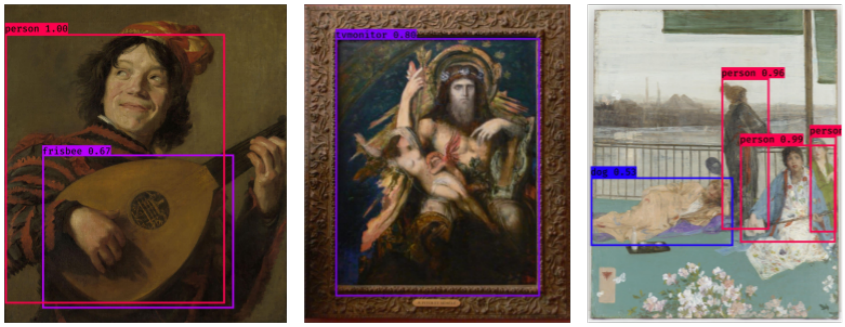


Figure 6.1: Some examples that show the limitations of object detectors that are trained on photorealistic images only. In the first image, we see how a model confidently detects a 'frisbee" for a 'lute", while in the second image, we can observe how next to being unable to detect the people in the painting, it also mistakenly detects the frame as a 'tv-monitor". Similar limitations can be observed in the third image, where we can see that the persons within the painting are only correctly detected as long as they are either sitting or standing.

This chapter takes inspiration from these limitations and uses them as a surrogate for introducing novel datasets that can be used as a benchmark for CV researchers. The purpose of such datasets is twofold: on the one hand, they represent, at least in part, a solution to the aforementioned issues that currently characterize CV, while on the other hand, they allow us to continue studying the transfer learning properties of convolutional neural networks which we started exploring in the previous chapter.

6.2 THE MINERVA DATASET

We now introduce MINERVA, a novel annotated dataset that can be used for object detection. More specifically, the main task that we present is that of the detection of musical instruments in non-photorealistic, unrestricted image collections from the artistic domain. We start by describing how its images have been first collected and then annotated, while we then move on towards quantitatively characterizing the dataset from a machine learning perspective.

6.2.1 *Data Collection*

The images constituting MINERVA come from three different data sources, which allow the dataset to be highly varied and unrestricted. Its images cover a large range of periods, genres, and materials and are both of photorealistic and not-photorealistic nature as visually represented in Fig. 6.2. The three data sources are the following:

- RIDIM: which stays for *Repertoire International d'Iconographie Musicale* is an international digital inventory for musical iconography that functions as a reference image database. Developed and curated by Green and Ferguson [76] it has been designed to facilitate the discovery of music-related artworks. Among the three different considered data sources, the images coming from the RIDIM collection are the ones of the highest quality in terms of resolution.

- RMFAB/RMAH: which stays for *Royal Museums of Fine Arts of Belgium* and *Royal Museums of Art and History*. These images come from a larger pool of digitized images that have been manually selected based on whether they included depictions of musical instruments or not. Among the different data sources, the amount of images coming from RMFAB/RMAH within MINERVA is the lowest compared to the other two data sources. These images are of midrange resolution.

- Flickr: is a well-known image hosting service from which we downloaded a large dataset of images depicting musical instruments in the visual arts pre-dating 1800. Most of the images present within MINERVA come from Flickr, although their resolution is not always on par with the one of the previous two data sources.

Once all these images have been collected we have started the labeling process.

Figure 6.2: Samples from Minerva.

6.2.2 *Annotation Process*

We manually annotated almost 10000 instruments by using the conventional method of rectangular bounding boxes. To this end, we have used the open-source `Cytomine` software [147], a rich web environment that allows highly collaborative analysis of multi-gigapixel imaging data. Initially developed for facilitating the task of image annotation in biomedical informatics, `Cytomine` has already been widely used for the annotation and creation of several datasets [164]. However, it is worth noting that its use within the present study is among the very first ones which use the software outside the context of large-scale bioimaging data. All the individual instruments within MINERVA have been unambiguously identified and labeled by using their MIMO codes. The MIMO (Musical Instrument Museums Online) initiative is an international consortium, well known for its online database of musical instruments, aggregating data and metadata from multiple heritage institutions [47]. An important contribution of Dolan [47] is the development of a uniform metadata documentation standard for the field, including a multilingual vocabulary that can be used for identifying musical instruments in an interoperable manner. We have followed this metadata standard and manually labeled the previously collected images within `Cytomine` as visually represented in Fig. 6.3.

6.2.3 *Versions and Splits*

MINERVA comes in four different, increasingly complex versions: `Minerva-0` which is arguably the easiest version of the dataset where

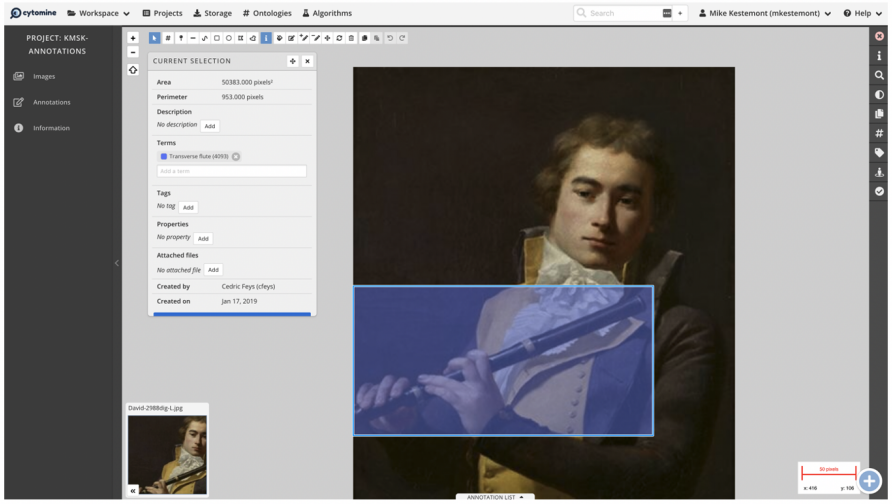


Figure 6.3: A visualization of the annotation process performed with the `Cytomine` platform.

the target task $\mathcal{T}_T$ simply consists in detecting whether a musical instrument is present within an image or not. We, therefore, do not yet consider the task of predicting the class of the detected instrument. The second version of the dataset is `Minerva-Hypernym` where the goal is that of detecting all the images present within `Minerva-0` and classify them according to their hypernym categories. All instruments present within MINERVA correspond to 5 different hypernyms which define them as: 'stringed instruments", 'wind instruments", 'percussion instruments", 'keyboard instruments" and 'electronic instruments'. The last two versions of MINERVA are `Minerva-5` and `Minerva-10` where the goal is to detect and classify the instruments depicted in the images according to the top 5 or top 10 most occurring classes. These classes are: 'Lute", 'Harp", 'Violin", 'Trumpet", 'Shawn", 'Bagpipe", 'Organ", 'Horn", 'Rebec" and 'Lyre". Naturally, in `Minerva-5` we only consider the first 5 of such classes, whereas in `Minerva-10` we consider all 10 of them. Each version of the dataset comes with its training, validation, and testing splits, where we offer the guarantee that at least one of the instrument classes in the task is represented in each of the splits. Additionally, the splits are stratified so that the class distribution is approximately the same in each split. The number of images per split in each version is summarized in Table 6.1 where N_t corresponds to the number of images present within the split, whereas I_t denotes the number of total instruments. The hypernym version of the dataset is not reported as it shares the same images and splits as `Minerva-0` (they both contain all instruments). However, a distribution of the hypernym classes within `Minerva-Hypernym` is reported in Fig. 6.4. All splits have been created with the `scikit-learn` software [179] by using 50% of the images

for training and the remaining 50% for validation and testing (25% respectively).

Table 6.1: An overview reporting how many images N_t and instruments I_t are present within the splits of the `Minerva-0`, `Minerva-5` and `Minerva-10` versions of the MINERVA dataset.

$\mathcal{T}_T$	training-set		validation-set		testing-set	
	N_t	I_t	N_t	I_t	N_t	I_t
`Minerva-0`	1857	4243	1137	2288	1182	2102
`Minerva-5`	952	1589	540	852	721	1173
`Minerva-10`	1227	2147	680	1127	897	1506

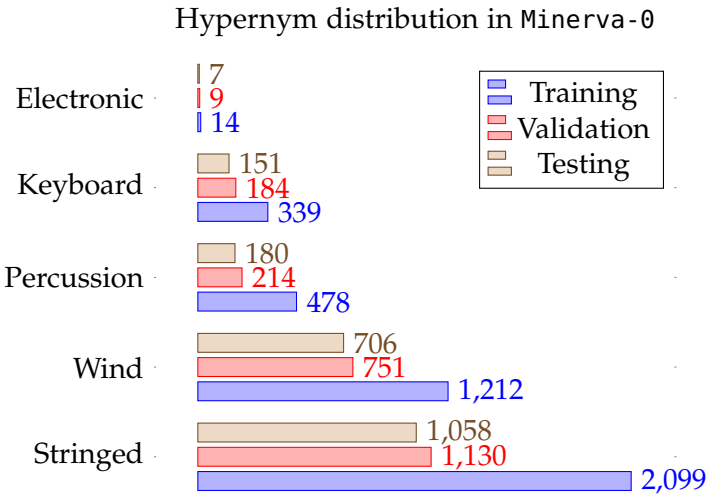


Figure 6.4: A visual representation of the distribution of the hypernym classes that are present within `Minerva-0` and that define the `Minerva-Hypernym` benchmark.

6.3 BENCHMARKING

While MINERVA has been created with the primary intention of serving as a novel dataset for object detection, it can nevertheless still be used to test the classification performance of convolutional neural networks. Although it could be argued that this task could be easier than object detection, it is still of high interest as it provides a novel benchmark for further characterizing the transfer learning properties of neural networks that we started studying in the previous chapter. Therefore, we hereafter report results both for classification experiments as well as for object detection experiments. In the first section, we consider the target task $\mathcal{T}_T$ of classifying the bounding boxes that have been annotated in MINERVA as standalone images, while in the second section, we aim at both detecting and classifying the content

of the potentially detected bounding boxes. We hereafter describe the experimental protocol used for both sets of experiments in detail.

6.3.1 *Classification*

The experimental setup used for the classification experiments largely builds on top of the study that we presented in Chapter 5. We continue to explore whether popular neural architectures, which have obtained state-of-the-art results on the ImageNet benchmark, can perform equally well when trained on datasets of non-natural images. To this end, we again consider the well known VGG19 [228], InceptionV3 [244] and ResNet50 [285] neural architectures. As done in the previous chapter, we keep investigating the effect that different weight initialization strategies have on the final performance of the networks to characterize further the potential benefits that can come from adopting transfer learning. Specifically, we train the three considered neural architectures by following three different initialization strategies: 'random" which simply initializes the model's parameters after following He's weight initialization strategy [89], 'ImageNet" which instead uses the weights that are obtained after training the networks on the ImageNet source task $\mathcal{T}_S$, and 'RijksNet" which are models that are trained both on the ImageNet dataset and on the Rijksmuseum collection, and that were also used for the final experiments reported in the previous chapter. As the benefits of fully fine-tuning the models over using them as off-the-shelf feature extractors were clear from the results obtained in Chapter 5, we now limit our analysis to this transfer-learning approach solely. We train all networks with the Adam optimizer [113] and by using an initial learning rate of 0.001. As already done for the previous study, we again controlled the training process by using early stopping and by interrupting the training regime as soon as the validation loss did not decrease for five epochs in a row. Naturally, all networks minimize the categorical cross-entropy loss function.

6.3.2 *Object Detection*

For this set of experiments we explore the performance of a YOLO object detector [192], a popular neural architecture that has obtained state-of-the-art results on the MS-COCO object detection benchmark. YOLO treats the task of object detection as a standard regression problem by dividing an image into a $S \times S$ grid and by predicting for each grid cell B bounding boxes and C class probabilities. The main assumption behind YOLO is that any of the $S \times S$ cells contains at most the center of one single object, therefore for every image, cell index $i = 1, ..., S \times S$, predicted box $j = 1, ..., B$ and class index $c = 1, ..., C$ we have the following components:

- $\mathbb{1}_i^{\text{obj}}$ which is 1 if there is an object in cell i, and 0 otherwise;
- $\mathbb{1}_{i,j}^{\text{obj}}$ which is 1 if there is an object in cell i and predicted box j that is the most fitting one, whereas is 0 otherwise;
- $p_{i,c}$ which is 1 if there in an object of class c in cell i, and o otherwise;
- x_i, y_i, w_i, h_i which are the coordinates of an annotated bounding box that are defined only if $\mathbb{1}_i^{\text{obj}} = 1$;
- $c_{i,j}$ which is the IoU between the predicted box and the ground truth target.

At training, YOLO computes the value of the $\mathbb{1}_{i,j}^{\text{obj}}$ for each image together with the respective $c_{i,j}$, and then minimizes the following multi-part loss function:

$$\lambda_{\text{coord}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} \mathbb{1}_{i,j}^{\text{obj}} \left((x_i - \hat{x}_{i,j})^2 + (y_i - \hat{y}_{i,j})^2 + (\sqrt{w_i} - \sqrt{\hat{w}_{i,j}})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_{i,j}})^2 \right)$$

$$+ \lambda_{\text{obj}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} \mathbb{1}_{i,j}^{\text{obj}} (c_{i,j} - \hat{c}_{i,j})^2 + \lambda_{\text{noobj}} \sum_{i=1}^{S\times S} \sum_{j=1}^{B} (1 - \mathbb{1}_{i,j}^{\text{obj}}) \hat{c}_{i,j}^2$$

$$+ \lambda_{\text{classes}} \sum_{i=1}^{S\times S} \mathbb{1}_i^{\text{obj}} \sum_{c=1}^{C} (p_{i,c} - \hat{p}_{i,c})^2 \tag{6.1}$$

where $\hat{p}i,c$, $\hat{x}i,j$, $\hat{y}i,j$, $\hat{w}i,j$, $\hat{h}i,j$ and $\hat{c}i,j$ are the predictions of the network.

In our experiments, we use the YOLO-V3 version of the network introduced by Redmon and Farhadi [193] and initialize it with the weights that are obtained after training the network on the MS-COCO dataset. Regarding the stochastic optimization procedure, we use two different optimizers: we train the network with the Adam optimizer for the first 10 epochs, while we then use the RMSprop optimizer for the remaining training epochs, which are again controlled through early stopping. To assess the final performance of the model, we follow an evaluation protocol that is typical for object detection problems in CV [138]. Each detected bounding box is compared to the bounding box, which has been annotated on the `Cytomine` platform. We only consider bounding boxes for which the confidence level is ≥ 0.05, following the protocol established by Everingham et al. [54]. We then compute the "Intersection over Union" (IoU) for measuring how much the detected bounding boxes differ from the ground-truth

ones. To assess whether a prediction can be considered as a true positive or a false positive, we define two increasingly restrictive metrics: first, IoU ≥ 10 and, secondly, IoU ≥ 50. This approach is inspired by the work of Gonthier et al. [72], where the authors report results for both IoU thresholds when assessing the performance of their weakly supervised learning system on the IconArt dataset.

6.4 RESULTS

6.4.1 *Quantitative Analysis*

Classification We start by discussing the results obtained with our classification experiments. The performance of all models is reported in Tables 6.2, 6.3 and 6.4, where we present the accuracy that the networks have obtained on the different MINERVA testing sets, together with their respective F1 scores. To this end for a given class c, a ground truth label y and a model's prediction $\hat{y}$, let us introduce the notions of precision and recall. The first is computed as:

$$\mathrm{P}(c) = p(y = c|\hat{y} = c) = \frac{TP}{TP + FP}, \tag{6.2}$$

while the latter as

$$\mathrm{R}(c) = p(\hat{y} = c|y = c) = \frac{TP}{TP + FN}. \tag{6.3}$$

Both quantities can be used for computing the F-1 score as follows:

$$\mathrm{F1}(c) = 2 \cdot \frac{\mathrm{P}(c) \cdot \mathrm{R}(c)}{\mathrm{P}(c) + \mathrm{R}(c)}. \tag{6.4}$$

Similarly to the results reported in Chapter 5 we again report the best performing architecture in a green cell, while the second-best performing model in a yellow one.

We can start by observing that among all the results presented in the three different tables, the best performing models are either the ones reported in Table 6.3 or the ones presented in Table 6.4. This confirms the results that were presented in the previous chapter: fine-tuning pre-trained models yields significantly better results than training models from scratch, even when the source and the target tasks can be particularly different (as is the case for musical instruments classification). While the results of this study confirm the conclusions that were drawn at the end of Chapter 5, they also provide some additional insights that were not observed before. First, it appears that the best performing architecture is not ResNet50 anymore, but rather the arguably older InceptionV3, a result which seems to suggest that there is no overall best-performing architecture for all target tasks $\mathcal{T}_T$, and that the best architecture is highly problem dependent. Second, and perhaps arguably more surprising, we can also

see that differently from what was observed in the last experiment in Chapter 5, it appears to be more beneficial to transfer models pre-trained on ImageNet only, instead of models that are additionally trained on the Rijksmuseum collection. Indeed, as can be observed by the results presented in Tables 6.3 and 6.4, the latter pre-training strategy outperforms the first one only when the ResNet50 and the InceptionV3 architectures are trained on the `Minerva-5` benchmark. These results can be explained as follows: in Chapter 5 the target task $\mathcal{T}_T$ tackled with a model pre-trained on the Rijksmuseum collection corresponded to the original source task $\mathcal{T}_S$ (classification of 'types" and 'artists"). In these experiments, however, albeit coming from similar domains, the considered source task $\mathcal{T}_S$ and target task $\mathcal{T}_T$ are unrelated, which could work in favor of an arguably more general ImageNet weight initialization.

Table 6.2: Results obtained when classifying the bounding boxes of the three different MINERVA benchmarks with models that do not come as pre-trained on any sort of source task $\mathcal{T}_S$. We can see that their performance is significantly worse than the one that is obtained when the same models come as pre-trained (see Table 6.3 and Table 6.4).

$\mathcal{T}_T$	ResNet50		InceptionV3		VGG19	
	Accuracy (%)	F1	Accuracy (%)	F1	Accuracy (%)	F1
`Minerva-Hypernym`	50.33	13.39	51.80	14.02	50.12	13.12
`Minerva-5`	40.83	21.88	40.49	21.65	41.26	22.01
`Minerva-10`	32.85	0.09	32.18	0.09	19.72	0.03

Table 6.3: Results obtained when classifying the bounding boxes of the three different MINERVA benchmarks after adapting transfer learning and considering the ImageNet dataset as the only source task $\mathcal{T}_S$. We observe that, compared to the results presented in Table 6.2, this approach yields significant benefits, therefore confirming the results presented in Chapter 5.

$\mathcal{T}_T$	ResNet50		InceptionV3		VGG19	
	Accuracy (%)	F1	Accuracy (%)	F1	Accuracy (%)	F1
`Minerva-Hypernym`	76.64	58.56	79.40	60.07	76.54	57.39
`Minerva-5`	60.41	49.10	72.06	68.89	70.43	68.42
`Minerva-10`	55.37	41.65	60.1	45.12	44.22	40.12

Object Detection The results for this set of experiments are reported in terms of average precision as for each class; we report the area under the precision-recall curve that is obtained by setting IoU ≥ 10 and ≥ 50 as explained in Sec. 6.3.2. We start by discussing the performance that is obtained after fine-tuning a pre-trained YOLO-V3 model on the `Minerva-0` benchmark, where, as a reminder, the goal is that of simply detecting a musical instrument within an image with-

Table 6.4: Results obtained when classifying the bounding boxes of the three different MINERVA benchmarks after adapting transfer learning and considering the ImageNet and the Rijksmuseum collection as source domains $\mathcal{D}_S$. Similarly to what was observed in Table 6.3, we can again see that transfer learning yields significant benefits although this weight initialization strategy does mostly not outperform the more common ImageNet one.

$\mathcal{T}_T$	ResNet50		InceptionV3		VGG19	
	Accuracy (%)	F1	Accuracy (%)	F1	Accuracy (%)	F1
`Minerva-Hypernym`	72.26	52.66	75.80	57.03	66.41	40.35
`Minerva-5`	68.71	64.10	73.66	70.29	48.33	33.92
`Minerva-10`	52.85	41.55	55.51	45.77	37.52	15.22

out classifying it. For an IoU ≥ 10 we report an average precision of 35.33%, while for an IoU ≥ 50, a final score of 22.31%. Both scores demonstrate that the model is successfully able to detect the musical instruments within MINERVA and that, on this task, its performance is on par with the one that is reported on more common object detection benchmarks [138]. More specifically, the model detects an instrument 1386 times, out of which when an IoU ≥ 10 is considered, 878 detections correspond to true positives, whereas 508 detections are false positives. Naturally, the model's performance decreases when an IoU ≥ 50 is considered, as the amount of true positive detections decreases to 648 and the number of false positives increases to 738.

We report a similar quantitative analysis for the `Minerva-Hypernym`, `Minerva-5` and `Minerva-10` benchmarks. We do this in Tables 6.5, 6.6 and 6.7 where we present the different average precision scores, and in Figures 6.5, 6.6 and 6.7, where we visualize the true vs false positives detections. We can see that out of these three benchmarks, the `Minerva-Hypernym` one appears to be the most challenging one as it results in the worst-performing models independently from which IoU threshold is considered. We can observe from Fig. 6.5 that the model can detect 'stringed instruments" successfully, whereas its performance in detecting the remaining four hypernyms of the dataset is drastically worse. When it comes to the `Minerva-5` benchmark, we can observe from Table 6.6 that the model can successfully detect three instruments out of the five instruments which constitute this benchmark, namely 'Harps', 'Lutes", and 'Violins". These results are only second to the ones obtained on `Minerva-0`, although the considered target task $\mathcal{T}_T$ is now significantly harder. Similar detections have been obtained after fine-tuning the model on the `Minerva-10` benchmark. Here we can again observe (see Table 6.7 and Fig. 6.7) that the network can only successfully detect the first three most occurring instruments of the dataset, whereas for the 'Horn", 'Bagpipe" 'Rebec" and 'Lyre" classes no detections at all are made.

Table 6.5: Average Precision (%) obtained when fine-tuning a pre-trained YOLO-V3 object detector on the `Minerva-Hypernyms` dataset. We can observe that satisfying results are obtained for both IoU thresholds when it comes to the detection of stringed instruments, whereas detecting the remaining four hypernyms of MINERVA appears to be much more challenging.

	Stringed	Wind	Percussion	Keyboard	Electronic	Mean
AP IoU $\geq$ 10	28.22	4.58	2.55	7.36	0	6.03
AP IoU $\geq$ 50	20.95	2.91	1.84	4.47	0	8.54

Table 6.6: Average Precision (%) obtained on the `Minerva-5` benchmark. We can observe that the fine-tuned model successfully detects 'Harps", 'Lutes" and 'Violins", whereas the detection of 'Shawns" and 'Trumpets" can be improved.

	Harp	Lute	Violin	Shawn	Trumpet	Mean
AP IoU $\geq$ 10	55.60	36.51	12.21	1.75	1.3	21.47
AP IoU $\geq$ 50	46.80	26.93	7.64	1.01	1.07	16.69

Table 6.7: Average Precision (%) obtained on the `Minerva-10` benchmark. Similarly to what was presented in Table 6.6, we can again observe that the model successfully detects the first three most occurring instruments within the dataset, whereas it appears to perform poorly on the remaining instrument classes.

	Harp	Lute	Violin	Shawn	Trumpet	Organ	Rebec	Lyre	Horn	Bagpipe	Mean
AP IoU $\geq$ 10	46.88	33.74	6.73	0.59	1.83	6.1	0	0	0	0	9.58
AP IoU $\geq$ 50	39.81	25.40	4.82	0.59	0.14	6.1	0	0	0	0	7.68

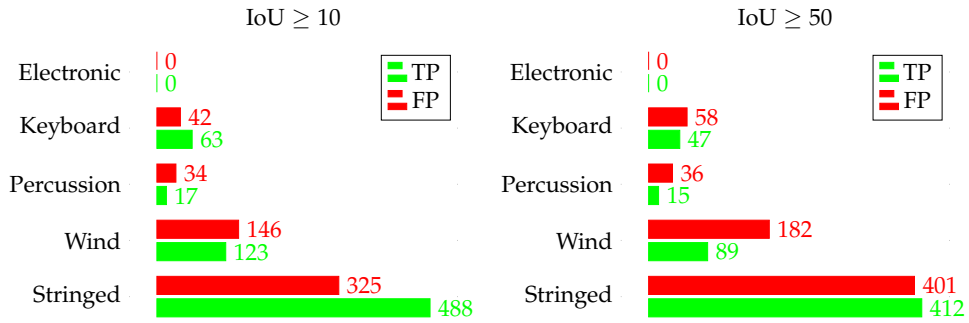


Figure 6.5: True Positive (TP) vs False Positive (FP) analysis on the `Minerva-Hypernym` benchmark for IoU $\geq$ 10 and IoU $\geq$ 50.

6.4.2 *Qualitative Analysis*

We now characterize the performance of the aforementioned fine-tuned models from a qualitative perspective.

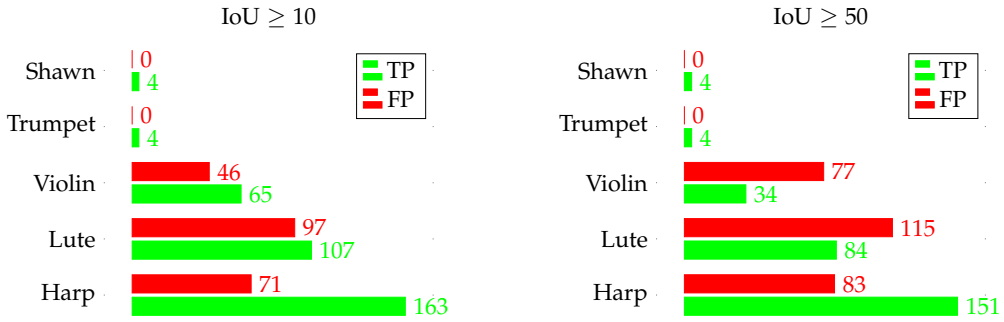


Figure 6.6: True Positive (TP) vs False Positive (FP) analysis on the `Minerva-5` benchmark for IoU $\geq$ 10 and IoU $\geq$ 50.

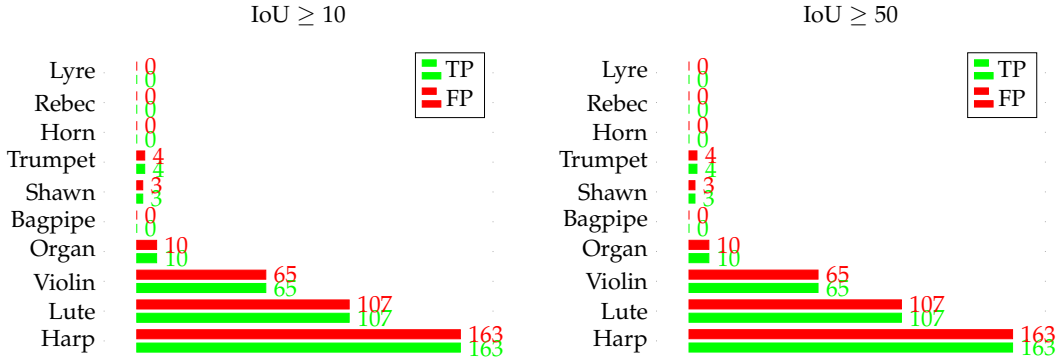


Figure 6.7: True Positive (TP) vs False Positive (FP) analysis on the `Minerva-Hypernym` benchmark for IoU $\geq$ 10 and IoU $\geq$ 50.

Object Classification For the classification experiments, we keep building on top of the study presented in the previous chapter and perform a qualitative evaluation of the models that is based on the visualization of saliency maps, as this allows us to investigate which visual properties in the image are exploited by the networks for correctly classifying the instruments in MINERVA. We hereafter report saliency maps that are obtained after fine-tuning an ImageNet pre-trained ResNet50 model on the `Minerva-Hypernym` benchmark, and that are computed with two different, gradient-based techniques: Grad-CAM [222] and Grad-CAM ++ [36]. Examples of computed saliencies are reported in Fig. 6.8. We can observe that the model focuses on two broad types of regions within the image: properties of the instruments themselves (which can be expected), but also the immediate context of the instruments, and more specifically, the way they are operated, handled, or presented. Let us, for example, consider the 'stringed instruments" category: as can be seen from the images in the first and third row of Fig. 6.8, the network happens to focus more on the strings of the instruments rather than on the, arguably more representative, resonance body of the instrument (which is however of interest in the second row of images). When it comes to the 'percussion instrument" represented in the fourth row and the 'wind in-

strument" represented in the last row, we can again observe that the model considers the fingers handling the instruments at least as important as the instruments themselves.

While saliency maps can produce appealing visual explanations of the performance of neural networks, it is also worth noting that the output of these methods should also be critically assessed. As reported by Alqaraawi et al. [7] saliency maps do not always necessarily explain the model's predictions, and there is a large body of work questioning their reliability [11, 213, 227]. Nevertheless, we also believe that they can still be interesting to visually inspect, as long as the resulting saliencies are taken with a grain of salt.

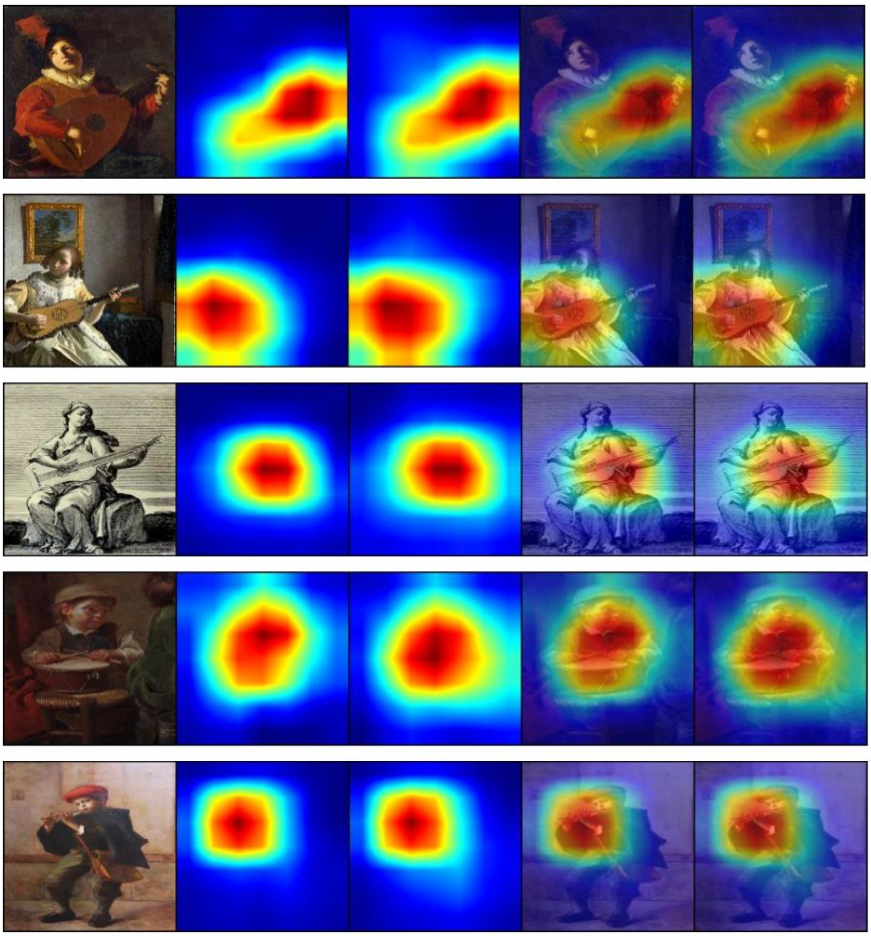


Figure 6.8: Saliency maps obtained after fine-tuning an ImageNet pre-trained ResNet50 on the `Minerva-Hypernym` benchmark. The first image corresponds to the original image, while the second and fourth images, and the third and fifth images, respectively report the performance of the Grad-CAM and Grad-CAM ++ methods.

Object Detection Regarding the models trained for object detection, we visually investigate the quality of the predictions on the IconArt dataset [72], a database of $\approx$ 6000 paintings that have been

collected with the aim of detecting classes that are specific to the analysis of artworks. Among such classes, IconArt tackles the detection of 'angels", 'Jesus" and 'Mary", or more simply 'ruins". IconArt however, does not come with any ground truth labels that are suitable for the task of instruments detection, as the dataset has been built for different purposes. Yet, musical instruments might still be depicted within its images, and trying to detect them corresponds to a nice proof of concept that can show the benefits of deploying MINERVA pre-trained models to different artistic collections. In Fig. 6.9 we show some successful examples of detections that were obtained after testing the performance of a YOLO-V3 model that was fine-tuned on the `Minerva-Hypernym` benchmark. We see that the model can successfully detect musical instruments within this new artistic collection, a result that can be exploited by art historians interested in the study of musical instruments.

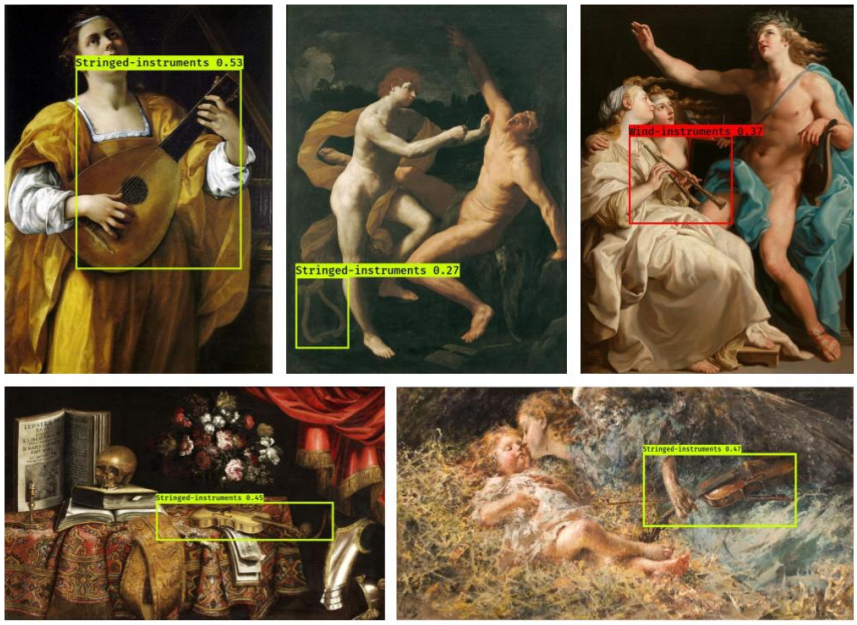


Figure 6.9: Some examples of successful detections that have been obtained on the IconArt dataset with a model fine-tuned on the `Minerva-Hypernym` benchmark.

While these results are undoubtedly nice and encouraging, it is arguably of even more considerable interest analyzing the model's erroneous detections. To this end, we have manually identified the incorrect predictions and grouped them into different categories. This process resulted in novel insights that, at least in part, explain the performance of the models that we have quantitatively assessed in Sec. 6.4.1. First and foremost, we have noticed that the model strongly gravitates towards the detection of 'stringed instruments" (a result which was already observed in Fig. 6.5) and that naturally stems from the fact that, as also presented in Fig. 6.4, stringed instruments are by

far the most occurring type of instruments within MINERVA. However, we also believe that there are two more reasons which drive such erroneous detections: the significant presence of dual, conic contour curves of naked women's bodies, which are reminiscent of the resonance box of guitar-like instruments (see Fig. 6.10), and the presence of book-like objects that, just as instruments, are mostly depicted next to hands and fingers (see Fig. 6.11). We have then also observed that long, often martial objects such as swords, arrows, and spears are mistakenly detected as 'wind instruments". We believe that the reason for this is that the shape between such objects and the one of instruments like 'shawns", is very similar, and sometimes even hard to distinguish for the human eye (see Fig. 6.12). Lastly, we have noticed that musical instruments are often mistakenly detected when regular patterns or parallel grids of straight lines (e.g. folds in clothing or wheel spokes) are present within the images. We hypothesize that the model associates these patterns to the presence of strings (see Fig. 6.13) within stringed instruments.

Figure 6.10: Examples of false detections of 'stringed instruments" within some images representing 'nudity" that are part of the IconArt dataset.

Figure 6.11: Additional examples of false detections of 'stringed instruments" that are triggered by the presence of objects close to hands and fingers.

Figure 6.12: Examples of false detections that are due to the strong resemblance between long martial objects and (mostly) 'wind instruments".

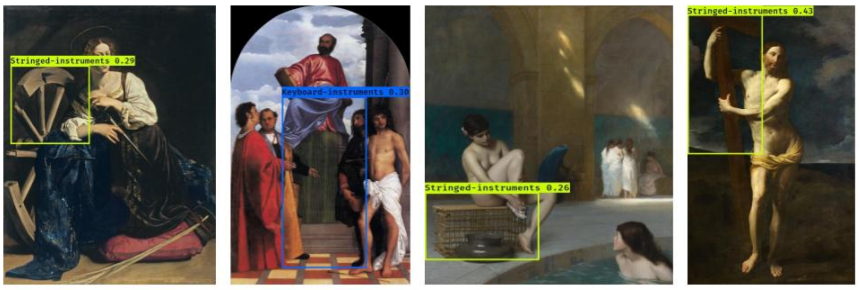


Figure 6.13: Examples of geometrical patterns that mistakenly yield the detection of instruments.

6.5 DISCUSSION AND CRITICAL ANALYSIS

In this chapter, we have introduced MINERVA, the first sizable benchmark dataset for identifying and detecting musical instruments in unrestricted, digitized images from the realm of the visual arts. We hope that this dataset can serve as a novel test-bed for the computer vision community as it provides, at least in part, a solution to some of the challenges that currently define the field (see Sec. 6.1). Our benchmark experiments have highlighted the feasibility of our newly proposed classification and object detection tasks and served us for further characterizing the degree of transferability of pre-trained convolutional neural networks. While, when it comes to the classification experiments presented in the first part of Sec. 6.4.1, the obtained results are lower in terms of accuracy when compared to the classification tasks that we tackled in the previous chapter, they nevertheless provide strong evidence in favor of adapting transfer learning. At the same time, these experiments also show how challenging the simple task of image classification can be, as we believe there is definitively room for improving the results presented in Tables 6.3 and 6.4. Similarly, the results presented in the second part of Sec. 6.4.1 show that it is equally possible to transfer models that have been initially built for the detection of objects in natural images and to use them on non-natural image distributions. We again believe that albeit satis-

fying results on MINERVA have been obtained by starting with an MS-COCO weight initialization, better performance than the one reported in Tables 6.5, 6.6 and 6.7 can be obtained. To this end, we recommend taking into account the qualitative analysis that we presented in Sec. 6.4.2. Overall, our study is a first step towards creating novel, arguably more challenging computer vision test-beds that we hope can be used to further characterize the potential, and limitations, of modern state-of-the-art neural networks. To this end, the methodological protocol that was used for the creation of MINERVA has already inspired the development of new datasets and experimental studies that will be briefly reviewed in the next section.

6.6 FUTURE WORK: TOWARDS MORE BENCHMARKS

The work of Claes [40] has taken inspiration from the process that has led to the development of MINERVA, and has used the `Cytomine` platform for the creation of a novel object detection dataset that tackles the problem of animals detection in paintings. The dataset, coming with $\approx$ 8000 images distributed over 25 different animals classes, has successfully been used for confirming some of the main results that we presented throughout this chapter. More specifically, the good transfer learning properties of object detectors have also been observed when using models that, differently from the aforementioned YOLO architecture, use region proposals and selective search techniques for identifying possible locations of objects of interest within images [69]. Furthermore, this work also shows the potential benefits that could come from using feature extractors that instead of being pre-trained on natural images, as was the case for the YOLO network used throughout this study, are pre-trained on artistic collections instead. Their study, which is in large part inspired by the research that we have performed at the end of the previous chapter also shows that the closer the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$, the better the performance of a transferred model, therefore confirming some of the conclusions that we had drawn for image classification and generalizing them to object detection problems. We report some of the successful detections of animals in artworks obtained by Claes in Fig. 6.14.

To conclude, we hope that MINERVA, together with the dataset created and benchmarked by Claes et al. [40] will push the Computer Vision community towards better identifying the transfer learning properties of convolutional neural networks. Furthermore, we also hope that the `Cytomine` platform, that was so successfully used for creating the aforementioned datasets, will in the future be used for developing novel datasets outside from the digital heritage and digital pathology domains.

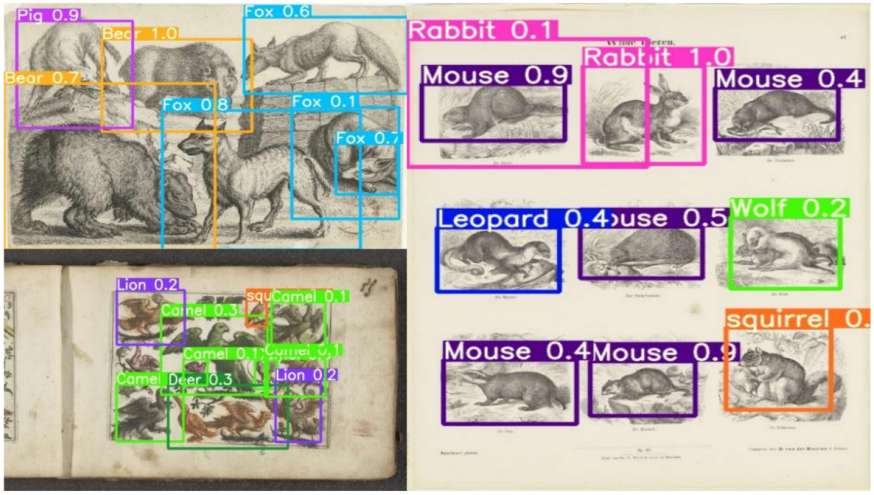


Figure 6.14: Some examples of animal detections within artworks obtained by Claes [40].

7

ON THE TRANSFERABILITY OF LOTTERY WINNERS

Contributions and Outline

In Chapters 5 and 6, we have always performed Transfer Learning (TL) with large and deep convolutional neural networks, as this is the type of models which have obtained state-of-the-art results in the naturalistic domain. While all the studies presented so far aimed at characterizing the TL properties of popular convolutional architectures, we explore a different approach in this chapter. We use TL as a tool for not only exploiting the performance of pre-trained neural networks but also for characterizing a relatively new deep learning phenomenon that comes with the name of the "Lottery Ticket Hypothesis" (LTH). Specifically, we investigate whether lottery winners found on datasets of natural images contain inductive biases that are generic enough to generalize to non-natural image distributions. To do so, we present the first results that study the transferability of winning initializations in this particular training setting. Furthermore, we also show that the LTH offers a novel way for doing TL when the training data is scarce. The rest of this chapter is structured as follows: Sec. 7.1 introduces the LTH and presents the reasons that have motivated studying this phenomenon from a TL perspective. Sec. 7.2 and Sec. 7.3 present the experimental setup that was used throughout this chapter, while Sec. 7.4 and Sec. 7.5 present the main findings of our research. The chapter ends by contextualizing its content with respect to the existing literature in Sec. 7.6 and by identifying some potential avenues for future work in Sec. 7.7.

This chapter is based on the publication Sabatelli, Kestemont, and Geurts [207].

7.1 THE LOTTERY TICKET HYPOTHESIS

The "Lottery-Ticket-Hypothesis" (LTH) [60] states that within large randomly initialized neural networks, there exist smaller sub-networks which, if trained from their initial weights, can perform just as well as the fully trained unpruned network from which they are extracted. This happens to be possible because the weights of these sub-networks seem to be particularly well initialized before training starts, therefore making these smaller architectures suitable for learning (see Fig

7.1 for an illustration). These sub-networks, i.e., the pruned structure and their initial weights, are called winning tickets, as they appear to have won the initialization lottery. Since winning tickets only contain a very limited amount of parameters, they yield faster training, inference, and sometimes even better final performance than their larger over-parametrized counterparts [60, 61]. So far, winning tickets are typically identified by an iterative procedure that cycles through several steps of network training and weight pruning, starting from a randomly initialized unpruned network. While simple and intuitive, the resulting algorithm has, unfortunately, a high computational cost. Even though the resulting sparse networks can be trained efficiently and in isolation from their initial weights, the LTH idea has not yet led to more efficient solutions for training a sparse network than existing pruning algorithms that all also require to first fully train an unpruned network [50, 83, 136, 162, 300].

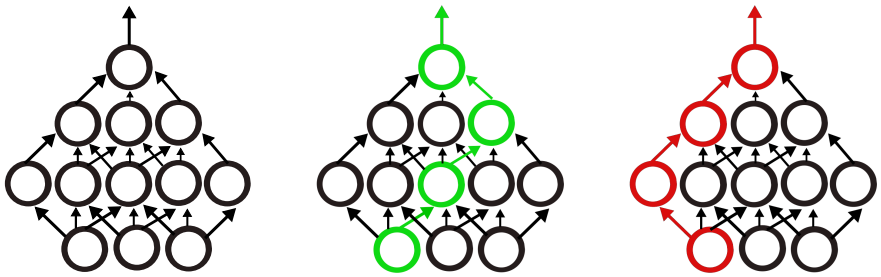


Figure 7.1: A visual representation of the LTH as introduced by Frankle and Carbin [60]. Let us consider a simplified version of a two hidden layer feedforward neural network as is depicted in the first image on the left. The LTH states that within this neural network, there exist multiple smaller networks (represented in green), which can perform just as well as their larger counterpart. Training these sparse models from scratch successfully is only possible as long as their weights are initialized with the same values that were also used when the larger (black) model was initialized. Furthermore, the structure of these sparse models appears to be crucial as well, as it is not possible to randomly extract any subset of weights from an unpruned model and successfully train the resulting sparse network (represented in red in the last figure) from scratch. We visually represent the performance of models that are the winners of the LTH in the two plots reported in Figure 7.2.

Since the introduction of the idea of the LTH, several research works have focused on understanding what makes some weights so special to be the winners of the initialization lottery. Among the different tested approaches, which will be reviewed in Sec. 7.6, one research direction, in particular, has looked into how well winning ticket initializations can be transferred among different training set-

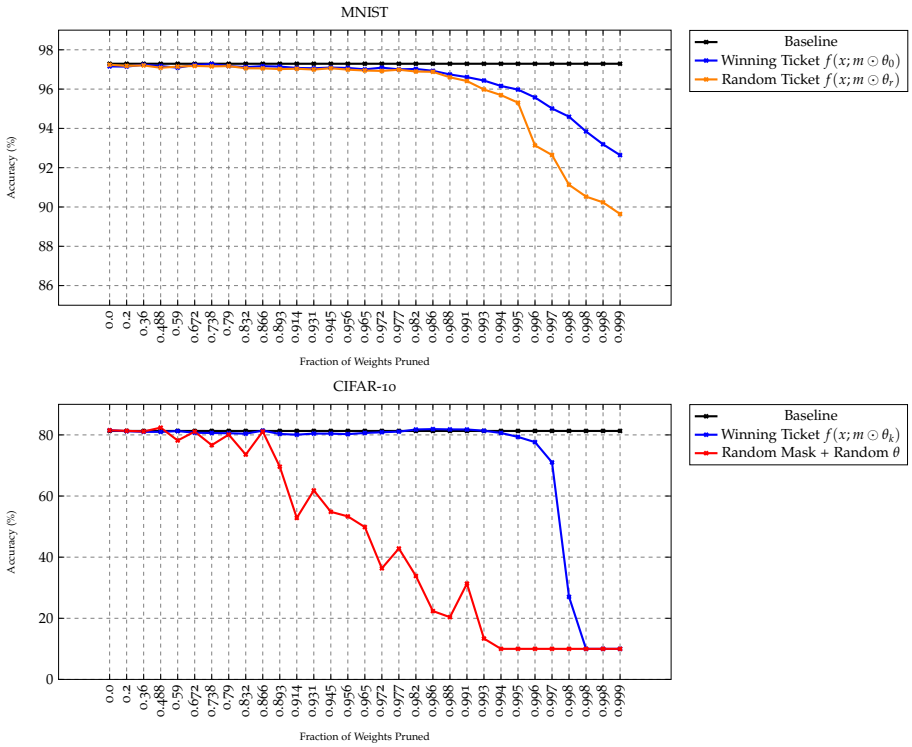


Figure 7.2: A visual representation of the performance of lottery winners that replicate the findings first presented by Frankle and Carbin [60]. In the first plot we consider a multilayer perceptron that gets trained on the MNIST dataset. After the network gets trained from scratch it obtains a final accuracy of $\approx 97\%$ as reported by the black line. We can observe that winning tickets $f(x; m \odot \theta_0)$ only start performing worse than the network they have been extracted from once a large fraction of their weights gets pruned. We can also observe how crucial it is to re-initialize the weights of the pruned models with the same weights that were used when initializing the unpruned model from scratch (θ_0). If random weights are used instead (θ_r), the pruned masks appear to be less robust to pruning (orange curve). In the second plot we show, for a ResNet-50 architecture on the CIFAR-10 dataset, how important it is for a pruned model to come in the form of $f(x; m \odot \theta_k)$, where θ_k are the parameters obtained after k training epochs ($k = 2$ in this plot). Indeed, the red curve shows that it is not possible to simply extract any random subset of weights from a deep convolutional network and obtain a performance that is robust to pruning after randomly initializing the parameters of the model.

tings (datasets and optimizers), an approach that aims at characterizing the winners of the LTH by studying to what extent their inductive biases are generic [163]. The most interesting findings of this study are that winning tickets generalize across datasets, within the natural image domain at least, and that tickets obtained from larger datasets typically generalize better. This opens the door to the transfer of win-

ning tickets between datasets, which makes the high computational cost required to identify them much more acceptable in practice, as this cost has to be paid only once and can be shared across datasets.

In this chapter, we build on top of this latter work. While Morcos et al. [163] focused on the natural image domain, we investigate the possibility of transferring winning tickets obtained from the natural image domain to datasets in non-natural image domains. This question has an important practical interest as datasets in non-natural image domains are typically scarcer than datasets in natural image domains. They would, therefore, potentially benefit more from a successful transfer of sparse networks since the latter can be expected to require less data for training than large over-parametrized networks. Furthermore, besides studying their generalization capabilities, we also focus on another interesting property that characterizes models that win the LTH, and which so far has received less research attention. As originally presented by Frankle and Carbin [60], pruned models, which are the winners of the LTH, can yield a final performance that is better than the one obtained by larger over-parametrized networks. In this chapter we explore whether it is worth seeking such pruned models when training data is scarce, a scenario that as reviewed in Chapter 4 is well known to constraint the training of deep neural networks. To answer these two questions, we carried out experiments on several datasets from two very different non-natural image domains: digital pathology and, similarly to Chapters 5 and 6, digital heritage.

7.2 DATASETS

We consider seven datasets that will serve as target domains $\mathcal{D}_T$, and that come from two different, unrelated sources: histopathology and digital heritage. Each dataset comes with its training, validation, and testing splits. Furthermore, the datasets change in size, resolution, and amount of labels that need to be classified. We report an overview about the size of these datasets in Table 7.1 while a visual representation of the samples constituting these datasets is given in Fig. 7.3. The Digital-Pathology (DP) data comes from the `Cytomine` [147] web application, the same open-source platform that allowed the creation of the MINERVA dataset described in the previous chapter. While `Cytomine` has collected many datasets over the years, in what follows, we have limited our analysis to a subset of four datasets that all represent tissues and cells from either human or animal organs. These datasets, which therefore correspond to the first four target tasks $\mathcal{T}_T$ that will be considered throughout this chapter are: `Human-LBA`, `Lung-Tissues`, and `Mouse-LBA` (which were originally proposed in [164]), and `Bone-Marrow` (which comes from [107]). All four datasets have been used by Mormont, Geurts, and Marée [164], who,

as described in Chapter 4, researched whether neural networks pre-trained on natural images could successfully be re-used in the DP domain. In this chapter, we explore whether an alternative to their transfer-learning approaches could be based on training pruned networks that are the winners of the LTH. This will allow us to investigate the two research questions introduced in Sec. 7.1: we will explore whether winning initializations that are found on datasets of natural images do generalize to non-natural domains and whether sparse models winners of the LTH can perform better than larger unpruned models that get trained from scratch. Regarding the field of Digital-Humanities (DH) we use three novel, small datasets that all revolve around the target task $\mathcal{T}_T$ that is the classification of artworks. We consider two different target tasks that were already studied in Chapter 5, namely type and artist classification. When it comes to the latter target task $\mathcal{T}_T$ we use two different datasets, referred to as `Artist` ① and `Artist` ②, which purpose will be better explained in Sec. 7.5. All images are publicly available as part of the WikiArt gallery [184] and can also be found within the large popular OmniArt dataset [235]. Albeit as we have seen in Chapter 5 in DH it is possible to find large datasets, which cannot be said for the field of histopathology, it is worth mentioning that we have kept the size of these datasets intentionally small in order to fit the research questions introduced in Sec. 7.1.

Table 7.1: A brief overview of the seven different datasets which have been used in this work. As usual throughout this thesis N_t corresponds to the total amount of samples that are present in the dataset, while Q_t represents the number of classes.

Dataset	Training-Set	Validation-Set	Testing-Set	N_t	Q_t
`Human-LBA`	4051	346	1023	5420	9
`Lung-Tissues`	4881	562	888	6331	10
`Mouse-LBA`	1722	716	1846	4284	8
`Bone-Marrow`	522	130	639	1291	8
`Artist` ①	3103	389	389	3881	20
`Type`	2868	360	360	3588	20
`Artist` ②	2827	353	353	3533	19

7.3 EXPERIMENTAL SETUP

We follow an experimental set-up similar to the one that was introduced in [163] (and that has been replicated and validated by Gohil, Narayanan, and Jain [71]). Let us define a neural network $f(x;\theta)$ that gets randomly initialized with parameters $\theta_0 \sim \mathscr{D}_\theta$ and then trained for j iterations over an input space $\mathcal{X}$, and an output space $\mathcal{Y}$. At the

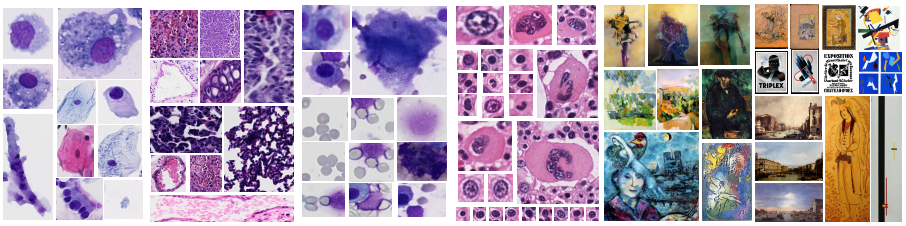


Figure 7.3: Some image samples that constitute the non-natural image datasets which have been used in this work. From left to right we have the `Human-LBA`, `Lung-Tissues`, `Mouse-LBA` and `Bone-Marrow` datasets, while finally we report some examples that represent artworks which come from the field of digital heritage and that are therefore similar to the images we have used for the experiments reported in Chapters 5 and 6.

end of training a percentage of the parameters in θ_j gets pruned, a procedure which results in a mask m. The parameters in θ_j which did not get pruned are then reset to the values they had at θ_k, where k represents an early training epoch. A winning ticket corresponds to the combination between the previously obtained mask, and the parameters θ_k, and is defined as $f(x; m \odot \theta_k)$ [1]. Constructing a winning ticket with parameters θ_k, instead of θ_0, is a procedure which is known as late-resetting [61], and is a simple but effective trick that makes it possible to stably find winning initializations in deep convolutional neural networks [61, 163]. In this study $f(x; \theta)$ comes in the form of a ResNet-50 architecture [83] which gets trained[2] on three popular Computer Vision (CV) natural image datasets serving as source domains $\mathcal{D}_S$: CIFAR-10/100 and Fashion-MNIST (see Fig. 7.4 for a visualization). Following [83, 163], 31 winning tickets $f(x; m \odot \theta_k)$ of increasing sparsity are obtained from each of these three datasets by repeating 31 iterations of network training and magnitude pruning with a pruning rate p of 20%. Specifically, given a tensor $\mathbf{T}$ representing the unpruned parameters in a layer, at each pruning iteration we first train the network for several epochs (using early stopping on the validation set), and then remove all entries in channel c along dimension d defined as:

$$\left\{ \mathbf{T}_{[\ldots,c,\ldots]} : c_{\text{th}} \left(\{ \mathbf{T}_{[\ldots,i,\ldots]} \}_{i=1}^{|d|} \middle| \, || \cdot ||_1 \right) \leq p \right\}, \tag{7.1}$$

where c_{th} is the relative rank of the c^{th} channel of $\mathbf{T}$ along direction d according to the L_1 norm and p ($= 0.2$) is the pruning fraction. This corresponds to the L_1-structured pruning method of [172]. The parameters θ_k that define each of the 31 tickets are then taken as the

1 Note that this formulation generalizes the original version of the LTH [60] that we have represented in Fig. 7.1, where a winning ticket is obtained after resetting the unpruned parameters of the network to the values they had right after initialization, therefore defining a winning ticket as $f(x; m \odot \theta_0)$.

2 All hyperparameters are detailed in Appendix A.

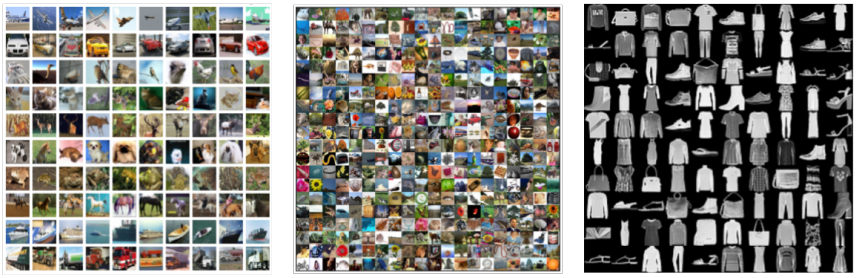


Figure 7.4: The three natural datasets constituting the source tasks $\mathcal{T}_S$ that are necessary for finding winning tickets. From left to right samples from the CIFAR-10/100 and Fashion-MNIST datasets.

weights of the corresponding pruned networks at the kth epoch of the first pruning iteration, with k set to 2 in all our experiments. Once these pruned networks are found, we aim at investigating whether their parameters θ_k contain inductive biases that allow them to generalize to the non-natural image domain. To do so, we replace the final fully connected layer of each winning ticket with a randomly initialized layer that has as many output nodes as there are classes to classify. We then fine-tune each of these networks on the non-natural target tasks $\mathcal{T}_T$ considered in this study. At the end of training, we study the performance of each winning ticket in two different ways. First, we compare the performance of each network to the performance of a fully unpruned network that gets randomly initialized and trained from scratch. Second, we also compare the performance of winning tickets that have been found on a natural image dataset to 31 new sparse models that are the winners of the LTH on the considered target dataset. Since it is not known to which extent pruned networks that contain weights that are the winners of the LTH on a natural image dataset can generalize to target domains $\mathcal{D}_T$ that do not contain natural images, we report the first results that investigate the potential of a novel transfer-learning scheme which has so far only been studied on datasets from the natural image domain. Moreover, testing the performance of sparse networks that contain winning tickets that are specific to a non-natural image target distribution also allows us to investigate whether it is worth pruning large networks with the hope of finding smaller models that might perform better than a large over-parametrized one. As mentioned in Sec. 7.1, pruned networks that are initialized with the winning weights can sometimes perform better than a fully unpruned network. Identifying such sparse networks leads to a very significant reduction of model size, which can be a very effective way of regularization when training data is scarce.

7.4 RESULTS

The results of all our experiments are visually reported in the plots of Fig. 7.5. Each line plot represents the final performance obtained by a pruned model containing a winning ticket initialization on the final testing set of our target datasets. This performance is reported on the y-axis of the plots, while on the x-axis we represent the fraction of weights that are pruned from the original ResNet-50 architecture. As explained in the previous section, the performance of each winning ticket is compared to the performance obtained by an unpruned, over-parametrized architecture reported by the black lines. The models that are the winners of the LTH on a natural image dataset are reported by the green, red and purple lines, while the blue lines report the winners of the LTH on a non-natural target dataset. Furthermore, when it comes to the latter lottery tickets, we also report the performance that is obtained by winning tickets that get randomly reinitialized ($f(x; m \odot \theta_0^{'})$ with $\theta_0^{'} \sim \mathscr{D}_\theta$). The orange lines report these results.

7.4.1 *On the Importance of Finding Winning Initializations*

We can start by observing that pruned models which happen to be the winners of the LTH either on a natural dataset or on a non-natural one can maintain a good final performance until large pruning rates are reached. This is particularly evident on the first two datasets, where models that keep only $\approx$ 1% of their original weights barely suffer from any drop in performance. On the other four datasets, the performance of winning tickets from natural images remain high even for large pruning rates, but winning ticket initializations that are directly found on the considered target dataset start getting harmed once a fraction of $\approx$ 97% of original weights are pruned. Nonetheless, these results show that an extremely large part of the parameters of a ResNet-50 architecture can be considered superfluous, therefore confirming the LTH when datasets contain non-natural images. More importantly, we also observe that pruned models winners of the LTH, significantly outperform larger over-parametrized models that get trained from scratch. This can be very clearly seen in all plots where the performance of pruned models is always consistently better than what is reported by the black line. To get a better sense of how much these pruned networks perform better than their larger unpruned counterparts, we report in Table 7.2 the performance that is obtained by the best performing pruned model, found over all 31 possible pruned models, and compare it to the performance of an unpruned architecture. The exact fraction of weights which is pruned from an original ResNet-50 architecture is reported in Table 7.3 for each configuration. We can observe that no matter which dataset has been

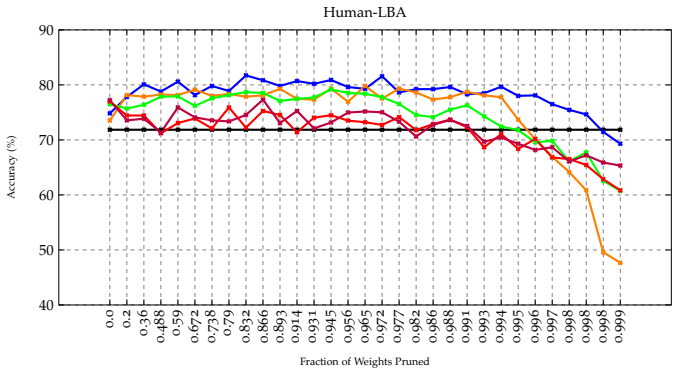
used as source domain $\mathcal{D}_S$ for finding a winning ticket initialization, all pruned networks reach a final accuracy that is significantly higher than the one that is obtained after training an unpruned model from scratch directly on the target task $\mathcal{T}_T$. While in most cases, the difference in terms of performance is of $\approx$ 10% (see e.g., the `Human-LBA`, `Lung-Tissues` and the `Type` datasets), it is worth highlighting that there are other cases in which this difference is even larger. This is the case for the `Mouse-LBA` and `Artist` ① datasets where a winning ticket coming from the CIFAR-10 dataset performs more than 20% better than a model trained from scratch. These results show that to maximize the performance of deep networks, it is always worth finding and training pruned models that are the winners of the LTH.

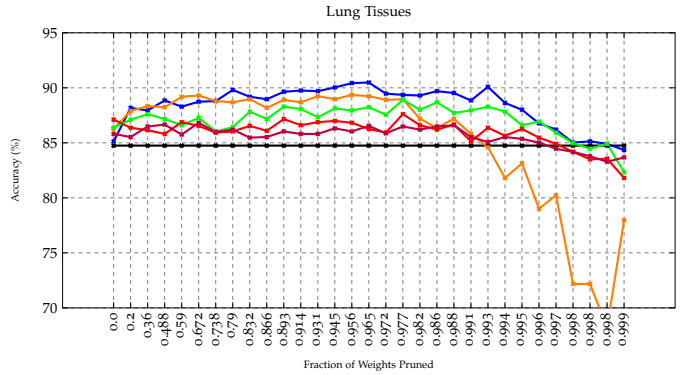
Table 7.2: The results comparing the performance that is obtained on the testing-set by the best pruned model winner of the LTH, and an unpruned architecture trained from scratch. The overall best performing model is reported in a green cell, while the second best one in a yellow cell. We can observe that pruned models winners of the LTH perform significantly better than a larger over-parametrized architecture that gets trained from scratch. As can be seen by the results obtained on the `Mouse-LBA` and `Artist` 1 datasets the difference in terms of performance can be particularly large ($\approx$ 20%). Results averaged over 5 different training runs $\pm$1 std.

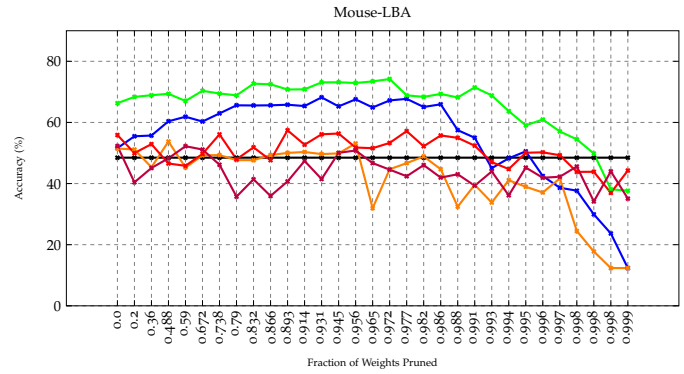
Target-Dataset	Scratch-Training	CIFAR-10	CIFAR-100	Fashion-MNIST	Target-Ticket
`Human-LBA`	$71.85_{\pm 1.12}$	$79.17_{\pm 1.85}$	$76.97_{\pm 0.73}$	$77.32_{\pm 1.85}$	$81.72_{\pm 0.39}$
`Lung-Tissues`	$84.75_{\pm 0.81}$	$88.90_{\pm 1.97}$	$87.61_{\pm 0.90}$	$87.61_{\pm 0.11}$	$90.48_{\pm 0.16}$
`Mouse-LBA`	$48.17_{\pm 1.18}$	$74.20_{\pm 2.04}$	$57.42_{\pm 0.48}$	$52.27_{\pm 1.73}$	$68.20_{\pm 3.79}$
`Bone-Marrow`	$64.66_{\pm 1.36}$	$71.75_{\pm 3.36}$	$69.87_{\pm 0.39}$	$68.77_{\pm 0.39}$	$72.55_{\pm 0.46}$
`Artist` ①	$45.88_{\pm 0.42}$	$66.58_{\pm 1.54}$	$65.55_{\pm 1.79}$	$63.88_{\pm 0.12}$	$58.74_{\pm 1.92}$
`Type`	$41.36_{\pm 2.31}$	$58.63_{\pm 2.97}$	$60.56_{\pm 0.44}$	$58.92_{\pm 0.59}$	$50.44_{\pm 2.23}$

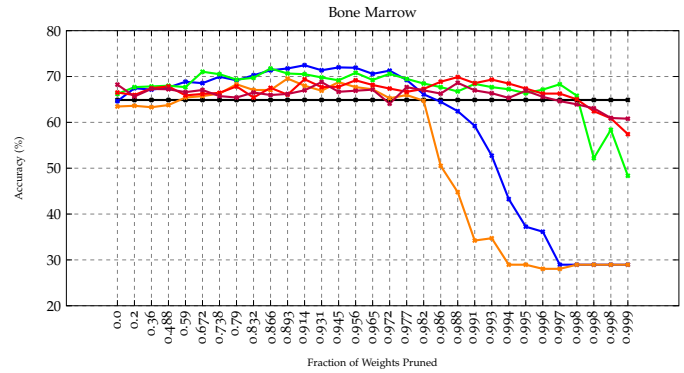
Table 7.3: Some additional information about the lottery winners which performance is reported in Table 7.2. For each winning ticket we report the fraction of weights that is pruned from an original ResNet-50 architecture and that therefore characterizes the level of sparsity of the overall best performing lottery ticket. The results in the Scratch-Training column are not reported as these are unpruned models that are trained from scratch.

Target-Dataset	Scratch-Training	CIFAR-10	CIFAR-100	Fashion-MNIST	Target-Ticket
`Human-LBA`	-	0.945	0.79	0.886	0.832
`Lung-Tissues`	-	0.977	0.977	0.672	0.965
`Mouse-LBA`	-	0.972	0.893	0.738	0.931
`Bone-Marrow`	-	0.866	0.988	0.931	0.914
`Artist` ①	-	0.972	0.993	0.991	0.931
`Type`	-	0.991	0.931	0.995	0.963


Human-LBA
Accuracy (%)
90
80
70
60
50
40
0.0 0.2 0.36 0.488 0.59 0.672 0.738 0.79 0.832 0.866 0.893 0.914 0.931 0.945 0.956 0.965 0.972 0.977 0.982 0.986 0.988 0.991 0.993 0.994 0.995 0.996 0.997 0.998 0.998 0.998 0.999
Fraction of Weights Pruned


Lung Tissues
Accuracy (%)
95
90
85
80
75
70
0.0 0.2 0.36 0.488 0.59 0.672 0.738 0.79 0.832 0.866 0.893 0.914 0.931 0.945 0.956 0.965 0.972 0.977 0.982 0.986 0.988 0.991 0.993 0.994 0.995 0.996 0.997 0.998 0.998 0.998 0.999
Fraction of Weights Pruned


Mouse-LBA
Accuracy (%)
80
60
40
20
0
0.0 0.2 0.36 0.488 0.59 0.672 0.738 0.79 0.832 0.866 0.893 0.914 0.931 0.945 0.956 0.965 0.972 0.977 0.982 0.986 0.988 0.991 0.993 0.994 0.995 0.996 0.997 0.998 0.998 0.998 0.999
Fraction of Weights Pruned


Bone Marrow
Accuracy (%)
80
70
60
50
40
30
20
0.0 0.2 0.36 0.488 0.59 0.672 0.738 0.79 0.832 0.866 0.893 0.914 0.931 0.945 0.956 0.965 0.972 0.977 0.982 0.986 0.988 0.991 0.993 0.994 0.995 0.996 0.997 0.998 0.998 0.998 0.999
Fraction of Weights Pruned

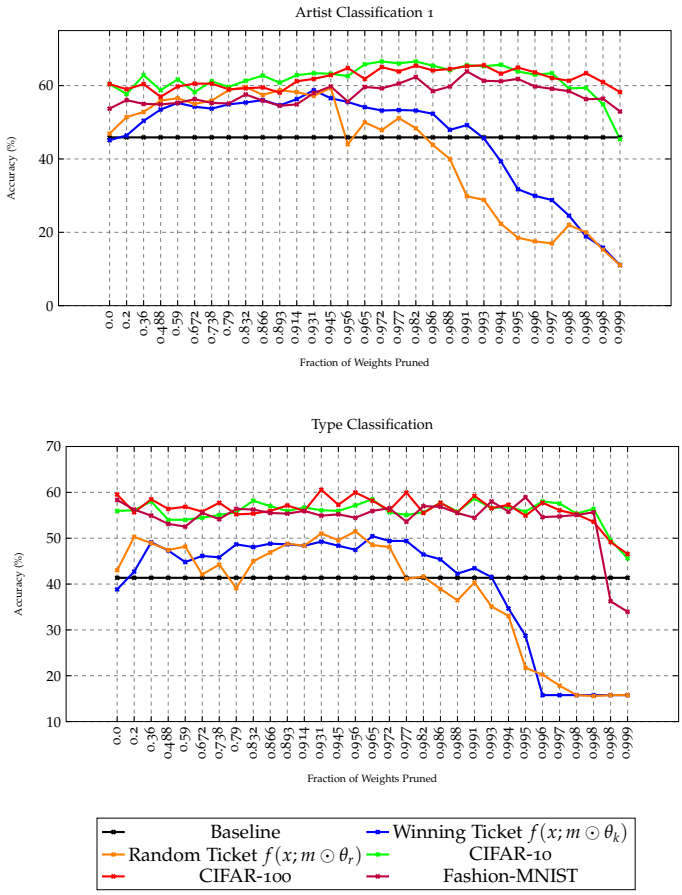


Figure 7.5: An overview of the results showing that sparse models that are the winners of the LTH (represented by the coloured lines) significantly outperform unpruned networks which get randomly initialized and trained from scratch (dashed black line). This happens to be the case on all tested datasets, no matter whether a winning initialization comes from a natural image source or not. It is however worth mentioning that, especially on the biomedical datasets, natural image tickets get outperformed by sparse networks that that are the winners of the LTH on a biomedical dataset. On the other hand this is not the case when it comes to the classification of arts where natural image tickets outperform the ones which are found within artistic collections.

7.4.2 *On the Generalization Properties of Lottery Winners*

We then investigate whether natural image tickets can generalize to the non-natural setting, therefore accounting for the distribution shift between domains $\mathcal{D}$. Findings differ across datasets. When considering the datasets that come from the DP field, we can see that, in three out of four cases, winning tickets that are found on a natural image dataset get outperformed by sparse winning networks that come after training a model on the biomedical dataset. This is particularly evident in the results obtained on the `Human-LBA` and

Lung-Tissues datasets, where the blue line plots consistently reach the highest testing-set accuracy. When it comes to the Bone-Marrow dataset, the difference in terms of performance between the best natural image ticket, in this case coming from the CIFAR-10 dataset, and the one coming from the biomedical dataset, is less evident (see Table 7.2 for the exact accuracies). Furthermore, it is worth highlighting that on the Bone-Marrow dataset, albeit natural image models seem to get outperformed by the ones found on the biomedical dataset, the performance of the latter ones appears to be less stable once substantial pruning rates are reached. When it comes to the Mouse-LBA dataset, these results slightly differ. In fact, this dataset corresponds to the only case where a natural image source ticket outperforms a non-natural one. As can be seen, by the green line plot, pruned models coming from the CIFAR-10 dataset outperform the ones found on the Mouse-LBA dataset.

When focusing our analysis on the classification of arts, we see that the results change significantly from the ones obtained on the biomedical datasets. In this case, all of the natural image lottery winners, no matter the source tasks $\mathcal{T}_S$ they were initially found on, outperform the same kind of models that were found after training a full network on the artistic collection. We can see from Table 7.2 that the final testing performance is similar among all of the best natural image tickets. Similar to what has been noticed on the Bone-Marrow dataset, we can again observe that tickets coming from a non-natural data distribution seem to suffer more from large pruning rates.

These results show both the potential and limitations that natural image winners of the LTH can offer when fine-tuned on non-natural images datasets. The results obtained on the artistic datasets suggest that winning initializations contain inductive biases that are strong enough to get at least successfully transferred to the artistic domain, therefore confirming some of the claims that were made by Morcos et al. [163]. However, it also appears that there are stronger limitations to transferring winning initializations which were not observed by Morcos and colleagues. In fact, our results show that on DP data, the best strategy is to find a winning ticket directly on the biomedical dataset, and that winning initializations found on natural image datasets, albeit outperforming a randomly initialized unpruned network, perform worse than pruned models that are the winners of the LTH on a biomedical dataset.

7.5 ADDITIONAL STUDIES

To characterize the transferability of winning initializations even more while at the same time gaining a deeper understanding of the LTH, we have performed a set of three additional experiments which help us characterize this phenomenon better.

7.5.1 *Lottery Tickets VS fine-tuned pruned models*

So far, we have focused our transfer learning study on lottery tickets that come in the form of $f(x; m \odot \theta_k)$, where, as mentioned in Sec. 7.3, θ_k corresponds to the weights that parametrize a neural network at a very early training iteration. This formalization is, however, different from the transfer learning scenarios that we have described in Chapter 4 and adapted in Chapters 5 and 6, where neural networks get transferred with the weights that are obtained at the end of the training process. We have therefore studied whether there is a difference in terms of performance between transferring and fine-tuning a lottery ticket with parameters θ_k, and the same kind of pruned network which is initialized with the weights that are obtained once the network is fully trained on a source task $\mathcal{T}_S$. We define these kind of models as $f(x; m \odot \theta_i)$ where i stays for the last training iteration. We report some examples of this behaviour in the plots presented in Fig. 7.6, where we consider $f(x; m \odot \theta_i)$ models which were trained on the CIFAR-10 and CIFAR-100 datasets, and then transferred and fine-tuned on the `Human-LBA` dataset. We found that these models overall perform worse than lottery tickets while also being less robust to pruning. This also shows that on this dataset, the slightly inferior performance of the natural image tickets with respect to the target tickets is not due to the weight re-initialization.

7.5.2 *Transferring tickets from similar non-natural domains*

Inspired by the results obtained in Sec. 5.3.3 of Chapter 5 we investigated whether it is beneficial to fine-tune lottery winners that come from a related non-natural image distribution instead of a natural dataset. Specifically we tested whether winning tickets generated on the `Human-LBA` dataset generalize to the `Mouse-LBA` one (since both datasets are representative of the field of Live-Blood-Analysis), and whether lottery winners coming from the `Artist` ① dataset generalized to the `Artist` ② one. We visually represent these results in Fig. 7.7. We observed that winning tickets from a related source are working at least as well as winning tickets obtained from the target dataset. Specifically, `Human-LBA` tickets can perform just as well as winning tickets that are generated on the `Mouse-LBA` dataset, while at the same time also being more robust to large pruning rates. When it comes to lottery winners found on the `Artist` ① dataset we observe that these tickets outperform the ones generated on the `Artist` ② one for all pruning levels. Tickets from a related source are however not necessarily better than tickets from natural image datasets. On `Mouse-LBA`, `Human-LBA` tickets perform better than `CIFAR-100` and `Fashion-MNIST` tickets but they are outperformed by `CIFAR-10` tickets for low pruning levels. On `Artist`②, all natural image tickets perform better than

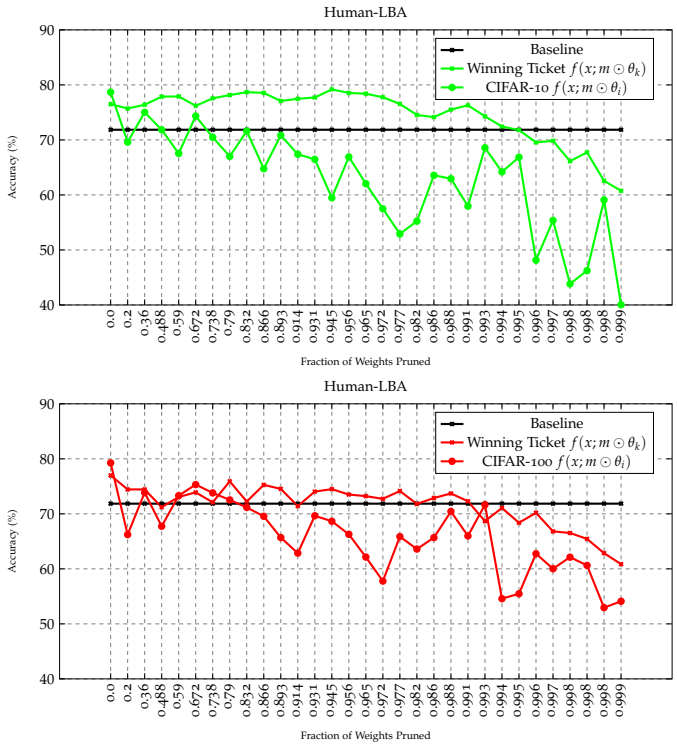


Figure 7.6: Our results showing the advantages of transferring lottery winners over pruned models that are fully fine-tuned on natural datasets (CIFAR-10/100). We can observe that their performance is overall inferior to the one of lottery tickets and that these models are significantly less robust to pruning. We believe that the reason behind their poor performance revolves around the fact that, once completely trained on a specific source task, and after having gone through the pruning stage, these models lose the necessary flexibility that is required for them to adapt to a new task.

the `Artist`① tickets. The better performance of natural image tickets could be explained by the fact that they are extracted from significantly larger datasets than winning tickets derived from the related datasets. Their even stronger performance on the `Artist`① dataset could furthermore be due to the fact that images from the DH domain are closer to natural images than images from the DP domain.

7.5.3 *On the size of the training set*

We have observed from the blue line plots of Fig. 7.5 that there are cases in which lottery winners are very robust to extremely large pruning rates (see as an example the first and second plots), while there are other cases in which their performance deteriorates faster with respect to the fraction of weights that get pruned. The most robust performance is obtained by winning tickets that are generated on the `Human-LBA` and `Lung-Tissues` datasets, which are the two target

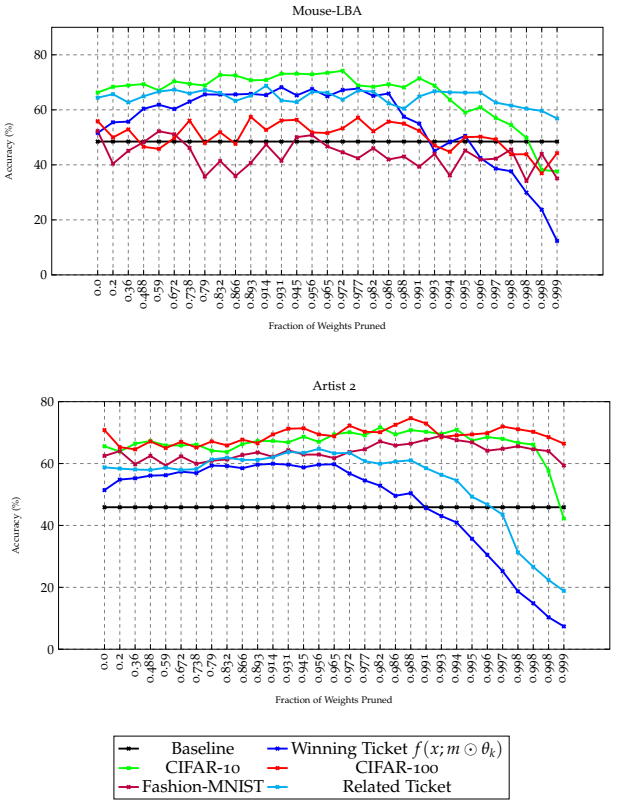


Figure 7.7: Our results showing the benefits of transferring lottery winners that have been identified on a related source task. We can observe that on the `Mouse-LBA` dataset, winning tickets that were obtained on the `Human-LBA` dataset are the ones that are the most robust ones to pruning, while on the `Artist 2` dataset we can observe that lottery winners that have been obtained on the `Artist 1` dataset are both more robust to pruning, while they also yield overall better performance.

datasets that contain the largest amount of training samples. Therefore, we have studied whether there is a relationship between the size of the training data used for finding lottery winners and the robustness in terms of performance of the resulting pruned models. We generated lottery winners after incrementally reducing the size of the training data by 75%, 50% and 25%, and then investigated whether we could observe a similar drop in performance like the one we have observed in the last three blue line-plots of Fig. 7.5 once a large fraction of weights got pruned. Perhaps surprisingly, we have observed that this was not the case, and as can be seen, by the plots represented in Fig. 7.8, the performance of lottery winners that are found when using only 25% of the training set is just as stable as the one of winning tickets which are generated on the entire dataset. However, it is worth mentioning that, albeit the performance of such sparse models is robust, their final performance on the testing set is lower than the

one obtained by winning tickets that have been trained on the full training data distribution.

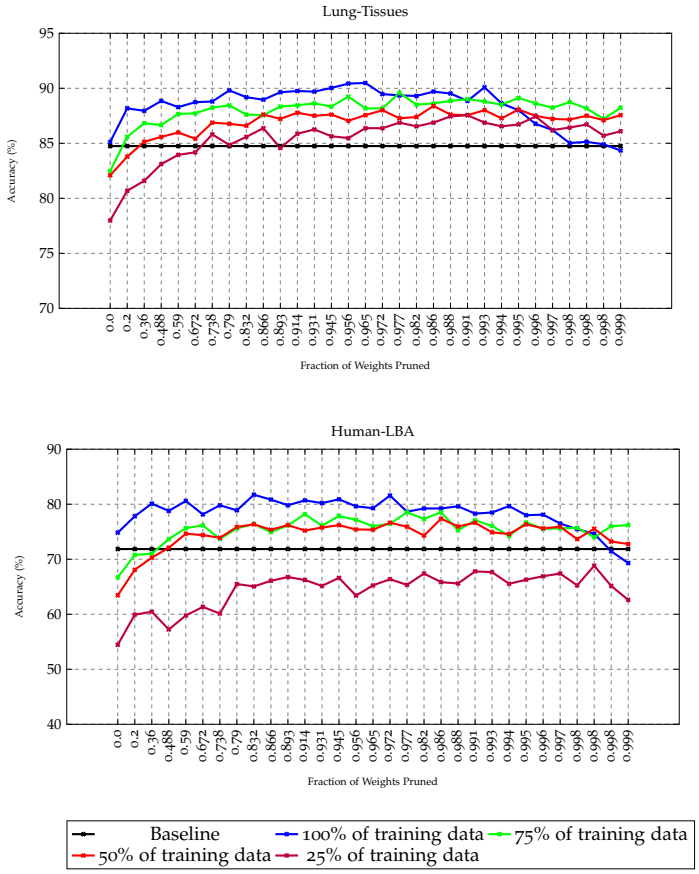


Figure 7.8: Our study showing that the robustness to large pruning rates of lottery winners does not depend from the size of the training dataset. We can observe that even when lottery tickets are trained on only 25% of the dataset their performance remains stable with respect to the fraction of pruned weights. These results suggest that the less stable performance of `Bone-Marrow` lottery tickets observed in Fig. 7.5 does not depend from the small training set.

7.6 RELATED WORK

The research presented in this chapter contributes to a better understanding of the LTH by exploring the generalization and transfer learning properties of lottery tickets. The closest approach to what has been presented in this work is undoubtedly the one presented by Morcos et al. [163], which shows that winning models can generalize across datasets of natural images and across different optimizers. As mentioned in Sec. 7.3, a large part of our experimental setup is based on this work. Besides the work presented in [163], there have been other attempts that aimed to better understand the LTH after studying it from a transfer learning perspective. However, just as the study

presented by Morcos et al. [163], all this research limited its analysis to natural images. Van Soelen and Sheppard [263] transfer winning tickets among different partitions of the CIFAR-10 dataset, while Mehta [152] shows that sparse models can successfully get transferred from the CIFAR-10 dataset to other object recognition tasks. While these results seem to suggest that lottery tickets contain inductive biases which are strong enough to generalize to different domains, it is worth highlighting that their transfer learning properties were only studied after considering the CIFAR-10 dataset as a possible source task $\mathcal{T}_S$ for winning ticket initializations, a limitation which we overcome in this chapter. It is also worth mentioning that the research presented in this part of the dissertation is strongly connected to the work presented by Frankle et al. [61]. While the first paper that introduced the LTH limited its analysis to relatively simple neural architectures, such as multilayer perceptrons and convolutional networks, which were tested on small CV datasets, the presence of winning initializations in larger, more popular convolutional models such as the ones that we used in Chapter 5 trained on large datasets [200] was only first presented in [61]. Since we have used a ResNet-50 architecture [90], we have followed all the recommendations that were introduced by Frankle et al. [61] for successfully identifying the winners of the LTH in larger models. More specifically, we mention the late-resetting procedure, which resets the weights of a pruned model to the weights that are obtained after k training iterations instead of to the values which were used at the beginning of training (as explained in Sec. 7.3), a procedure which has shown to be related to linear mode connectivity [62]. While the work presented in this chapter has limited its analysis to networks that minimize an objective function that is relevant for classification problems, it is worth noting that more recent approaches have identified lottery winners in different training settings. Yu et al. [287] have shown that winning initializations can be found when neural networks are trained on tasks ranging from natural language processing to reinforcement learning, while Sun et al. [236] successfully identify sparse winning models in a multi-task learning scenario. As future work, one could study whether lottery tickets can be found on different neural architectures and also whether they appear when neural networks are trained on CV tasks other than classification. To this end, the YOLO-V3 architecture used in Chapter 6 comes to mind, alongside CV tasks such as object detection and image segmentation.

7.7 CONCLUSION

We have investigated the transfer learning potential of pruned neural networks that are the winners of the LTH from datasets of natural images to datasets containing non-natural images. We have explored

this in training conditions where the size of the training data is relatively small. All of the results presented in this chapter confirm that it is always beneficial to train a sparse model, winner of the LTH, instead of a larger over-parametrized one. Regarding our study on the transferability of winning tickets, we have reported the first results, which study this phenomenon under non-natural data distributions by using datasets coming from the fields of digital pathology and heritage. While for the case of artistic data, it seems that winning tickets from the natural image domain contain inductive biases which are strong enough to generalize to this specific domain, we have also shown that this approach can present stronger limitations when it comes to biomedical data. This probably stems from the fact that DP images are further away from natural images than artistic ones. We have also shown that lottery tickets perform significantly better than fully trained pruned models, that it is beneficial to transfer lottery winners from different but related, non-natural sources, and that the performance of lottery tickets is not dependent on the size of the training data. To conclude, we have provided a better characterization of the LTH while simultaneously showing that when training data is limited, the performance of deep neural networks can get significantly improved by using lottery winners over larger over-parametrized ones.

Takeaway of Part II

This chapter ends the second part of this dissertation where we have studied the transfer learning properties of convolutional neural networks trained for solving supervised learning problems. The results presented throughout this part have consistently shown the benefits that can arise from adopting transfer learning training strategies, as over all its three chapters we have seen that convolutional networks can be generic and powerful feature extractors. We have also seen that their final performance can usually be further improved if a fine-tuning training approach is adopted, and how studying their behavior under the lens of transfer learning allows to gain deeper insights when such networks are combined with the phenomenon of the Lottery Ticket Hypothesis. We now move on towards studying the transfer learning properties of convolutional networks that get trained for solving reinforcement learning tasks, with the aim of investigating whether this family of models is as transferable within this machine learning paradigm as it is in the supervised learning one.

Part III

TRANSFER LEARNING FOR DEEP REINFORCEMENT LEARNING

8

THE DEEP QUALITY-VALUE LEARNING FAMILY OF ALGORITHMS

Contributions and Outline

In the second part of this thesis we have thoroughly studied the level of transferability of deep neural networks that get trained in a supervised learning fashion. From the results of our studies we concluded that significant benefits can come from using pre-trained models over networks that get trained from scratch, and that transfer learning can be a valuable machine learning paradigm for studying the generalization properties of neural networks. In this third, last part of this dissertation we will study whether adopting transfer learning strategies can be as useful in a Deep Reinforcement Learning (DRL) context, where convolutional neural networks get trained for solving optimal control problems. Before studying such transfer learning properties, however, we will start by contributing to the DRL literature by introducing a novel family of DRL algorithms. Therefore, this chapter does not study DRL algorithms from a transfer learning perspective yet, but rather introduces some novel techniques whose transfer learning properties will be researched in the next chapter. The structure of this chapter is the following: in Sec. 8.1 we remind the reader with some background information about the field of DRL and recall the mathematical notation that will be used throughout this chapter. In Sec. 8.2 we introduce the main algorithmic contributions of our research: a novel family of DRL algorithms the performance of which is thoroughly studied from different perspectives in Sec. 8.3. The chapter ends with Sec. 8.4 and Sec. 8.5 where we provide a set of additional studies that characterize the performance of our newly introduced algorithms further, and critically discuss their properties.

This chapter combines the work presented in the following publications: Sabatelli et al. [210], Sabatelli et al. [208], and Sabatelli et al. [209]

8.1 MOTIVATION

In Chapter 3 we have seen that the aim of value-based Reinforcement Learning (RL) is to construct algorithms which learn value functions that are either able to estimate how good or bad it is for an agent to be in a particular state, or how good it is for an agent to perform a particular action in a given state. Such functions are respectively denoted

as the state-value function $V(s)$, and the state-action value function $Q(s,a)$ [239]. We have then seen that in Deep Reinforcement Learning (DRL) the aim is to approximate these value functions with e.g., deep convolutional neural networks [128], as these kind of networks can serve as universal function approximators as well as powerful feature extractors. Classic model-free RL algorithms like Q-Learning [271], Double Q-Learning [259] and SARSA [199] have all led to the development of a "deep" version of themselves in which the original RL update rules are expressed as objective functions that can be minimized by gradient descent [161, 261, 290]. Despite their successful applications [134], however, the aforementioned algorithms only aim at approximating the Q function, while completely ignoring the V function, which is an approach that is prone to issues that go back to standard RL literature. As shown by Van Hasselt, Guez, and Silver [261] the DQN algorithm [161] is known to overestimate the values of the Q function and requires an additional target network to not diverge (which role, as shown by Achiam, Knight, and Abbeel [2], is not yet fully understood). These overestimations can partially be corrected by the DDQN [261] algorithm (see Sec. 3.7 of Chapter 3), which, despite yielding stability improvements, does not always prevent its Q networks from diverging [260] and sometimes even underestimating the Q function. Furthermore, DRL algorithms are also extremely slow to train. In what follows, we introduce a new family of DRL algorithms based on the key idea of simultaneously learning the V function alongside the Q function with two separate neural networks. Our main insight is that by jointly approximating the V function and the Q function, the task of learning one of these value functions can be sped up if the model that is responsible for learning it, can rely on what is being learned by the model responsible for learning the other value function. We show that this simple, yet effective idea yields faster, more robust and better model-free Deep Reinforcement Learning.

8.2 A NOVEL FAMILY OF DEEP REINFORCEMENT LEARNING ALGORITHMS

Just as much as DQN and DDQN are based on two tabular RL algorithms, so is the new family of algorithms presented in this chapter. More specifically we extend two RL algorithms which were first introduced by Wiering [276] and then extended by Wiering and Van Hasselt [277] to the use of deep neural networks that serve as function approximators. Training these algorithms robustly is done by taking advantage of some of the techniques which have been reviewed in Chapter 3.

8.2.1 *DQV-Learning*

Our first contribution is the Deep Quality-Value (DQV) Learning algorithm, a novel DRL algorithm which aims at jointly approximating the V function alongside the Q function in an on-policy learning setting. This algorithm is based on the QV(λ) algorithm [276], a tabular RL algorithm which was reviewed in Chapter 3 and that learns the V function via the simplest form of TD-Learning [237]. The estimates that are learned by this value function are then used to update the Q function in a Q-Learning resembling way. Specifically, after a RL transition $\langle\, s_t, a_t, r_t, s_{t+1} \,\rangle$, QV($\lambda$) uses the TD($\lambda$) learning rule [237] to update the V function for all states:

$$V(s) := V(s) + \alpha\big[r_t + \gamma V(s_{t+1}) - V(s_t)\big]e_t(s), \tag{8.1}$$

where α stands for the learning rate and γ is the discount factor, while $e_t(s)$ are the elibility traces [66, 180, 278] that are necessary for keeping track if a particular state has occurred before a certain time-step or not. These are updated for all states as follows:

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \eta_t(s), \tag{8.2}$$

where $\eta_t(s)$ is an indicator function that returns a value of 1 whether a particular state occurred at time t and 0 otherwise. Before updating the V function, QV(λ) updates the Q function first, and does this via the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\big[r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)\big]. \tag{8.3}$$

We take inspiration from this specific learning dynamic and aim at learning an approximation of both the V function, and the Q function, with two neural networks that are respectively parametrized by Φ and θ. To do so, we follow the same principles which have led to the development of the DQN algorithm. Therefore, starting from Eq. 8.1, and after removing $e_t(s)$ for simplicity, we get the following objective function which is used by DQV for learning the state-value function:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\Big[\big(r_t + \gamma V(s_{t+1}; \Phi^-) - V(s_t; \Phi)\big)^2\Big], \tag{8.4}$$

while the following loss is minimized for learning the Q function when starting from Eq. 8.3:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\Big[\big(r_t + \gamma V(s_{t+1}; \Phi^-) - Q(s_t, a_t; \theta)\big)^2\Big], \tag{8.5}$$

where D is the Experience-Replay memory buffer, used for uniformly sampling batches of RL trajectories $\langle s_t, a_t, r_t, s_{t+1}\rangle$, and Φ^- is the target-network used for the construction of the TD-errors. Note that the role

of this target network is different from its role within the DQN algorithm reviewed in Chapter 3. In DQV, this network corresponds to a copy of the network which approximates the state-value function and not the state-action value function. It is also worth noting that both networks learn from the same TD-target which comes in the following form:

$$y_t^{DQV} = r_t + \gamma V(s_{t+1}; \Phi^-). \tag{8.6}$$

8.2.2 *DQV-Learning with Multilayer Perceptrons*

We start by exploring whether this learning dynamic of jointly approximating two value functions simultaneously, and let the Q function bootstrap from the TD-targets that are learned from the V network, can yield successful results on a set of preliminary experiments. To do so, we use two classic control problems that are well known in the RL literature: `Acrobot` [238] and `Cartpole` [14] with both environments being provided by the Open-AI Gym package [28]. We approximate the V function and the Q function with a two hidden layer Multilayer Perceptron (MLP) that is activated by a ReLU non linearity ($f(x) = max(0, x)$) and compare the performance of DQV to the one of the DQN and the DDQN algorithms, which use the same MLP but for approximating the Q function only. Given the simplicity of these two control problems we did not integrate DQV with the target network Φ^- yet. Our preliminary results reported in Fig. 8.1, show the benefits that can come from training two separate networks with the update rules reported in Eq. 8.4 and Eq. 8.5. We can in fact observe that on both control problems DQV-Learning outperforms DQN and DDQN, by converging significantly faster.

8.2.3 *DQV-Max Learning*

Based on the successful results presented in Fig. 8.1 that highlight the potential benefits that could come from jointly approximating two value functions over one, we now introduce the Deep Quality-Value-Max (DQV-Max) algorithm, a novel DRL algorithm which builds on top of some of the ideas that characterize DQV. Similarly as done for DQV, we still aim at jointly learning an approximation of the V function and the Q function, but in this case, the goal is to do this with an off-policy learning scheme. To construct this algorithm we take inspiration from the QV-Max RL algorithm introduced by Wiering and Van Hasselt [277]. The key component of QV-Max is the use of the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ operator, which makes RL algorithms learn off-policy. We use this operator when approximating the V function and for computing TD-errors which correspond to the ones that are also used by the DQN algorithm. However, within DQV-Max, these TD-

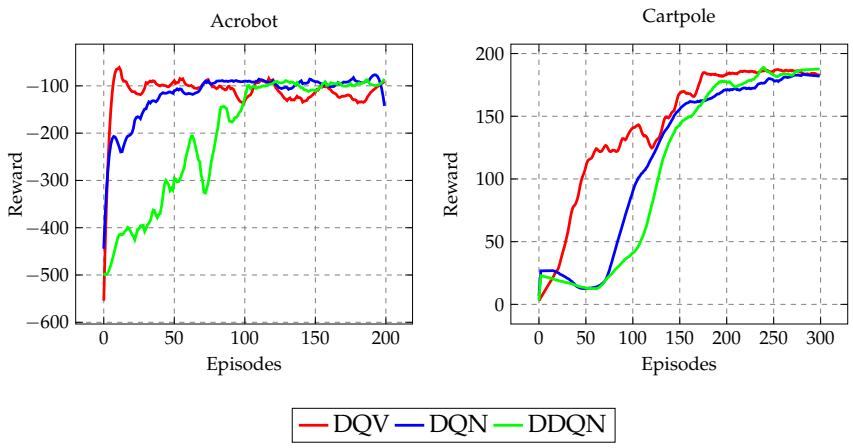


Figure 8.1: Our preliminary results that show the benefits in terms of convergence time that can come from jointly approximating the V function alongside the Q function. We can observe that the DQV-Learning algorithm yields faster convergence when compared to popular algorithms which only approximate the Q function: DQN and DDQN.

errors are used by the state-value network and not by the state-action value network. This results in the following loss which is used for learning the V function:

$$L(\Phi) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \left[\left(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - V(s_t; \Phi) \right)^2 \right]. \tag{8.7}$$

In this case the target network θ^- corresponds to the same target network that is also used by DQN. The TD-error $r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-)$ is however only used for learning the V function. When it comes to the Q function we use the same update rule that is presented in Eq. 8.5 with the only difference being that in this case no Φ^- target network is used. Despite requiring the computation of two different targets for learning, we noticed that DQV-Max did not benefit from using two distinct target networks, therefore its loss function for approximating the Q function is simply:

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \left[\left(r_t + \gamma V(s_{t+1}; \Phi) - Q(s_t, a_t; \theta) \right)^2 \right]. \tag{8.8}$$

The pseudocode of both DQV and DQV-Max is presented at the end of this thesis in Algorithm 1 which can be found in Appendix B. The pseudocode is an adaptation of a standard DRL training loop which corresponds to what is usually presented within the literature [161]. We just make explicit use of the hyperparameters `total_a` and `c` which ensure that enough actions have been performed by the agent

before updating the weights of the target network. We also ensure via the hyperparameter `total_e`, that enough episodes are stored within the memory buffer (which has capacity $\mathcal{N}$) before starting to optimize the neural networks.

8.3 RESULTS

8.3.1 *Global Evaluation*

We evaluate the performance of DQV and DQV-Max on a subset of 15 games coming from the popular `Atari-2600` benchmark [17]. Our newly introduced algorithms are compared against DQN and DDQN. To keep all the comparisons as fair as possible we follow the same experimental setup and evaluation protocol which was used in [161] and [261]. The only difference between DQV and DQV-Max, and DQN and DDQN is the exploration schedule which is used. Differently from the latter two algorithms, which use an epsilon-greedy strategy which has an ϵ starting value of 1.0, DQV and DQV-Max's exploration policy starts with an initial ϵ value of 0.5. All other hyperparameters, ranging from the size of the Experience-Replay memory buffer to the architectures of the neural networks, are kept the same among all algorithms. We refer the reader to the original DQN paper [161] for an in-depth overview of all these hyperparameters. The performance of the algorithms is tested based on the popular `no-op action` evaluation regime. At the end of the training, the learned policies are tested over a series of episodes for a total amount of 5 minutes of emulator time. All testing episodes start by executing a set of partially random actions to test the level of generalization of the learned policies. We present our results in Table 8.1 where the best performing algorithm is reported in a green cell while the second-best performing algorithm is reported in a yellow cell. As is common within the DRL literature, the table also reports the scores which would be obtained by an expert human player and by a random policy. When the scores over games are equivalent, we report in the green and yellow cells the fastest and second fastest algorithm with respect to its convergence time (determined by a visual inspection of the learning curves).

We can start by observing that DQV and DQV-Max successfully master all the environments on which they have been tested, with the only exception being the `Montezuma's Revenge` game. It is well-known that this game requires more sophisticated exploration strategies than the epsilon-greedy one [57], and was also not mastered by DQN and DDQN when these algorithms were introduced. We can also observe that there is no algorithm which performs best on all the tested environments even though, as highlighted by the green and yellow cells, the algorithms of the DQV-family seem to generally perform better than DQN and DDQN, with DQV-Max being the

overall best performing algorithm in our set of experiments. When either DQV or DQV-Max are not the best performing algorithm (see for example the `Boxing` and `Crazy Climber` environments), we can still observe that our algorithms managed to converge to a policy which is not significantly worst than the one learned by DQN and DDQN. There is however one exception being the `Road Runner` environment. In fact, in this game, DDQN significantly outperforms DQV and DQV-Max. It is also worth noting the results on the `Bank Heist` and `Enduro` environments. Both DQN and DDQN failed to achieve super-human performance on these games, while DQV and DQV-Max successfully managed to obtain a significantly higher score than the one obtained by a professional human player. On the `Bank Heist` environment DQV and DQV-Max obtain ≈ 400 points more than an expert human player, while on the `Enduro` environment their performance is almost three times better than the one obtained by DQN and DDQN.

Table 8.1: The results obtained by DQV and DQV-Max on a subset of 15 `Atari` games, compared with those obtained by DQN and DDQN (reproduced from their corresponding publications). We can see that our newly introduced algorithms have a comparable, and often even better performance than DQN and DDQN. As highlighted by the green cells the overall best performing algorithm in our set of experiments is DQV-Max while the second-best performing algorithm is DQV (as reported by the yellow cells). Specific attention should be given to the games `BankHeist` and `Enduro` where DQV and DQV-Max are the only algorithms which can master the game with a final super-human performance.

Environment	Random	Human	DQN [161]	DDQN [261]	DQV	DQV-Max
`Asteroids`	719.10	13156.70	1629.33	930.60	1445.40	1846.08
`Bank Heist`	14.20	734.40	429.67	728.30	1236.50	1118.28
`Boxing`	0.10	4.30	71.83	81.70	78.66	80.15
`Crazy Climber`	10780.50	35410.50	114103.33	101874.00	108600.00	1000131.00
`Enduro`	0.00	309.60	301.77	319.50	829.33	875.64
`Fishing Derby`	-91.70	5.50	-0.80	20.30	1.12	20.42
`Frostbite`	65.20	4334.70	328.33	241.50	271.86	281.36
`Gopher`	257.60	2321.00	8520.00	8215.40	8230.30	7940.00
`Ice Hockey`	-11.20	0.90	-1.60	-2.40	-1.88	-1.12
`James Bond`	29.00	406.70	576.67	438.00	372.41	440.80
`Montezuma's Revenge`	0.00	4366.70	0.00	0.00	0.00	0.00
`Ms.Pacman`	307.30	15693.40	2311.00	3210.00	3590.00	3390.00
`Pong`	-20.70	9.30	18.90	21.00	21.00	21.00
`Road Runner`	11.50	7845.00	18256.67	48377.00	39290.00	20700.00
`Zaxxon`	32.50	9173.30	4976.67	10182.00	10950.00	8487.00

8.3.2 *Convergence Time*

While DRL algorithms have certainly obtained impressive results on the `Atari-2600` benchmark, it is also true that the amount of training

time which is required by these algorithms can be very long. Over the years, several techniques reviewed in Sec. 3.7 of Chapter 3, ranging from Prioritized Experience Replay (PER) [270] to the Rainbow extensions introduced by Hessel et al. [95], have been proposed to reduce the training time of DRL algorithms. It is therefore natural to investigate whether jointly approximating the V function alongside the Q function can lead to significant benefits in this behalf. Unlike the Q function

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \middle| s_t = s, a_t = a, \pi\right],$$

recall that the state-value function

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \middle| s_t = s, \pi\right],$$

is not conditioned on the set of possible actions that the agent may take, and therefore requires fewer parameters to converge. Since DQV and DQV-Max use the estimates of the V network to train the Q function, it is possible that the Q function could directly benefit from these estimates and as a result converge faster than when regressed towards itself (as happens in DQN).

We use two self-implemented versions of DQN and DDQN for comparing the convergence time that is required during training by all the tested algorithms on three increasingly complex `Atari` games: `Boxing`, `Pong` and `Enduro`. Our results, reported in Fig. 8.2, show that DQV and DQV-Max converge significantly faster than DQN and DDQN, therefore confirming the preliminary results which we reported in Fig. 8.1, and highlighting once again the benefits of jointly approximating two value functions instead of one when it comes to the overall convergence time that is required by the algorithms. Even though, as presented in Table 8.1, DQV and DQV-Max do not always significantly outperform DQN and DDQN in terms of the final cumulative reward which is obtained, it is worth noting that these algorithms require significantly less training episodes to converge on all tested games. This benefit makes our two novel algorithms faster alternatives within model-free DRL.

8.3.3 *Quality of the Learned Value Functions*

It is well-known that the combination of RL algorithms with function approximators can yield DRL algorithms that diverge. The popular Q-Learning algorithm is known to result in unstable learning both if linear [256] and non-linear functions are used when approximating the Q function [260]. As seen at the end of Chapter 3, this divergence according to Sutton and Barto [239] is caused by the interplay of three elements that are known as the *'Deadly Triad'* of DRL. The elements of this triad are:

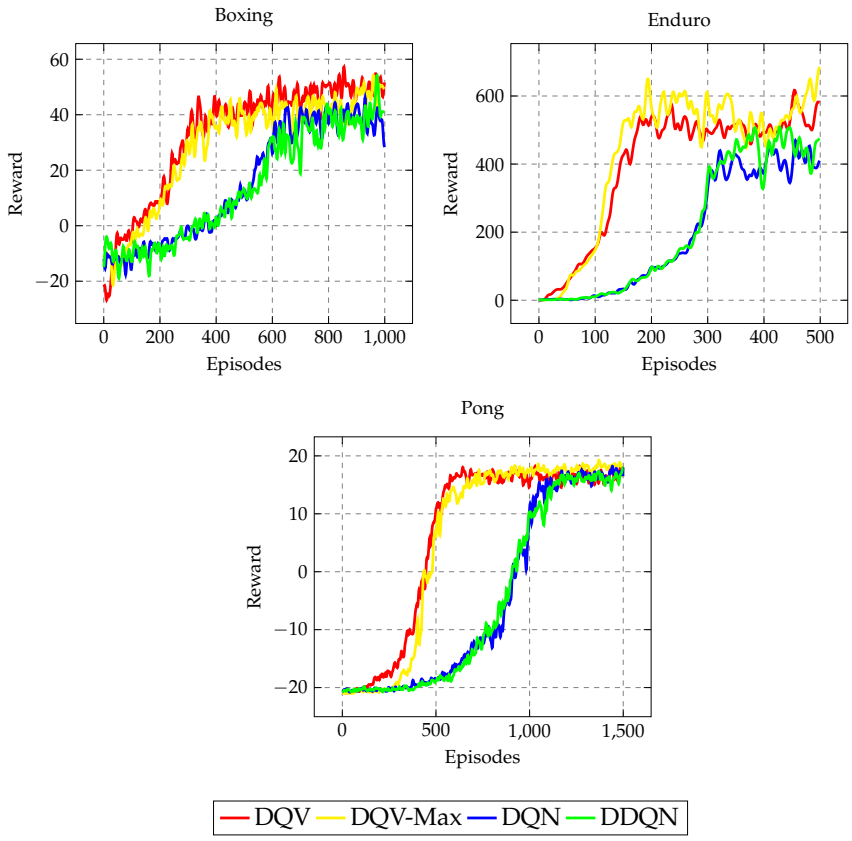


Figure 8.2: Learning curves obtained during training on three different `Atari` games by DQV and DQV-Max, and DQN and DDQN. We can observe that on these games both DQV and DQV-Max converge significantly faster than DQN and DDQN and that they obtain higher cumulative rewards on the `Enduro` environment.

- *a function approximator*: which is used for learning an approximation of a value function that could not be learned in the tabular RL setting due to a too large state-action space.
- *bootstrapping*: when the algorithms use a future estimated value for learning the same kind of estimate.
- *off-policy learning*: when a future estimated value is different from the one which would be computed by the policy the agent is following.

Van Hasselt et al. [260] have shown that the *'Deadly Triad'* is responsible for enhancing one of the most popular biases that characterize the Q-Learning algorithm: the overestimation bias of the Q function [259]. It is therefore natural to study how DQV and DQV-Max relate to the *'Deadly Triad'* of DRL, and to investigate up to what extent these algorithms suffer from the overestimation bias of the Q function. To do this we monitor the estimates that are given by the network that is responsible for approximating the Q function. More specifically, at

training time, we compute the averaged $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ over a set (n) of full evaluation episodes as defined by

$$\frac{1}{n} \sum_{t=1}^{n} \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta). \tag{8.9}$$

As suggested by Van Hasselt, Guez, and Silver [261] these estimates can then be compared to the averaged discounted return of all visited states that comes from an agent that has already concluded training. By analyzing whether the Q values which are estimated while training differ from the ones which should be predicted by the end of it, it is possible to quantitatively characterize the level of divergence of DRL algorithms. We report our results in Figs. 8.3, 8.4, 8.5 and 8.6 where the black full lines correspond to the value estimates that come from each algorithm at training time, while the coloured lines correspond to the actual averaged discounted return that is given by an already trained agent.

We can start by observing that the values denoting the averaged discounted return obtained by each algorithm differ among agents. This is especially the case when it comes to the `Enduro` environment, and is a result which is in line with what has been presented in Table 7.2: DQV and DQV-Max lead to better final policies than DQN and DDQN. Furthermore, when we compare these baseline values to the value estimates that are obtained during training, we can observe that the ones obtained by the DQN algorithm significantly diverge from the ones which should be predicted by the end of training. This behavior is known to be caused by the overestimation bias of the Q function which can be corrected by the DDQN algorithm. By analyzing the value estimates of DQV and DQV-Max we can observe that both algorithms produce value estimates which are more similar to the ones computed by DDQN than to the ones given by DQN. This is especially the case for DQV (Fig. 8.5). In fact, its value estimates nicely correspond to the averaged discounted return baseline, both on the `Pong` environment and on the `Enduro` environment. The estimates coming from DQV-Max, however, seem to diverge more when compared to DQV and DDQN's ones. This is clearer on the `Enduro` environment, where the algorithm does show some divergence (right plot of Fig. 8.6). However, we can also observe that this divergence is less strong when compared to DQN's one. The value estimates of the latter algorithm keep growing over time, while DQV-Max's ones get bounded while training progresses (see the two right plots of Fig. 8.3 and Fig. 8.6). This results in smaller estimated Q values. We believe that there are mainly two reasons why our algorithms suffer less from the overestimation bias of the Q function. When it comes to DQV, we believe that this algorithm suffers less from this bias since it is an on-policy learning algorithm. Such algorithms are trained on exploration actions with lower Q values. Because of its *on-policy* learning scheme,

DQV also does not present one element of the 'Deadly Triad', which might help reducing divergence. When it comes to DQV-Max, we believe that the reason why this algorithm does not diverge as much as DQN can be found in the way it approximates the Q function. One key component of the *'Deadly Triad'*, is that divergence occurs if the Q function is learned by regressing towards itself. As given by Eq. 8.8 we can see that this does not hold for DQV-Max, since the Q function bootstraps with respect to estimates that come from the V network. We believe that this specific learning dynamic, which also holds for the DQV algorithm, makes our algorithms less prone to estimate large Q values.

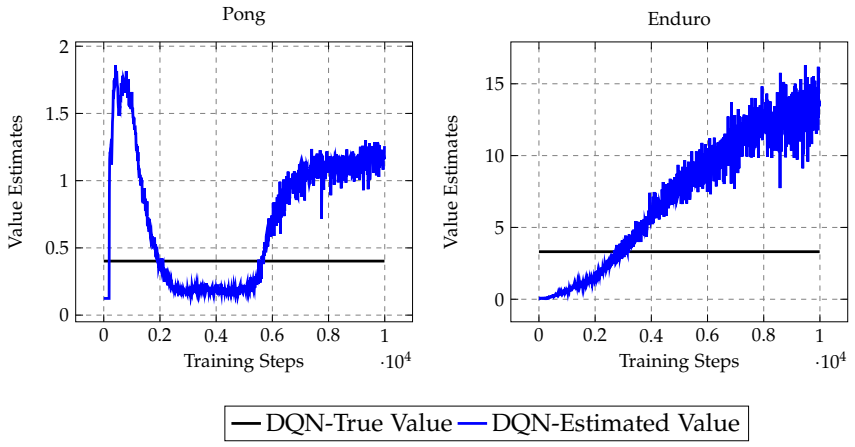


Figure 8.3: Results investigating the extent to which the DQN algorithm suffers from the overestimation bias of the Q function. We can observe that on the `Pong` environment during the early stages of training, the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates quickly grow, while on the `Enduro` game the values estimated by the Q network keep indefinitely growing, therefore making the algorithm significantly diverge from the real return that is obtained by a trained agent.

8.4 ADDITIONAL STUDIES

As introduced in Sec. 8.2 DQV and DQV-Max use two separate neural networks for approximating the Q function and the V function. To verify whether two different architectures are needed for making both algorithms perform well, we have experimented with a series of variants of the DQV-Learning algorithm. The aim of these experiments is that of reducing the number of trainable parameters that are required by the original version of DQV, and investigate whether its performance could get harmed when reducing the capacity of the algorithm. The studied DQV's extensions are the following:

1. *Hard-DQV*: a version of DQV which uses one single common neural network for approximating both the Q and the V functions. An additional output node (see Fig. 8.7a for an impression

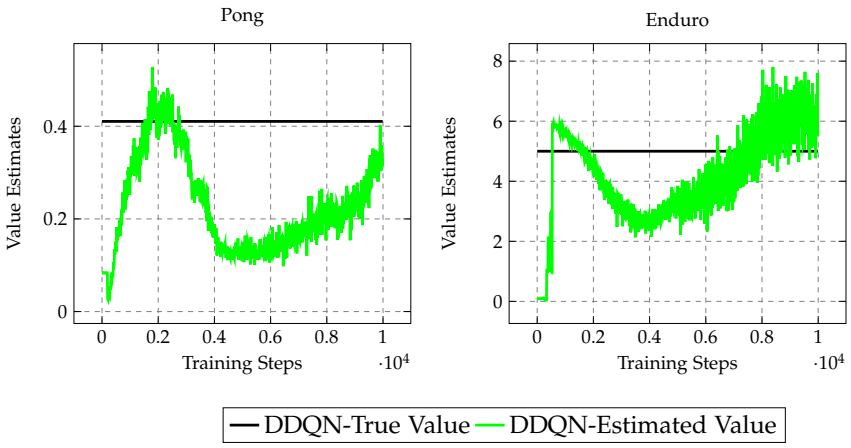


Figure 8.4: Results investigating the extent to which the DDQN algorithm suffers from the overestimation bias of the Q function. We can observe that compared to the analysis presented in Fig. 8.3, the DDQN algorithm prevents its Q-Network from diverging since on both `Atari` environments the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates do not diverge from the observed real return of a trained agent. These results replicate the findings reported by Van Hasselt et al. [260].

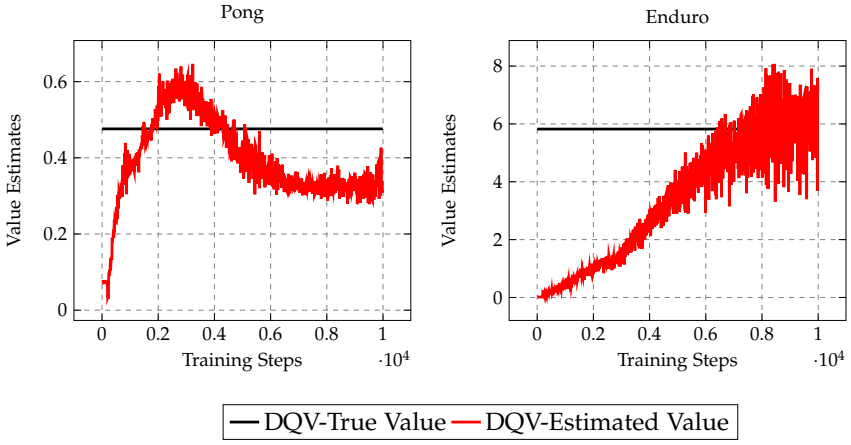


Figure 8.5: Results investigating the extent to which the DQV algorithm suffers from the overestimation bias of the Q function. We can observe that the performance of the algorithm is similar to the one observed in Fig. 8.4 for the DDQN algorithm. On both environments the estimated cumulative reward does not diverge from the real return that is obtained by the end of training, therefore suggesting that DQV-Learning does not suffer from the overestimation bias of the Q function. It is also worth noting the difference between the real return obtained by the DQN and the DDQN algorithms on the `Enduro` environment, and the one obtained by DQV. As can be seen by the black line, the real return obtained by a DQV agent is higher than DQN and DDQN's one, a result which shows that DQV converges to a better policy than DQN and DDQN.

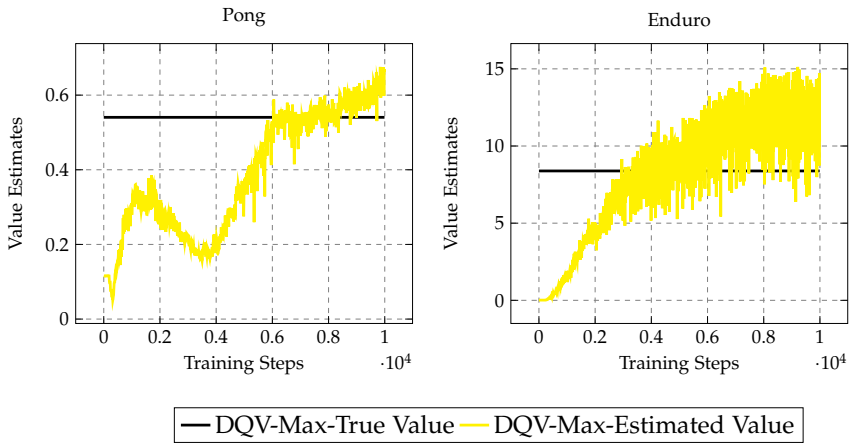


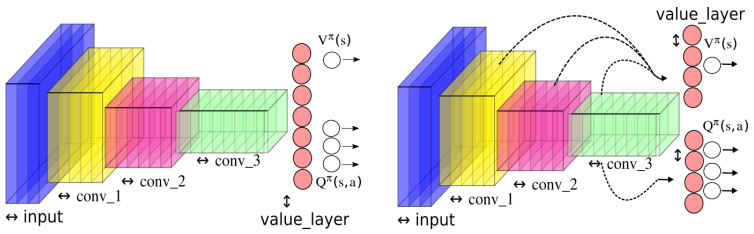
Figure 8.6: Results investigating the extent to which the DQV-Max algorithm suffers from the overestimation bias of the Q function. We can observe that on the `Pong` environment the value estimates of the algorithm are comparable to the ones of DDQN and DQV, therefore showing that DQV-Max also diverges significantly less than DQN. On the `Enduro` environment we can observe that the algorithm does diverge, although, differently from what is reported in Fig. 8.3 for the DQN algorithm, the $\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ estimates seem to converge towards an upper bound (≈ 15). Similarly to what is reported in Fig.8.5 we can again observe that the real return obtained by a trained agent is higher compared to the one obtain by DQN, DDQN and DQV, therefore confirming the results presented in Table 8.1 which see the DQV-Max algorithm as the best performing algorithm on the `Enduro` game.

of the architecture), needed for estimating the value of a state, is added next to the output nodes which estimate the different Q values (one per action). The parameters of this algorithm are therefore 'hardly-shared' among the agent, and provide the benefit of halving the total amount of trainable parameters of DQV. The different outputs of the network get then alternatively optimized according to Eq. 8.4 and 8.5.

2. *Dueling-DQV*: a slightly more complicated version of Hard-DQV which adds one specific hidden layer before the output nodes that estimate the Q and V functions. In this case, the outputs of the neural network which learn one of the two value functions, partly benefit from some specific weights that are not shared within the neural network. This approach is similar to the one used by the 'Dueling-Architecture' presented by Wang et al. [270], therefore the name Dueling-DQV. While it is well established that three convolutional layers are needed [161, 261] for learning the Q function, the same might not be true when it comes to learning the V function. We thus report experiments with three different versions of Dueling-DQV: Dueling-1st, Dueling-2nd, and Dueling-3rd. The difference between these

methods is simply the location of the hidden layer which precedes the output that learns the V function. It can be positioned after the first convolutional layer, the second or the third one (see Fig. 8.7b). Training this architecture is done as for Hard-DQV.

3. *Tiny-DQV*: the neural architectures used by DQV and DQV-Max that approximate the V function and the Q function follow the one which was initially introduced by the DQN algorithm [161]. This corresponds to a three-hidden layer convolutional neural network which is followed by a fully connected layer of 512 hidden units. The first convolutional layer has 32 channels while the last two layers have 64 channels. In Tiny-DQV we reduce the number of trainable parameters of DQV by reducing the number of channels at each convolution operation. Tiny-DQV only uses 8 channels after the first convolutional layer and 16 at the second and third convolutional layers. Furthermore, the size of the final fully connected layer is reduced to only 128 hidden units. The choice of this architecture is motivated by the work presented in [260] which studies the role of the capacity of the DDQN algorithm. Unlike the Hard-DQV and Dueling-DQV extensions, the parameters of Tiny-DQV are not shared at all among the networks that are responsible for approximating the V function and the Q function.



(a) (b)

Figure 8.7: Two representations of convolutional neural networks which jointly approximate the V function and the Q function with the aim of reducing DQV's learning parameters. On the left, an architecture which simply adds one output node to the network next to the output nodes which estimate the Q function. On the right an architecture in which a specific hidden layer precedes the output that is necessary for computing each value function. When it comes to the V function we experiment with different locations of such hidden layer, which is positioned after each possible convolution block.

The results obtained by these alternative versions of DQV are presented in Figs. 8.8, 8.9 and 8.10 where we report the learning curves obtained by the tested algorithms on six different `Atari` games. Each DQV extension is directly compared to the original DQV algorithm.

We can observe that all the extensions of DQV, which aim at reducing the number of trainable parameters of the algorithm, fail in performing as well as the original DQV algorithm. Starting from Fig. 8.8 we can observe that *Hard-DQV* does not only yield significantly lower rewards (see the results obtained on `Boxing`) but also presents more unstable training (as highlighted by the results obtained on the `Pong` environment). Lower rewards and unstable training also characterize the *Tiny-DQV* algorithm (see results on `Bank Heist` and `Crazy Climber` reported in Fig.8.10). Overall the most promising extensions of DQV are its *Dueling* counterparts, we have observed in particular that the best performing architecture over most of our experiments was the *Dueling-DQV-3rd* one. As can be seen by the results reported in Fig. 8.9 on the `Pong` environment we can observe that *Dueling-DQV-3rd* has a comparable performance to DQV, even though it converges slower. Unfortunately, *Dueling-DQV-3rd* still shows some limitations, in particular when tested on more complicated environments such as `Enduro`, we can observe that it under-performs DQV with $\approx$ 200 points. It is also worth mentioning that the idea of approximating the V function before the Q function explored by *Dueling-DQV-1st* and *Dueling-DQV-2nd* yielded negative results.

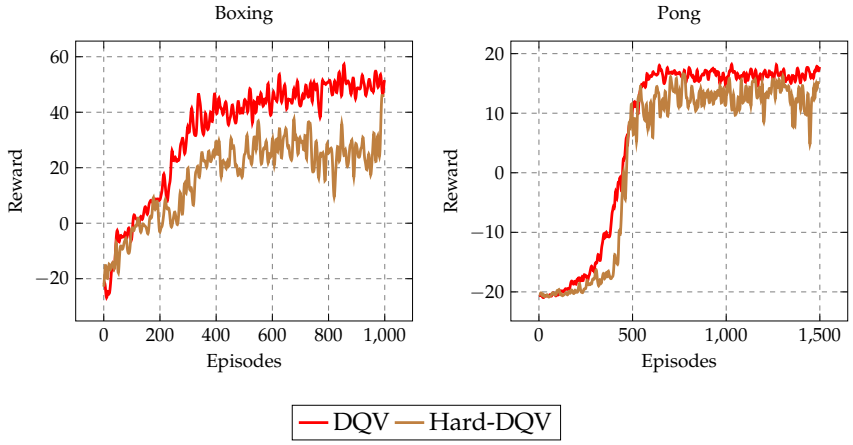


Figure 8.8: Our results which aim to approximate the V and the Q function with a unique, shared paramterized network, an approach that is heavily inspired by multi-task learning studies that can be found in supervised learning [33, 165, 292]. We can see that this extension of DQV, named Hard-DQV, significantly underperforms the original DQV-Learning algorithm.

8.5 DISCUSSION AND CONCLUSION

We have presented two novel model-free DRL algorithms which in addition to learning an approximation of the Q function also aim at learning an approximation of the V function. We have compared DQV and DQV-Max Learning to DRL algorithms which only learn

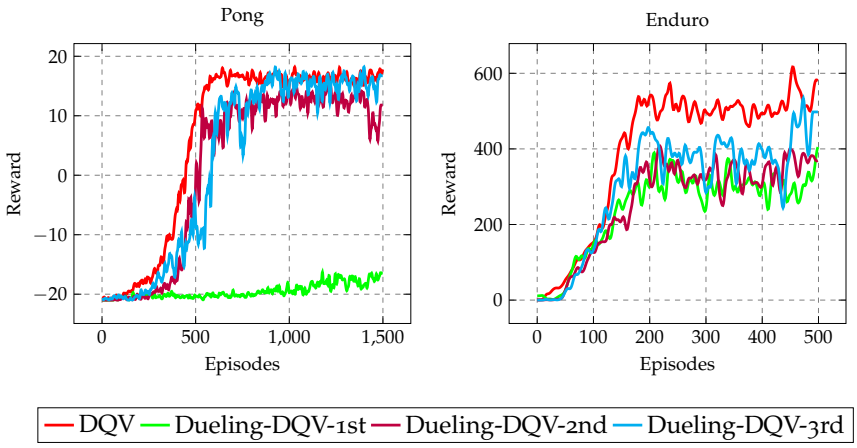


Figure 8.9: Our extensions of DQV that aim to reduce the amount of trainable parameters of the algorithm by following an approach similar to the one presented by Wang et al. [270] when "Dueling Networks" for DRL have been introduced. We can observe that among all the three Dueling-DQV extensions, only the Dueling-DQV-3rd one yielded good performance, but as highlighted by the results obtained on the `Enduro` environment, its performance is still inferior when compared to the one of the original DQV algorithm.

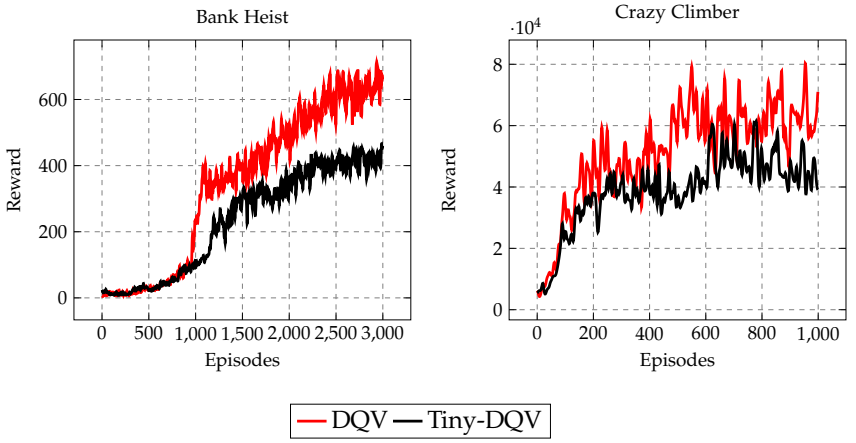


Figure 8.10: Learning curves obtained when reducing the capacity of the convolutional networks that approximate the V and the Q functions. We can observe that albeit each value function is approximated with its own parametrized network, the tiny-dqv extension still yields worse performance. These results highlight that it is not sufficient to simply have two separate neural networks for DQV to perform well, but that a crucial role in DQV's performance is played by the capacity of the networks that are used as well.

an approximation of the Q function, and showed the benefits which come from jointly approximating two value functions over one. Our newly introduced algorithms learn significantly faster than DQN and DDQN and show that approximating both the V function and the Q

function can yield significant benefits both in an on-policy learning setting as in an off-policy learning one. This specific training dynamic allows for a better learned Q function which makes DQV and DQV-Max less prone to estimate unrealistically large Q values. All these benefits come however at a price: to successfully learn two value functions, two separate neural networks with enough capacity are required.

In the coming chapter we will analyze the transfer learning properties of convolutional neural networks that get trained with the DQN, DDQN and the newly introduced DQV-Learning algorithms.

9

ON THE TRANSFERABILITY OF DEEP-Q NETWORKS

Outline

Throughout the second part of this dissertation we have seen how Transfer Learning (TL) has become an efficient machine learning paradigm that allows overcoming some of the hurdles that characterize the successful training of deep neural networks, ranging from long training times to the needs of large datasets. While exploiting TL is a well established and successful training practice in Supervised Learning (SL), its applicability in Deep Reinforcement Learning (DRL) is rarer. Therefore, in this chapter, we study the level of transferability of three different variants of Deep-Q Networks on popular DRL benchmarks as well as on a set of novel, carefully designed control tasks. Differently from what we have extensively observed in Chapters 5, 6 and 7, we now show that transferring neural networks in a DRL context can be particularly challenging and is a process which in most cases results in negative transfer. In the attempt of understanding why Deep-Q Networks transfer so poorly, we gain novel insights into the training dynamics that characterizes this family of algorithms. The chapter is structured as follows: we start with a general introduction in Sec. 9.1 while we then move towards presenting a large-scale TL study that considers agents that are trained with the DQV and DDQN DRL algorithms and that get transferred across different Atari games. The results of this study are critically discussed and studied in Sec. 9.3 through the use of novel, carefully designed control problems, that in Sec. 9.4 allow us to discover how Deep-Q Networks tackle RL problems. The chapter finishes with Sec. 9.5 where we describe how its content contributes to the DRL transfer literature.

This chapter is based on the following publication: Sabatelli and Geurts [205].

9.1 INTRODUCTION

In the previous chapter we have seen that neural networks can be extremely successful in a model-free RL setting. Large part of their success can be attributed to their ability of serving as feature extractors as well as function approximators, a property that allows them to successfully learn optimal value functions [160, 161, 209, 261, 270,

290], but also stochastic policies [65, 78, 135, 158, 219–221, 269], and models of an environment [77, 79–81, 108] that as reviewed in Chapter 3 is usually formalized as a Markov Decision Process (MDP) [186]. Despite the many remarkable achievements, training a DRL agent is a process that can be very time-consuming. The task of solving an optimal decision making problem is, in fact, a challenging problem of its own, which is sometimes made even more difficult by the DRL community itself, which requires DRL practitioners to test the performance of their algorithms on benchmarks that are computationally very expensive (for a position paper about this topic see [170]). One way of overcoming the need of individually training a DRL agent from scratch each time a new RL problem is encountered is based on Transfer Learning (TL). Despite being largely adopted by the Supervised Learning (SL) community, as extensively documented throughout the second part of this dissertation [48, 96, 103, 164, 206, 264], the typical TL approaches reviewed in Chapter 4 such as off the shelf feature extraction, or fine-tuning [224], have rarely been thoroughly studied from a DRL perspective. Therefore, the degree of transferability of DRL algorithms is not yet known. In this chapter, we keep focusing on value-based, model-free algorithms, a family of techniques which trains neural networks with the intent of learning an approximation of an optimal value function. While several of such algorithms, commonly denoted as Deep-Q Networks, exist, research studying their TL properties is, on the contrary scarce, and a clear answer to the question *"How transferable are Deep-Q Networks?"* has yet to be given. In the attempt to clearly answering this question, this chapter presents the following three contributions:

- We present a first large scale empirical study that analyses the TL properties of popular model-free DRL algorithms on the Atari Arcade Learning Environment (ALE), where we show that transferring pre-trained networks in a DRL context can be a very challenging task.
- We design a set of novel, control experiments which allows us to thoroughly characterize the TL dynamics of Deep-Q Networks.
- While studying Deep-Q Networks from a TL perspective, we discover novel learning dynamics that provide a better understanding of how this family of algorithms deals with RL tasks.

9.2 A LARGE-SCALE EMPIRICAL STUDY

In this section, we carry out a large-scale TL experiment on several games from the Atari Environment (Sec. 9.2.1). The experimental protocol is detailed in Sec. 9.2.2 and results are discussed in Sec. 9.2.3.

9.2.1 *The Atari Environments*

In this study, we keep using the Atari Arcade Learning Environment (ALE) [17] that was also used in Chapter 8. Next to being one of the most popular benchmarks in DRL, the ALE is particularly well suited for TL research as it allows to choose among a set of 57 Atari games that can be used as source $\mathcal{M}_S$ and target $\mathcal{M}_T$ MDPs within a deep transfer learning setting. Since training a model-free agent on the games of the ALE is a process which can be computationally very expensive, we have carefully selected a subset of 10 different environments. Numerous reasons guided the game selection process. First, we have selected games for which we guarantee that a model-free DRL agent can learn a good policy for. Since, as discussed by Lazaric [125], one of the key requirements of TL is that of correctly identifying and transferring knowledge across source and target tasks, we naturally ensured that some knowledge coming in the form of neural network parameters representing a near-optimal value function was available for transfer. Second, while it is true that all of the selected games result in an agent that can improve its policy over time, some games were chosen because the learned policy resulted in a final performance that was not on par with that of a human expert player. This is, for example, the case of the `Frostbite` game, where the gap in performance between an agent trained with our DQV-Learning algorithm [210] ($\approx$ 270) and a human expert player ($\approx$ 4300) is particularly significant (see Table 8.1 for a reminder). It follows that `Frostbite` is an interesting target task for transfer, as the agent's performance can potentially be improved through TL. Furthermore, we have also ensured that among the selected games, some environments are more similar to each other than others. This is, for example, the case for the `Ms. Pacman` and `Bank Heist` games which, as can be seen in Fig. 9.1, are two games where the state space is represented as a maze, and where the end goal of an agent is that of learning how to navigate it. In like manner, we have also included games that are very different from each other as is, e.g., the case for the `Crazy Climber` and `Pong` games, where it is clear from Fig. 9.2 that no visual similarities are shared among the two environments. Including visually similar and dissimilar games allows us to investigate whether, as is the case for supervised learning, a source task is particularly well suited for transfer if it is similar to its respective target task [153].

9.2.2 *Experimental Setup*

We investigate the TL performance of agents that get trained with the DQV-Learning algorithm [206], and with the DDQN algorithm [261]. We take models which come as pre-trained on 10 Atari games and transfer them to all remaining environments. We mostly consider

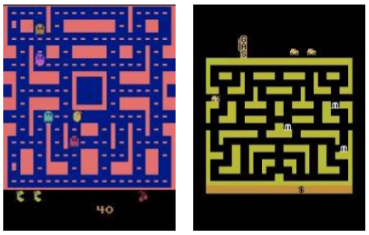


Figure 9.1: The visually similar `Ms Pacman` and `Bank Heist` games.

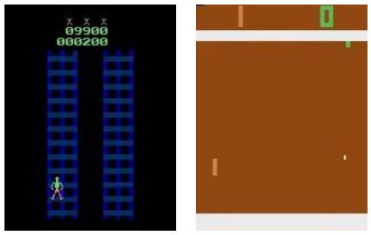


Figure 9.2: The highly different `Crazy Climber` and `Pong` games.

the same games for both algorithms (`Bank Heist`, `Boxing`, `Crazy Climber`, `Fishing Derby`, `Frostbite`, `James Bond`, `Ms. Pacman`, `Pong` and `Zaxxon`) with the only exception that for DDQN `Frostbite` is replaced by `Gopher`, and `Zaxxon` is replaced by `Ice Hockey` as the `Frostbite` and `Zaxxon` DDQN agents failed to improve their policy whilst training. It is worth noting that TL is particularly easy to perform as both algorithms learn an approximation of the optimal state-action value function $Q(s, a; \theta)$ by training a convolutional neural network directly on the images representing the state of the game. Since the state space across Atari games is always represented as an $84 \times 84 \times 4$ tensor, it is straightforward to transfer the same neural architecture among various Atari environments without needing special modifications. However, the only modification that we apply to a pre-trained network concerns its last layer responsible for estimating the different Q values, which we always replace and randomly re-initialize. Following the typical deep transfer learning literature, we investigate whether in DRL it is as beneficial as it is in supervised learning to transfer a network that comes as pre-trained on $\mathcal{M}_S$ and fine-tune it on $\mathcal{M}_T$. We do this by quantitatively assessing the transfer learning benefits on each $\mathcal{M}_S/\mathcal{M}_T$ pair by computing the area ratio metric $\mathscr{R}$ [249]. Specifically, given a learning curve representing the performance of an agent pre-trained on $\mathcal{M}_S$, and that of an agent that is instead trained from scratch, we compute $\mathscr{R}$ as follows:

$$\mathscr{R} = \frac{\text{area of } \mathcal{M}_S - \text{area of } \mathcal{M}_T}{\text{area of } \mathcal{M}_T}. \tag{9.1}$$

where the area under the curve is approximated via trapezoidal numerical integration

$$\int_a^b f(x)\, dx \approx (b-a) \cdot \frac{1}{2}(f(a) + f(b)) \tag{9.2}$$

and a and b correspond to the first and last training epochs respectively.

9.2.3 *Results*

The results on each $\mathcal{M}_S/\mathcal{M}_T$ pair for both the DQV and DDQN algorithms are presented in Table 9.1. In each cell of the tables, we report the area ratio metric defined in Eq. 9.1: the lower (resp. higher) this score, the less (resp. more) beneficial it is to transfer and fine-tune a pre-trained agent. When it comes to the DQV algorithm, we can see that, out of nine target environments, there is only one Atari game for which it is always beneficial to transfer and fine-tune a pre-trained model: `Fishing Derby`. In fact, a positive area ratio score is obtained no matter which source environment is used for pre-training, although the best results have been obtained when starting from an `Enduro` or `Pong` pre-trained network, which both resulted in an area ratio score of ≈ 0.72. Positive transfer can also be observed on the `Frostbite` and `James Bond` games, but only for a limited number of source games. For example, a `Bank Heist` pre-trained agent transfers well to both target games as it obtains an area ratio score of 0.729 and 0.973 respectively, but the same cannot be said for an `Enduro` pre-trained network, which on `Frostbite` results in absent transfer (the area ratio score is, in fact, -0.017), and yields negative transfer on `James Bond` ($\mathscr{R} = -0.41$). We can also observe that there are environments where it is surprisingly never beneficial to transfer and fine-tune a pre-trained agent. This is, for example, the case for the `Bank Heist` and `Pong` games, where independently from which source game $\mathcal{M}_S$ is used for pre-training, a negative area ratio score is always obtained. Furthermore, it can also be observed that transfer learning across environments is not symmetric, as one source game $\mathcal{M}_S$ can result in positive transfer when it gets transferred to a certain target game $\mathcal{M}_T$, but the same outcome is not obtained when transfer is performed in the opposite direction. As an example we can consider the `Boxing/Fishing Derby` games: positive transfer is obtained when transferring from `Boxing`→`Fishing Derby` ($\mathscr{R} = 0.552$), but negative transfer is obtained when transferring from `Fishing Derby` → `Boxing` ($\mathscr{R} = -0.893$). When it comes to the DDQN algorithm, similar conclusions can be drawn: we can again observe that there are only very few cases for which it is beneficial to transfer and fine-tune a pre-trained DRL agent. Examples of such cases are networks that are pre-trained on `Ice Hockey` and `James Bond` which get transferred to `Boxing` ($\mathscr{R} = 0.245$ and $\mathscr{R} = 0.232$ respectively), or `Boxing` and `Enduro` models that get transferred to `Pong` ($\mathscr{R} = 0.936$ and $\mathscr{R} = 0.248$). `Bank Heist` and `Pong` are again the two target environments for which most of the transferred source models resulted in negative transfer, while differently from the experiments performed with the DQV algorithm, this time no positive transfer can be observed on `Fishing Derby`. Overall, the process of fine-tuning a pre-trained DDQN agent mostly results in absent transfer, as can

Table 9.1: The results obtained when fine-tuning ten different pre-trained agents (rows) on nine other Atari games (columns), with DQV (top table) and DDQN (bottom table). Positive values (in green) represent positive transfer, while negative values (in red) represent negative transfer. The darker the color, the higher the absolute value of the area ratio score.

DQV	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Frostbite	JamesBond	MsPacman	Pong	Zaxxon
BankHeist	-	-0.019	-1	-0.317	0.5	0.729	0.973	-0.089	-1.238	-0.998
Boxing	-0.494	-	-0.278	-0.852	0.552	-0.01	0.247	-0.184	-0.841	-0.999
CrazyClimber	-0.569	-0.261	-	-0.593	0.19	0.277	0.621	-0.111	-1.206	-0.178
Enduro	-0.571	-0.018	-0.25	-	0.726	-0.017	-0.41	-0.08	-0.466	-0.164
FishingDerby	-1	-0.893	-0.093	-0.45	-	0.068	0.197	-0.136	-3.083	-0.999
Frostbite	-0.933	0.024	-1	-0.348	0.222	-	0.569	0.009	-0.663	-0.076
JamesBond	-0.123	-0.106	-0.131	-0.033	0.519	0.262	-	0.218	-1.329	-1
MsPacman	-0.985	-0.219	-0.012	-0.494	0.6	0.346	0.398	-	-1.646	-0.997
Pong	-1	-0.083	-0.428	-0.476	0.725	-0.024	0.896	0.123	-	-0.729
Zaxxon	-0.76	-0.028	0.037	-0.116	0.385	0.16	-0.253	0.06	-1.602	-

DDQN	BankHeist	Boxing	CrazyClimber	Enduro	FishingDerby	Gopher	IceHockey	Jamesbond	MsPacman	Pong
BankHeist	-	0.121	-0.378	-0.006	-0.107	0.042	-0.006	-0.058	0.001	-3.013
Boxing	-0.316	-	-0.104	-0	0.038	0.06	0.015	-0.225	-0.027	0.936
CrazyClimber	-0.192	-0.487	-	-0.012	-0.084	0.016	0.015	0.016	-0.015	-2.64
Enduro	-0.296	0.193	-0.167	-	0.039	0.03	0.019	-0.235	-0.039	0.248
FishingDerby	-0.212	-0.545	-1	-0.085	-	0.016	0.001	-0.055	-0.026	-0.935
Gopher	-0.466	0.044	-0.108	-0.005	0.007	-	-0.005	-0.094	-0.02	-1.816
IceHockey	-0.046	0.245	-0.067	0.014	-0.178	0.072	-	0.037	-0.015	0.112
Jamesbond	-0.145	0.232	-0.064	0.005	-0.267	0.031	-0.092	-	-0.006	-1.578
MsPacman	-0.173	-1.179	-0.129	-0.06	0.003	-0.019	0.007	0.071	-	-2.774
Pong	-0.127	0.028	-0.12	0.01	0.037	0.042	0.002	-0.174	-0.006	-

be observed by the area ratio scores obtained on `Enduro`, `Fishing Derby`, `Gopher` and `Ice Hockey` which are all ≈ 0 on average.

9.3 CONTROL EXPERIMENTS

The results presented in the previous section seem to be questioning the level of transferability of DRL agents. In fact, the training strategy of fine-tuning a pre-trained model on $\mathcal{M}_T$ does not result in the same type of performance gains that we have extensively observed throughout the second part of this dissertation. To better characterize their TL properties, we have designed a set of simple control experiments that allow us to examine their transfer learning behavior in training conditions that do not require extraordinarily long training times for learning an optimal policy.

9.3.1 *The Catch Environments*

To this end, we have implemented four different versions of the `Catch` game, a simple RL task that was first presented by Mnih, Heess, Graves, et al. [159], and that has been widely used within the literature for investigating the performance of DRL algorithms in a fast, and computationally less expensive manner than the one required by the `Atari` games [5, 282]. In the game of `Catch`, an agent controls a paddle at the bottom of the environment, represented by a

21×21 grid, and has to catch a ball falling from top to bottom, which can potentially bounce off walls. At each time step, the agent can choose between three actions: move the paddle one pixel to the right, move it to the left, or do not perform any of the aforementioned actions, therefore keeping its paddle in the same position in the grid. An RL episode ends either when the agent manages to catch the ball, in which case it receives a reward of 1, or when it misses the ball, which naturally results in a reward of 0. Following the design choices presented in [280], we model the ball to have vertical speed of $v_y = -1\, cell/s$ and horizontal speed of $v_x \in \{-2, -1, 0, 1, 2\}$. From now on, we will refer to this version of the game as `Catch-v0`, as it is the most basic and simplest form of the game that will be used throughout our experiments. Next to `Catch-v0` we have implemented three slightly different and arguably more complex versions of the game as well: `Catch-v1`, where we increased the complexity of the game by reducing the size of the paddle that the agent controls. While for `Catch-v0` its size is of five pixels, in `Catch-v1` it is of two pixels, therefore requiring the agent to be more precise if it wants to successfully catch the falling ball. The second alternative version of `Catch` is `Catch-v2`. In this case, the dynamics of the game are identical to the ones that define `Catch-v0`; however, the way the 21×21 grid is represented changes. While in `Catch-v0` as well as in `Catch-v1` the state is represented by a binary grid where all pixels, but the ones representing the paddle and the ball have a value of 0, in `Catch-v2` the cells around the paddle and the ball can have a random value between 0 and 255. This design choice makes it much

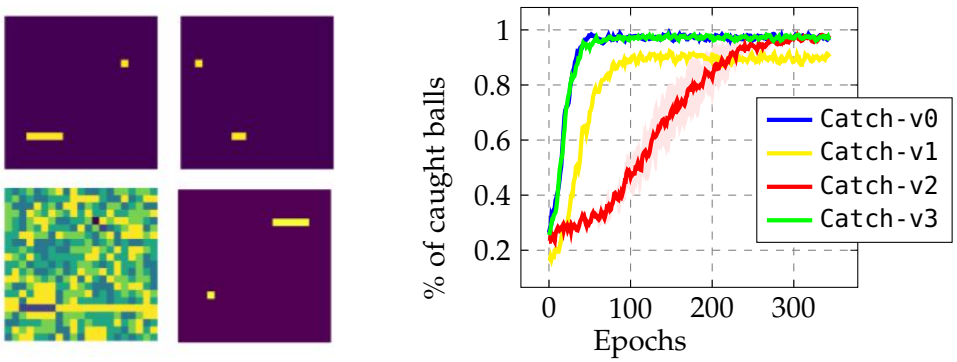


Figure 9.3: Image on the left: the four different versions of the `Catch` environment. In clockwise order: `Catch-v0`, `Catch-v1`, `Catch-v3` and `Catch-v2`. Image on the right: learning curves obtained by a DQN agent that is trained from scratch on the aforementioned `Catch` versions. Shaded areas correspond to ± 1 std. obtained over 5 different random seeds.

harder for a convolutional network to correctly locate and identify the position of the paddle and of the falling ball and makes `Catch-v2` the arguably most complex version among the different `Catch` envi-

ronments. Lastly, we have implemented `Catch-v3`, a version of `Catch` which is identical to the one that is modeled by `Catch-v0` with the main difference that the representation of the state is now mirrored, therefore requiring the agent to look at different parts of the grid if it wants to locate the paddle, and understand that the ball is unnaturally moving from the bottom to the top. For an impression of all four `Catch` versions see the left image of Fig. 9.3. Given the overall simplicity of the different `Catch` environments, we now train a DQN agent instead of the arguably more complex DQV and DDQN agents that we considered in Sec. 9.2. As we can see from the results reported in the right plot of Fig. 9.3, averaged over five different runs, the agent is able to successfully learn a near optimal policy for all `Catch` versions. When it comes to `Catch-v0`, `Catch-v2` and `Catch-v3` we can observe that by the end of training, the agent is able to catch $\approx 100\%$ of the falling balls, whereas its performance is slightly worse ($\approx 90\%$) when it comes to `Catch-v1` [1]. We can also observe that among the different `Catch` versions, `Catch-v0` and `Catch-v3` appear to be the easiest ones, as the agent requires significantly less training episodes to converge when compared to `Catch-v1` and `Catch-v2`. Furthermore, in line with the explanation presented beforehand, our results also confirm the hypothesis that `Catch-v2` is the overall most complicated `Catch` version, as learning requires significantly more, and potentially unstable, training epochs.

9.3.2 *From one Catch to Another*

We now replicate the TL study presented in Sec. 9.2 on the aforementioned `Catch` environments, with the hope of identifying why the process of fine-tuning a pre-trained convolutional neural network in a DRL context, seems to not be as beneficial as it is in the supervised learning one. Our goal is to find at least one pair of `Catch` environments which results in positive transfer, and to then potentially identify some properties within the different `Catch` versions that could also hold for the pairs of `Atari` games which have yielded positive transfer in Tables 9.1 and 9.1. We formulate two hypothesis, based on which, we expect to experimentally observe positive transfer. First, we foresee that positive transfer will happen for all possible `Catch` combinations, as in the end the source MDP $\mathcal{M}_S$ and the target MDP $\mathcal{M}_T$ do not significantly differ from each other: in fact, the main task across different `Catch` versions remains that of catching a falling ball; the action space is identical; and so is the reward function that always returns a value of 1 when the agent succeeds in catching the

1 Please note that the performance on `Catch-v1` can be improved by increasing the complexity of the DQN agent by adding one more convolutional layer. However, as the goal is to transfer the same type of model across `Catch` games, we did not modify the architecture of the DQN agent, at the cost of having a slightly worse performing model.

falling ball. This hypothesis is also motivated by the results obtained in Chapter 5 and 7 were we have shown that the higher the similarity between the source domain $\mathcal{D}_S$ and the target domain $\mathcal{D}_T$, the better the performance of a transferred pre-trained network. Second, in the case the previous hypothesis will not be empirically supported, we expect to at least observe positive transfer when using a model that comes as pre-trained either on `Catch-v1` or on `Catch-v2`. In fact, as described above and also shown by the performance reported in Fig. 9.3, these are the two most complicated versions of the `Catch` environment. As reviewed in Chapter 4, in supervised learning one of the main factors that makes a certain source task $\mathcal{T}_S$ good for transferring is its complexity [153], which is usually defined in terms of dataset size and number of classes to classify, therefore we expect the complexity of the source game to play an important role within DRL as well. Similarly to what we did for DQV and DDQN in Sec. 9.2, we take the four different models that have been trained from scratch on their respective `Catch` version, randomly re-initialize their last layer (responsible for estimating the different state-action values $Q(s, a)$) and fully fine-tune the pre-trained network on the three remaining `Catch` environments. The results of this study are reported in

Table 9.2: The area ratio obtained after fine-tuning a pre-trained DQN agent on the different `Catch` environments. We can see that no matter which source game is used for pre-training, transfer learning surprisingly never results in positive transfer.

	Catch-v0	Catch-v2	Catch-v3	Catch-v4
Catch-v0	-	-0.026	-0.486	-0.479
Catch-v2	-0.16	-	-0.121	-0.248
Catch-v3	-0.406	-0.313	-	-0.465
Catch-v4	-0.016	-0.24	-0.179	-

Fig. 9.4, where, from left to right, we show the performance that is obtained when considering `Catch-v0`, `Catch-v1`, `Catch-v2` and `Catch-v3` as target MDP $\mathcal{M}_T$. The performance of each transferred network is compared against the performance that is obtained after training a DQN agent from scratch which matches with the results reported in Fig. 9.3. Surprisingly we found that fine-tuning a pre-trained DQN agent never resulted in positive transfer learning. This can clearly be seen in all plots represented in Fig. 9.4 and by the results reporting the area ratio metric in Table 9.2. The only case where starting from a pre-trained network appeared to be at least in part beneficial is represented by the first plot of Fig. 9.4 when `Catch-v1` is considered as source MDP $\mathcal{M}_S$. In this case we can in fact observe some learning speed improvements within the first 25 learning epochs. This is not surprising as an agent which is able to catch a ball with a small pad-

dle (as defined by the game `Catch-v1`) should in principle also be able to do this when the size of its paddle is larger (which is the case for `Catch-v0`). What is more surprising, however, is that while training progresses we see that the performance of a `Catch-v1` pre-trained model starts deteriorating and that this model barely converges to the same performance that is obtained by a model trained from scratch. When it comes to all other $\mathcal{M}_S/\mathcal{M}_T$ pairs we see that pre-trained networks always perform significantly worse than randomly initialized models trained from scratch, with some extreme cases, as the one reported in the last plot of Fig. 9.4, where a `Catch-v0` pre-trained agent is barely able to improve its policy over time at all. These results invalidate our two hypotheses mentioned above as they clearly show that positive transfer in DRL does not arise when $\mathcal{M}_S$ and $\mathcal{M}_T$ are similar, nor when $\mathcal{M}_S$ is more complex than $\mathcal{M}_T$.

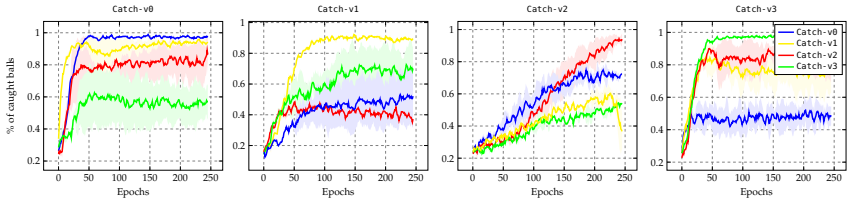


Figure 9.4: The results obtained after using a pre-trained `Catch` agent and fine-tuning it on a different `Catch` version. We can observe that despite all `Catch` versions being very similar no positive transfer is ever observed, as a model trained from scratch always outperforms a pre-trained, fine-tuned network.

9.3.3 *Self-Transfer*

The surprisingly poor transfer learning performance observed in the previous experiment made us question the level of transferability of Deep-Q Networks even more. To further characterize their TL properties, we decided to investigate whether pre-trained DRL agents are at least able to transfer to themselves. To this end, we studied what happens when a DQN agent gets transferred to a version of `Catch` that matches with the version of the game that was also used during the pre-training stage. This experiment is in large part identical to the one presented in Sec. 9.3.2, with the only difference being that now $\mathcal{M}_S = \mathcal{M}_T$. Moreover, differently from the previous study, we now also investigate what happens if instead of fine-tuning the network completely, we just use the pre-trained DQN agent as a simple feature extractor, therefore only training its last layer (the head) responsible for estimating the different state-action values. Our results are presented in Fig. 9.5, where for each `Catch` version, the full lines represent the performance of a network that is trained from scratch, whereas the dashed and dotted lines respectively report the perfor-

Table 9.3: The area ratio scores obtained after performing self-transfer. We can see that if only the last linear layer is trained, then positive transfer is obtained on all `Catch` environments, whereas if the network is fine-tuned, positive transfer is (in part) only obtained on `Catch-v2`.

	Catch-v0	Catch-v1	Catch-v2	Catch-v3
Only-Head	0.05	0.141	0.674	0.059
Fine-Tuning	0.017	-0.218	0.393	-0.236

mance that is obtained when the network is either used as simple feature extractor or entirely fine-tuned. We can see that if a pre-trained Deep-Q Network is used as a simple feature extractor, the agent can converge to the optimal policy almost immediately. In fact, as can consistently be observed in all plots of Fig. 9.5 and from the results presented in Table 9.3, training only the last layer of a pre-trained network yields positive transfer for all different `Catch` versions. However, when a fine-tuning training strategy is adopted, much more surprising results have been obtained. First, and more importantly, we can see that despite all models showing some learning speed improvements at early training iterations, their final performance is never on par with the one that is obtained when the same kind of model is either used as a feature extractor or trained from scratch (the dotted lines are consistently below the dashed and full lines). While when it comes to `Catch-v0` the policy learned by a fine-tuned model still allows the agent to successfully catch $\approx$ 95% of the falling balls, the same cannot be said when the models are tested on `Catch-v1`, `Catch-v2` and `Catch-v3`, where the difference in terms of performance between a model trained from scratch and a fine-tuned one is much more significant. Please also note that special attention should be given to `Catch-v2`, which is an environment where the area ratio score reported in Table 9.3 can be misleading as it does not entirely reflect the quality of the final policy learned by the agent. In fact, while it is true that an $\mathscr{R}$ value of 0.393 is obtained, it is worth noting that a fine-tuned network converges to a policy that is significantly worse than the one of a network trained from scratch, as the agent is only able of catching $\approx$ 80% of the falling balls. Second, as highlighted by the large variance across different training runs, fine-tuning on `Catch-v1` and `Catch-v3` resulted in highly unstable learning as well.

9.4 THE TWO LEARNING PHASES OF DEEP-Q NETWORKS

As reviewed in Sec. 3.7 of Chapter 3 a prototypical Deep-Q Network takes as input an image representing the state of the environment and processes it through a series of convolutions and a fully connected layer. When this is done, it outputs as many $Q(s, a)$ values as

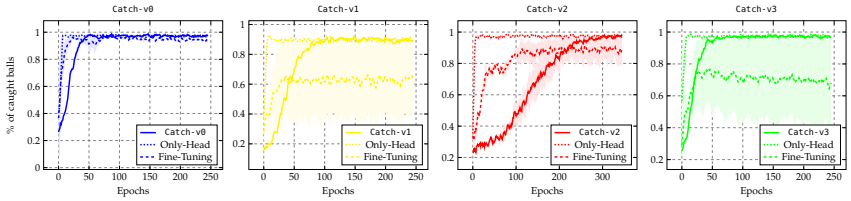


Figure 9.5: The results of our self-transfer experiments. From left to right the performance obtained on `Catchv-0`, `Catch-v1`, `Catch-v2` and `Catch-v3` after either training only the last linear layer of a pre-trained Deep-Q Network (dotted lines), or after wholly fine-tuning the model (dashed lines). We can see that the former transfer learning strategy yields significantly better results, and that a fine-tuning approach results in networks that in three cases out of four are not even able to transfer to themselves.

there are actions available to the agent, a process that corresponds to learning a linear policy in the latent feature space of the network. It follows that by the end of training, such a model has to serve two purposes: it has to perform as a feature extractor as well as an optimal value function approximator. Extracting relevant features from high dimensional inputs and learning an optimal value function can arguably be seen as two separate tasks; yet, despite their dissimilarity, we believe that they are more interconnected than one might expect. Specifically, we hypothesize that while learning, a Deep-Q Network has to carefully find a balance between training the parts that serve as feature extractors and the components that are responsible for estimating a policy. The poor TL performance observed throughout this chapter could therefore be the result of using models where the feature extractor component of an agent, as it comes as pre-trained, is too detached from the respective final layer of the network, which is randomly initialized instead. To show that a Deep-Q Network has to carefully coordinate training its feature extractor components and its final layer, let us consider the left image of Fig. 9.6. The figure depicts how the weights of an agent, whose feature extractor layers are represented by a square and the linear layer is represented through circles, change according to the self-TL experiments presented in Sec. 9.3.3. Each experiment is represented through two networks, one on the left side of the arrow representing the source model, and a second one, on the right part of the arrow, obtained by the end of training. From top to bottom, and from left to right, the first three network pairs represent the training process of: a randomly initialized model trained from scratch, a pre-trained model whose only last linear layer is trained after random initialization, and a pre-trained agent who gets fully fine-tuned. Following the results presented in Sec. 9.3.3, we know that positive transfer is only obtained when the last linear layer is trained in isolation after being randomly re-initialized,

whereas negative transfer is obtained if a fine-tuning training strategy is adopted.

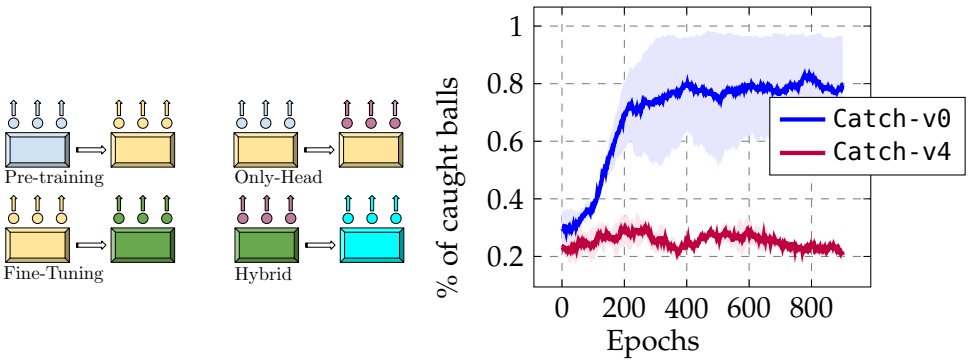


Figure 9.6: Image on the left: a visualization of differently initialized Deep-Q Networks before and after training. Image on the right: a successful example of positive transfer.

We now investigate the TL performance of a model that is a combination of the Only-Head and Fine-Tuning settings (see bottom right image of Fig. 9.6 for a visualization). Specifically, we fine-tune a Deep-Q Network whose last linear layer is initialized with the parameters that yielded positive transfer in Sec. 9.3.3 (Only-Head in Fig. 9.6), whereas its convolutional and fully connected layers are initialized with the parameters that yielded negative transfer (Fine-Tuning in Fig. 9.6). We visualize the self-transfer performance of these models, denoted as 'Hybrid", as they are a hybrid combination of two differently pre-trained networks, in Fig. 9.7 with a cyan dashed dotted line. We can observe that learning is characterized by a very atypical behavior: the network starts by improving its performance (thanks to the already trained final layer); it then goes through a second stage where it starts to perform more poorly (due to the poor feature extractor part), and then finally starts learning stably (when the feature extractor and the head of the model are synchronized). We believe that the poor TL performance observed throughout this chapter is, therefore, the result of models which could not find a balance between a randomly initialized head and their respective pre-trained layers which are too biased towards the source task.

Based on these results, one critical question still remains to be answered: how come positive transfer for some Atari games was observed in Sec. 9.2?. We believe that the answer to this question does not lie within the feature representations that a Deep-Q Network learns, but rather in some inner properties of the environment that is used as target task and that favors a Deep-Q Network to correctly synchronize its components. As a proof of concept, we have created one final `Catch` environment, called `Catch-v4`, which is identical to `Catch-v0` with the only difference being that a positive reward is returned to the agent only if it manages to catch five falling balls in a

row. We can see from the right image of Fig. 9.6 that a model trained from scratch (represented by the purple line) is not able to improve its policy over time at all, as the reward signal is probably too sparse for learning, whereas a `Catch-v0` model is now able to yield positive transfer. We hence believe that some environments, if combined with certain learning algorithms, are more prone to positive transfer than others, as was, e.g., the case for `Fishing Derby` and DQV-Learning where positive transfer was observed no matter what source task was used for pre-training. This is not due to the representations that are learned by a pre-trained network but rather because of some specific dynamics within the target MDP $\mathcal{M}_T$.

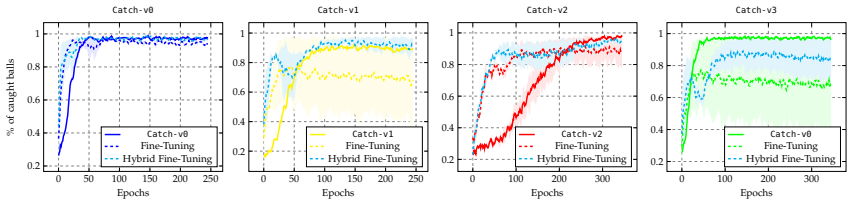


Figure 9.7: The performance (in cyan) of a fine-tuned pre-trained network whose last layer is initialized with parameters that yielded positive transfer, whereas its convolutional and fully conntected layers are initialized with parameters that yielded negative transfer.

9.5 RELATED WORK & CONCLUSION

The closest research to the one presented in this chapter is certainly that of Farebrother, Machado, and Bowling [55] and Tyo and Lipton [257] that we reviewed in Sec. 4.4.2 of Chapter 3. In fact, both studies investigated the generalization properties of Deep-Q Networks in a model-free DRL context. Our extensive experiments confirm some of the preliminary claims that were made by the former about the potentially poor TL properties of Deep-Q Networks, but contradict the study of the latter, who suggested that fine-tuning DRL agents results in positive transfer when moving from a simpler task to a harder. While both works are certainly valuable, we also believe that their experimental results are not as thorough and on par with the ones of this study as they both considered a very limited number of RL problems (four and three respectively), therefore leaving the question *"How transferable are Deep-Q Networks?"* unanswered. We believe that the answer to it is *"barely"*, but we also believe that their poor TL properties, as well as the learning dynamics identified in Sec. 9.4, are inherent to this family of algorithms only. In fact, it is worth noting that several works describing the benefits of TL in RL do exist (Sec. 4.3.2 of Chapter 4): Tirinzoni, Sanchez, and Restelli [253] show that it possible to successfully transfer value functions across tasks, yet their work does not consider deep networks as function approxima-

tors but rather Gaussian mixtures. Parisotto, Ba, and Salakhutdinov [174] show that it can be beneficial to fine-tune a pre-trained DRL agent, but they consider multi-task learning and policy gradient algorithms as a way of pre-training. Rusu et al. [201] also show that fine-tuning can be beneficial, but in the context of progressive networks and again of policy gradient techniques. Similar conclusions for Actor-Critic algorithms can also be found in the works of Zhu et al. [298] and Chen et al. [37]. Furthermore, Landolfi, Thomas, and Ma [121] and Sasso, Sabatelli, and Wiering [214] show that fine-tuning a pre-trained network can be beneficial for DRL tasks, but for model-based RL approaches, which are again part of a family of techniques that is different from the ones analyzed throughout this dissertation. To conclude, we would like to stress out that despite the overall poor TL performance observed throughout this chapter, positive transfer can nevertheless be obtained in a model-free DRL setup, and hope that this work can serve as a solid starting point for the DRL community which is interested in designing general and transferable agents.

10 CONCLUDING REMARKS

Outline

This is the concluding chapter of this dissertation which is divided in two parts: we start by answering the research questions that were presented at the beginning of this work. Each question is answered with respect to the research that has been presented throughout the earlier chapters of this work and is followed by a brief critical conclusion. We then move to the second part of this chapter, presented in Sec. 10.2, where we will critically analyze how over the last decade, the fields of supervised learning and reinforcement learning have been affected by the rise of deep neural networks and give our personal interpretation of what the future of both research fields could look like in the coming years.

10.1 ANSWERS TO THE ORIGINAL RESEARCH QUESTIONS

We now answer the research questions that we have introduced at the beginning of this manuscript and that have served as inspiration for all the work that has been presented throughout the thesis.

1. *Can convolutional neural networks be transferred and trained across different source and target domains? if so, which target domains could be of interest for investigating their transfer learning properties?*

   The research presented in Chapters 5, 6 and 7 shows that when it comes to supervised learning problems, convolutional neural networks exhibit strong transfer learning properties. In Chapter 5 we have in fact seen that five popular neural architectures, originally designed for tackling computer vision problems on datasets containing natural images, can be transferred for targetting classification problems that come from the field of digital heritage. Furthermore, we have also empirically shown that all such pre-trained architectures, are able to learn features on datasets of natural images that generalize to the non natural image domain. Similar conclusions can also be drawn from the results presented in Chapter 6, where we have shown that the good transfer learning properties of convolutional neural networks go beyond the computer vision task of classification, and also hold for object detection problems. Similarly to the research presented in Chapter 5 we have again considered the field of

digital heritage as target domain, as it offers numerous potentially interesting practical applications such as the one modeled by the MINERVA dataset. In Chapter 7 we have then seen that an equally important target domain for exploiting the transfer learning properties of pre-trained image classifiers is that of digital pathoology, as it is a field that is potentially characterized by a lack of appropriate training data, a hurdle that can strongly limit the training process of a convolutional neural network.

While all the research presented in the second part of this thesis provides strong evidence in favour of transferring, and potentially fine-tuning, pre-trained convolutional neural networks, the same can however not be said for the results obtained in Chapter 9, where we have instead seen that the transfer learning potential of such models can be much more limiting when it comes to the model-free deep reinforcement learning domain.

2. *What Transfer Learning training strategy should be adopted to maximize the performance of pre-trained networks?*

   As presented in Chapter 4, there are two main approaches for performing transfer learning in the context of convolutional neural networks: an off-the-shelf feature extraction approach, and a fully fine-tuning approach. The research presented in Chapter 5 clearly shows that when it comes to image classification problems, the latter training strategy results in significantly better final performance, as it allows networks to better adapt to the target domain, and therefore learn new feature representations that are relevant for the target task. The results of this study served as inspiration for the research presented in Chapter 6 where a fine-tuning training strategy was preferred over an off-the-shelf approach when tackling object detection problems and, in the end, resulted in models that were able to successfully detect musical instruments in paintings. Surprisingly, however, in Chapter 9 we have then seen that a fine-tuning transfer learning approach can be detrimental in the context of model-free deep reinforcement learning, as deep reinforcement learning agents transfer very poorly across tasks and even to themselfes. In the case of the latter, however, this only happens if a fine-tuning training strategy is adopted, and not if the networks are used as simple feature extractors.

   Despite the negative performance presented in Chapter 9 we overall still believe that if enough computational resources are available, pre-trained convolutional neural networks should always be fine-tuned, especially when it comes to supervised learning problems.

3. *Can Transfer Learning be a valuable tool for better understanding convolutional neural networks?*

Throughout this thesis we have argued that convolutional neural networks should be studied from a transfer learning perspective, not only because this would allow practictioners to know whether such algorithms could be deployed outside the realm of natural images, but also because their transfer learning properties could deliver novel insights into their inner properties. The research presented in Chapters 7 and 9 is a clear example that shows that a better understanding of convolutional neural networks can be obtained thanks to transfer learning. In Chapter 7 we have used transfer learning as a tool for better understanding the phenomenon of the Lottery Ticket Hypothesis (LTH). This allowed us to show that pruned convolutional neural networks winners of the LTH contain inductive biases that are generic at least to some extent, and therefore gain a deeper understanding of this deep learning phenomenon. In Chapter 9 we have instead discovered that convolutional neural networks transfer very poorly when they get trained for solving reinforcement learning tasks in a model-free deep reinforcement learning setting. With the intent of understanding why their transfer learning potential was not on par with the one that was extensively observed in Chapters 5 and 6, we have gained novel insights that allowed us to show that models commonly denoted as Deep-Q Networks go through two very distinct training phases.

We therefore believe that transfer learning is an extremely valuable tool for better understanding neural networks, as it allows to study their training process from a perspective which is unique and that goes beyond that of models that get trained from scratch.

4. *Do different machine learning paradigms result in convolutional neural networks with different transfer learning properties?*

   Based on the significant gap in terms of performance between the results obtained in the second part of this dissertation, and the results obtained in the third part of this thesis, we strongly believe that the answer to this research question is *yes*. Specifically, we have seen that as long as convolutional neural networks are trained for solving supervised learning tasks, then they will very likely exhibit strong transfer learning properties (see Chapters 5 and 6). However, if such models will be used for tackling deep reinforcement learning problems in a model-free reinforcemnt learning context, then their transfer learning potential will be much more limited. The reason of this is that when it comes to supervised learnig, convolutional neural networks will only have to serve as feature extractors, whereas the same cannot be said for model-free deep reinforcement learning,

where such models have also to serve as optimal value function approximators.

The transfer learning potential of convolutional networks is therefore highly dependant from the machine learning problem at hand, although we also believe that the poor transfer learning properties observed in Chapter 9 are inherent to model-free deep reinforcement learning algorithms only.

10.2 CRITICAL DISCUSSION & FUTURE PERSPECTIVES

We now present a critical discussion which addresses some of the limitations that currently characterize the fields of deep supervised and deep reinforcement learning and see how they relate to possible future work.

10.2.1 *Deep Supervised Learning*

Data and Computing Power Neural networks only require two simple ingredients for tackling most of supervised learning problems: 1) large amounts of training data and 2) enough computational resources. If both of such ingredients are available, we believe that most supervised learning problems within Computer Vision can be addressed successfully. While it is true that both ingredients can not always be available to machine learning practitioners, it is also true that as extensively demonstrated throughout the second part of this thesis, one could overcome their scarcity thanks to deep transfer learning. We therefore disagree with Marcus [146] who at the present moment considers the deep learning field as too data hungry, and believe that this is an issue that thanks to the availability of large pre-trained models has been successfully addressed.

Convolutional Neural Networks and Vision Transformers As extensively documented throughout this dissertation, when it comes to classification and regression problems that are characterized by high-dimensional and spatially organized inputs, convolutional neural networks have become the de facto neural architecture. Yet, the last years have seen the emergence of a novel type of neural architecture called Vision Transformers (ViTs) [51]. Next to obtaining overall excellent performance in the domain of Computer Vision, ViTs also appear to be requiring substantially fewer computational resources for training. It is, therefore, natural to wonder whether this family of models could in the future become as popular as convolutional neural networks are nowadays, and even if they could eventually replace convolutional architectures completely. We certainly think that ViTs will in the coming years gain in popularity, but also that their potential within Computer Vision will highly depend from the transfer

learning properties that these models will show (a research direction that at the present moment has not been thoroughly explored yet). We therefore believe that a replication of the studies presented in Chapters 5, 6 and 9 could certainly be of great interest to the Computer Vision and Reinforcement Learning communities, and recommend them as potential avenues for future work as recently explored by Liu et al. [142].

10.2.2 *Deep Reinforcement Learning*

Tedious Hyperparameters In Chapter 3 we have seen that Deep Reinforcement Learning (DRL) has gained a lot of attention from the machine learning community over the last decade, as the number of successful applications showcasing its potential have in fact become countless, ranging from agents achieving super-human performance on popular boardgames such as chess and go, to neural networks able to autonomously navigate stratospheric balloons. To an untrained eye, or more simply to a RL practitioner with few years of experience, successfully training a DRL agent might look like a straightforward task. However, behind the largely acclaimed accomplishments praised by the DRL literature, there is an equivalently large, and mostly hidden, process of hyperparameter tuning which is essential for successfully training a neural network. Throughout this thesis, convolutional neural networks have been trained both in a supervised learning context as well as in a reinforcement learning one, and we have in first person experienced how unrobust and oversensitive such algorithms can be when targeting the latter type of problems. This susceptability, on the contrary, was never encountered when tackling supervised learning tasks. We believe that at the present moment, despite all of its remarkable achievements, DRL cannot yet be considered as a successful application of convolutional neural networks. As long as a state of the art Rainbow agent [95] can only get successfully trained if its learning rate is set to the unintuitive value of 0.0000625, and a DRL practitioner has to wait for weeks before being able to see a DQN agent play certain games of the Atari Arcade Learning Environment [108], we believe that DRL is far from being solved.

The Deadly Triad In Chapters 3 and 8 we have mentioned the Deadly Triad of Deep Reinforcement Learning, a combination of three elements, which if present within the same learning agent can result into algorithms that diverge and are unable to learn an approximation of an optimal value function. As one of the elements of the Deadly Triad is that of a function approximator, it naturally follows that the stability of deep reinforcement learning algorithms will constantly be questioned by the deep learning community which will regularly introduce novel neural architectures in the coming years. This raises

the natural question *Which element of the triad should eventually be given up when developing DRL algorithms?*. So far, the community seems to agree that giving up on function approximators is clearly not possible as neural networks will always play a crucial role in the development of future DRL algorithms [56, 93, 260] thanks to their properties that we discussed in Sec. 3.6 of Chapter 3. We agree with this point of view and therefore believe that either off-policy learning or bootstrapping should be given up: if the end goal is that of developing algorithms that do not diverge whilst training we believe that it is the off-policy learning element of the triad which should be withdrawn during the development of novel DRL techniques. While this could come at the price of restricting the number of possible policies that could be learned by an agent, we believe that this is a problem which could be controlled as long as agents will be combined with proper exploration policies. The DQV-Learning algorithm presented in Chapter 8 is an example of a DRL algorithm that can avoid divergence through on-policy learning, while at the same time making use of a neural network trained through temporal-difference learning.

Part IV

APPENDIX

A

HOW TO IDENTIFY LOTTERY WINNERS

We hereafter report all the hyperparameters that have guided the research presented in Chapter 7 and that could be of interest for researchers interested in identifying pruned models winners of the Lottery Ticket Hypothesis. In all of our experiments we have used a ResNet-50 convolutional neural network which has the same structure as the one presented in [83]. We have chosen this specific architecture since it has proven to be successful both when used on DP data [164] as on DH datasets [206]. Specifically when it comes to the amount of strides, the sizes of the filters, and the number of output channels, the residual blocks of the network come in the following form: $(1 \times 1, 64, 64, 256) \times 3$, $(2 \times 2, 128, 128, 512) \times 4$, $(2 \times 2, 256, 256, 1024) \times 6$, $(2 \times 2, 512, 512, 2048) \times 3$. The last convolution operation of the network is followed by an average pooling layer and a final linear classification layer which has as many output nodes as there is classes to classify in our datasets. Since we only considered classification problems, the model always minimizes the categorical-crossentropy loss function. When feeding the model with the images of the datasets discussed in Chapter 7 we extract a random crop of size 224×224 and used mini-batches of size 64. No data-augmentation was used. We train the neural network with the Stochastic Gradient Descent (SGD) algorithm with an initial learning rate of 10^{-1}. SGD is used in combination with Nesterov Momentum ρ, set to 0.9, and a weight decay factor α set to 10^{-5}. Training is controlled by the early-stopping regularization method which stops the training process as soon as the validation loss does not decrease for five epochs in a row. When it comes to the parameters used for pruning we follow a magnitude pruning scheme as the one presented in [84] which has a pruning-rate of 20%. In order to construct winning-tickets we have used the late-resetting procedure with $k = 2$. We summarize all this information in Table A.1.

Hyperparameter	
Neural Network	ResNet-50
Weight-Initialization	Xavier
Optimizer	SGD
Size of the mini-batches	64
Learning-rate	10^{-1}
Momentum ρ	0.9
Decay-Factor α	10^{-5}
Annealing-epochs	$[50, 60, 75]$
Early-Stopping	5
Pruning-Rate	0.20
Late-resetting k	2

Table A.1: Hyperparameters for the experimental setup adopted in Chapter 7.

B

THE DEEP QUALITY-VALUE LEARNING ALGORITHMS

In this appendix, we report the pseudocode of the DQV-Learning and DQV-Max Learning algorithms described in Sec. 8.2 of Chapter 8. The DQV-Learning algorithm is also used for the transfer learning studies presented in Sec. 9.2 of Chapter 9.

In Tables B.1, B.2 and B.3 we also summarize all the hyper-parameters that we have used during the experiments reported in Chapters 8 and 9, ranging from the pre-processing values to the neural architectures. We have followed the typical experimental setup that is usually reported in the DRL literature [35, 161, 209, 270, 281] that trains Deep Q-Networks on the Atari 2600 testbed.

Table B.1: Hyper-parameters used in all our experiments that use the DQV and DQV-Max algorithms. All hyper-parameters coincide with the ones used by e.g. DQN and DDQN with the only difference being the epsilon greedy parameter ϵ that is set to 0.5 instead of 1.0. DQV-Learning is known to converge faster than these algorithms [210], since it is based on an on-policy learning algorithm [276]. Therefore it requires to explore less than typical off-policy models.

Hyperparameter	
Atari Arcade Learning Version	`Deterministic-v4`
Frame-Skipping	True
Reward Clipping	$[-1, 1]$
Epsilon Greedy ϵ	0.5
Discount Factor γ	0.99
Pre-processing scheme	$84 \times 84 \times 4$
Q-optimizer	RMSprop
Q Learning rate	0.00025
V-optimizer	SGD
V Learning rate	0.001
Optimizer ρ	0.95
Optimizer ϵ	0.01
Memory size S	1M trajectories

Algorithm 1 DQV and DQV-Max Learning

Require: Experience Replay Queue D of maximum size N
Require: Q network with parameters θ ▷ Network required by DQV
Require: V networks with parameters Φ and Φ^- ▷ Networks required by DQV
Require: Q networks with parameters θ and θ^- ▷ Networks required by DQV-Max
Require: V network with parameters Φ ▷ Network required by DQV-Max
Require: total_a = 0
Require: total_e = 0
Require: c = 10000
Require: $\mathcal{N} = 50000$
while True **do**
  set s_t as the initial state
  while s_t is not terminal **do**
    select $a_t \in \mathcal{A}$ for s_t with policy π (using the epsilon-greedy strategy)
    get r_t and s_{t+1}
    store $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in D
    $s_t := s_{t+1}$
    total_e += 1
    if total_e $\geqq \mathcal{N}$ **then**
      sample a minibatch $B = \{\langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle | i = 1, \ldots, 32\}$ of size 32 from D
      for i = 1 to 32 **do**
        if s_{t+1}^i is terminal **then**
          $y_t^i := r_t^i$ ▷ TD-Error for DQV
          $v_t^i := r_t^i$ ▷ 1st TD-Error for DQV-Max
          $q_t^i := r_t^i$ ▷ 2nd TD-Error for DQV-Max
        else
          $y_t^i := r_t^i + \gamma V(s_{t+1}^i, \Phi^-)$ ▷ TD-Error for DQV
          $v_t^i := r_t^i + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}^i, a, \theta^-)$ ▷ 1st TD-Error for DQV-Max
          $q_t^i := r_t^i + \gamma\, V(s_{t+1}^i, \Phi)$ ▷ 2nd TD-Error for DQV-Max
        end if
      end for
      $\theta := \arg\min_\theta \sum_{i=1}^{32} (y_t^i - Q(s_t^i, a_t^i, \theta))^2$ ▷ Train the Q network for DQV
      $\Phi := \arg\min_\Phi \sum_{i=1}^{32} (y_t^i - V(s_t^i, \Phi))^2$ ▷ Train the V network for DQV
      $\theta := \arg\min_\theta \sum_{i=1}^{32} (q_t^i - Q(s_t^i, a_t^i, \theta))^2$ ▷ Train the Q network for DQV-Max
      $\Phi := \arg\min_\Phi \sum_{i=1}^{32} (v_t^i - V(s_t^i, \Phi))^2$ ▷ Train the V network for DQV-Max
      total_a += 1
      if total_a = c **then**
        $\Phi^- := \Phi$ ▷ Update the target V network in DQV
        $\theta^- := \theta$ ▷ Update the target Q network in DQV-Max
        total_a := 0
      end if
    end if
  end while
end while

Table B.2: Architecture used by DQV-Learning for estimating the Q function, the n parameter in the last layer of the network changes with respect to the number of actions that is required by each different Atari game. Please note that this architecture corresponds to a typical Q network that is also used by popular algorithms like DQN [161], DDQN [281] and Rainbow [95]

Layer	Output Shape	Param
Conv2D	(None, 20, 20, 32)	8224
Conv2D	(None, 9, 9, 64)	32832
Conv2D	(None, 7, 7, 64)	36928
Flatten	(None, 3136)	0
Dense	(None, 512)	1606144
Dense	(None, n)	1539
Activations	ReLU	

Table B.3: Architecture used for estimating the V function. This architecture corresponds exactly to the one presented in Table B.2 with the only difference being the last output layer which in this case is set to one.

Layer	Output Shape	Param
Conv2D	(None, 20, 20, 32)	8224
Conv2D	(None, 9, 9, 64)	32832
Conv2D	(None, 7, 7, 64)	36928
Flatten	(None, 3136)	0
Dense	(None, 512)	1606144
Dense	(None, 1)	1539
Activations	ReLU	

C

REINFORCEMENT LEARNING UPSIDE DOWN

In this appendix, we report some preliminary experiments that we have performed in the context of Upside-Down Reinforcement Learning and reflect on the potential that this learning paradigm can offer to the reinforcement learning community.

Despite many successes, it is clear that the combination of reinforcement learning algorithms and deep neural networks can present some significant limitations. Throughout this dissertation we have mainly focused on the aforementioned Deadly Triad phenomenon and on the poor transfer learning properties of Deep-Q Networks which both can limit the development of robust and general agents. Next to these issues, DRL algorithms are characterized by numerous other problems as well (the discussion of which is out of the scope of this thesis) and that all mainly arise because tabular RL algorithms are combined with function approximators [8, 64, 65, 120, 148, 256, 261]. Although the DRL community has been able to address most of such issues successfully, we believe that the way optimal control problems are currently being solved by the DRL community might need revision. In fact, we argue that most of the current DRL research focuses on introducing solutions that, albeit valuable, are only able to solve very specific and limited problems to a (sometimes very) small extent. As a result, very few DRL practitioners have been critically questioning the long term value of modern DRL algorithms, and the way RL problems are currently being addressed. An exception to this common trend, however, has recently been introduced by Schmidhuber [217] who proposed the idea of "Upside-Down Reinforcement Learning" (UDRL). In UDRL the main goal is to solve typical Markovian control problems via classic supervised learning techniques, and to replace common RL concepts such as value function approximation and policy search, through maximum likelihood estimation techniques. In principle this should be done by learning a mapping from states to actions (see [233] for all the mathematical details), which could overcome the need of learning an optimal value function or stochastic policy. We agree with Schmidhuber's ideas, and support the goal of potentially solving optimal control problems via supervised learning techniques. In fact we strongly believe that DRL in its current form is already reducing reinforcement learning problems to supervised learning problems. As an example let us consider the role of the experience replay memory buffer reviewed in Chapter 3 and adopted by the DQV and DQV-Max Learning algorithms presented in Chapter 8. We have seen that the role of experience replay is that

of storing RL trajectories which can in a later moment be sampled and used for minimizing the objective function of a Deep-Q Network. By taking a critical look at this buffer, we can see that it is in large part equivalent to the datasets that are typically used in supervised learning: it needs to store a large amount of RL trajectories which can then be used for modeling the task of learning a value function as a regression problem where $\mathcal{Y}$ is modeled by the space of all TD-errors stored within the buffer. The idea of constructing a dataset of previous trajectories is very far from the ideal RL setting reviewed at the beginning of Chapter 3, where we have indeed seen that a RL agent should be able to learn in an online fashion based on the last trajectory only. We therefore believe that the successess obtained by DRL stem from the fact that RL problems have been modeled as supervised learning ones, which as discussed in Sec. 10.2.1 are well suited for deep neural networks.

We have experimented with some of the UDRL ideas presented in [217, 233] and compared the performance of a preliminary version of an UDRL agent to that of a DQN, DDQN and DQV agent on two different benchmarks: the popular control problem `Cartpole` and the Atari game `Pong`. We report the results of these very preliminary experiments in Fig. C.1. We can see that on the `Cartpole` benchmark the UDRL learns significantly better than all three model-free DRL algorithms, but is unable to improve its policy over time at all when it comes to the more complicated `Pong` game. These results show that UDRL has definitely some potential, although furhter research will be required to investigate whether this type of algorithm can scale to more complicated problems. We believe that if such scaling issues will be solved, then UDRL could become a valid alternative to popular DRL algorithms. More specifically we foresee that in the future UDRL will find a successful application in the following three RL areas: [1].

- Transfer Learning: as neural networks will not have to jointly serve as optimal value function approximators and feature extractors, we believe that UDRL agents should in principle be more suitable for transfer learning. In fact such models will not have to go through the training dynamics that we have identified in Chapter 9, and will in essence be more similar to the type of models that we have successfully transferred throughout the second part of this dissertation (as they will be trained according to supervised learning principles).

- Batch Reinforcement Learning: a common problem of current DRL algorithms is that they suffer from a phenomenon known

1 Please note that these claims are at the present moment not empirically supported in any way, and are made on top of some intuition that has been built after performing the preliminary experiments presented in Fig. C.1.

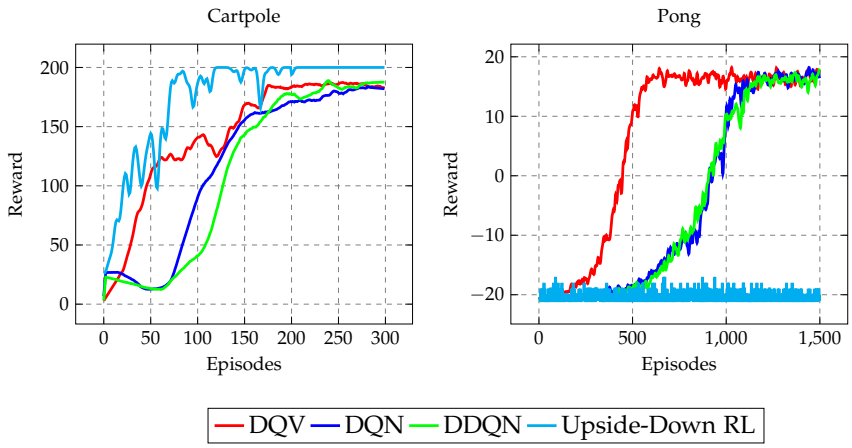


Figure C.1: Two examples of an UDRL agent described by Schmidhuber [217]. When combined with a multi-layer perceptron the agent significantly outperforms the DQV, DQN and DDQN agents on the Cartpole environment (left figure). However, it is unable to improve its policy over time at all when trained on the Pong game and combined with a convolutional neural network (right figure).

as "extrapolation error" [64]. When such agents are trained in a batch setting, they fail in improving their policy because the trajectories contained within the experience replay memory buffer have been collected by a different agent. This can pose numerous practical issues as the process of collecting RL trajectories can sometimes be particular expensive, therefore limiting the potential interaction between the agent and its environment. As UDRL technically overcomes the need of learning an optimal value function, it follows that such agents should not suffer from the extrapolation bias and could therefore generalize well in a batch RL set-up.

- Interpretable Reinforcement Learning: the main training principle of UDRL agents of maximum likelihood estimation opens the door to the use of supervised learning algorithms other than deep neural networks. Deep neural networks are often considered to be black boxes and highly uninterpretable models, which is a quality that when it comes to optimal decision making problems could be desirable [166]. However, supervised learning algorithms that are largely more interpretable than neural networks do exist [27], and we therefore foresee that these could be used in combination with UDRL in situations where it is desirable to interpret the decisions taken by an agent.

BIBLIOGRAPHY

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

[2] Joshua Achiam, Ethan Knight, and Pieter Abbeel. "Towards Characterizing Divergence in Deep Q-Learning." In: *arXiv preprint arXiv:1903.08894* (2019).

[3] Sandro Ackermann, Kevin Schawinksi, Ce Zhang, Anna K Weigel, and M Dennis Turp. "Using transfer learning to detect galaxy mergers." In: *Monthly Notices of the Royal Astronomical Society* (2018).

[4] Charu C Aggarwal et al. *Neural networks and deep learning*. Springer, 2018.

[5] Samy Aittahar, Raphaël Fonteneau, and Damien Ernst. "Empirical Analysis of Policy Gradient Algorithms where Starting States are Sampled accordingly to Most Frequently Visited States." In: *IFAC-PapersOnLine* 53.2 (2020), pp. 8097–8104.

[6] Nancy Allen. "Collaboration through the Colorado digitization project." In: *First Monday* 5.6 (2000).

[7] Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. "Evaluating saliency map explanations for convolutional neural networks: a user study." In: *Proceedings of the 25th International Conference on Intelligent User Interfaces*. 2020, pp. 275–285.

[8] Oron Anschel, Nir Baram, and Nahum Shimkin. "Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning." In: *International conference on machine learning*. PMLR. 2017, pp. 176–185.

[9] Andrew Arnold, Ramesh Nallapati, and William W Cohen. "A comparative study of methods for transductive transfer learning." In: *Seventh IEEE international conference on data mining workshops (ICDMW 2007)*. IEEE. 2007, pp. 77–82.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep reinforcement learning: A brief survey." In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[11] Nishanth Arun, Nathan Gaw, Praveer Singh, Ken Chang, Mehak Aggarwal, Bryan Chen, Katharina Hoebel, Sharut Gupta, Jay Patel, Mishka Gidwani, et al. "Assessing the (un) trustworthiness of saliency maps for localizing abnormalities in medical imaging." In: *arXiv preprint arXiv:2008.02766* (2020).

[12] Nikolay Banar, Matthia Sabatelli, Pierre Geurts, Walter Daelemans, and Mike Kestemont. "Transfer Learning with Style Transfer between the Photorealis-tic and Artistic Domain." In: (2021).

[13] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. "Successor features for transfer in reinforcement learning." In: *Advances in neural information processing systems*. 2017, pp. 4055–4065.

[14] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[15] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic differentiation in machine learning: a survey." In: *Journal of machine learning research* 18 (2018).

[16] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 449–458.

[17] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. "The arcade learning environment: An evaluation platform for general agents." In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[18] Richard Bellman. "Dynamic programming." In: *Science* 153.3731 (1966), pp. 34–37.

[19] Dimitri P Bertsekas. "Value and policy iterations in optimal control and adaptive dynamic programming." In: *IEEE transactions on neural networks and learning systems* 28.3 (2015), pp. 500–509.

[20] Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

[21] Dimitri P Bertsekas and John N Tsitsiklis. "Neuro-dynamic programming: an overview." In: *Proceedings of 1995 34th IEEE conference on decision and control*. Vol. 1. IEEE. 1995, pp. 560–564.

[22] Dimitri P Bertsekas et al. *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

[23] Francesco Bidoia, Matthia Sabatelli, Amirhossein Shantia, Marco A. Wiering, and Lambert Schomaker. "A Deep Convolutional Neural Network for Location Recognition and Geometry based Information." In: *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2018, Funchal, Madeira - Portugal, January 16-18, 2018*. 2018, pp. 27–36.

[24] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry Jackel, Urs Muller, and Karol Zieba. "VisualBackProp: efficient visualization of CNNs." In: *arXiv preprint arXiv:1611.05418* (2016).

[25] Léon Bottou and Olivier Bousquet. "13 the tradeoffs of large-scale learning." In: *Optimization for machine learning* (2011), p. 351.

[26] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. "Audio Chord Recognition with Recurrent Neural Networks." In: *ISMIR*. Citeseer. 2013, pp. 335–340.

[27] Leo Breiman. "Random forests." In: *Machine learning* 45.1 (2001), pp. 5–32.

[28] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym." In: *arXiv preprint arXiv:1606.01540* (2016).

[29] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners." In: *arXiv preprint arXiv:2005.14165* (2020).

[30] Arthur E Bryson. "Applied optimal control: Optimization." In: *Estimization and Control* 2 (1975).

[31] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. Vol. 39. CRC press, 2010.

[32] Lucian Buşoniu, Damien Ernst, Bart De Schutter, and Robert Babuška. "Approximate reinforcement learning: An overview." In: *2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE. 2011, pp. 1–8.

[33] Rich Caruana. "Multitask learning." In: *Machine learning* 28.1 (1997), pp. 41–75.

[34] Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping." In: *Advances in neural information processing systems*. 2001, pp. 402–408.

[35] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. "Dopamine: A research framework for deep reinforcement learning." In: *arXiv preprint arXiv:1812.06110* (2018).

[36] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks." In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 839–847.

[37] Lili Chen, Kimin Lee, Aravind Srinivas, and Pieter Abbeel. "Improving Computational Efficiency in Visual Reinforcement Learning via Stored Embeddings." In: *arXiv preprint arXiv:2103.02886* (2021).

[38] François Chollet. "Xception: Deep learning with depthwise separable convolutions." In: *arXiv preprint* (2016).

[39] François Chollet et al. *Keras*. 2015.

[40] Yann Claes et al. "Deep Learning for the Classification and Detection of Animals in Artworks." In: (2021).

[41] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. "Deep learning for classical japanese literature." In: *arXiv preprint arXiv:1812.01718* (2018).

[42] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)." In: *arXiv preprint arXiv:1511.07289* (2015).

[43] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. "Self-taught clustering." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 200–207.

[44] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection." In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[45] Li Deng, Geoffrey Hinton, and Brian Kingsbury. "New types of deep neural network learning for speech recognition and related applications: An overview." In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8599–8603.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[47] Emily I Dolan. *MIMO: Musical Instrument Museums Online*. 2017.

[48] H Domínguez Sánchez, M Huertas-Company, M Bernardi, S Kaviraj, JL Fischer, TMC Abbott, FB Abdalla, J Annis, S Avila, D Brooks, et al. "Transfer learning for galaxy morphology from one survey to another." In: *Monthly Notices of the Royal Astronomical Society* 484.1 (2019), pp. 93–100.

[49] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "Decaf: A deep convolutional activation feature for generic visual recognition." In: *International conference on machine learning*. PMLR. 2014, pp. 647–655.

[50] Xin Dong, Shangyu Chen, and Sinno Pan. "Learning to prune deep neural networks via layer-wise optimal brain surgeon." In: *Advances in Neural Information Processing Systems*. 2017, pp. 4857–4867.

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

[52] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[53] Stamatatos Efstathios. "A survey of modern authorship attribution methods." In: *Journal of the American Society for Information Science and Technology* 3 (2009), pp. 538–556. DOI: 10.1002/asi.21001.

[54] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge." In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[55] Jesse Farebrother, Marlos C Machado, and Michael Bowling. "Generalization and regularization in DQN." In: *arXiv preprint arXiv:1810.00123* (2018).

[56] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. "Revisiting fundamentals of experience replay." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3061–3071.

[57] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. "Noisy networks for exploration." In: *arXiv preprint arXiv:1706.10295* (2017).

[58] Vincent François-Lavet, Raphaël Fonteneau, and Damien Ernst. "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies." In: *NIPS 2015 Workshop on Deep Reinforcement Learning*. 2015.

[59] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. "An introduction to deep reinforcement learning." In: *arXiv preprint arXiv:1811.12560* (2018).

[60] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks." In: *arXiv preprint arXiv:1803.03635* (2018).

[61] Jonathan Frankle, G Karolina Dziugaite, DM Roy, and M Carbin. "Stabilizing the Lottery Ticket Hypothesis." In: *arXiv preprint arXiv:1903.01611* (2019).

[62] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. "Linear mode connectivity and the lottery ticket hypothesis." In: *arXiv preprint arXiv:1912.05671* (2019).

[63] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[64] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. "Benchmarking Batch Deep Reinforcement Learning Algorithms." In: *arXiv preprint arXiv:1910.01708* (2019).

[65] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[66] Matthieu Geist, Bruno Scherrer, et al. "Off-policy learning with eligibility traces: a survey." In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 289–333.

[67] Daniel George, Hongyu Shen, and EA Huerta. "Deep Transfer Learning: A new deep learning glitch classification method for advanced LIGO." In: *arXiv preprint arXiv:1706.07446* (2017).

[68] Pierre Geurts. "Contributions to decision tree induction: bias/-variance tradeoff and time series classification." PhD thesis. ULiège-University of Liège, 2002.

[69] Ross Girshick. "Fast r-cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[70] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[71] Varun Gohil, S Deepak Narayanan, and Atishay Jain. "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers." In: *ReScience-C* (2020).

[72] Nicolas Gonthier, Yann Gousseau, Said Ladjal, and Olivier Bonfait. "Weakly supervised object detection in artworks." In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0.

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

[74] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks." In: *International conference on machine learning*. PMLR. 2013, pp. 1319–1327.

[75] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 369–376.

[76] Alan Green and Sean Ferguson. "RIdIM: cataloguing music iconography since 1971." In: *Fontes artis musicae* (2013), pp. 1–8.

[77] David Ha and Jürgen Schmidhuber. "World models." In: *arXiv preprint arXiv:1803.10122* (2018).

[78] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.

[79] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. "Dream to control: Learning behaviors by latent imagination." In: *arXiv preprint arXiv:1912.01603* (2019).

[80] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[81] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. "Mastering atari with discrete world models." In: *arXiv preprint arXiv:2010.02193* (2020).

[82] Travis Hammond, Dirk Jelle Schaap, Matthia Sabatelli, and Marco A Wiering. "Forest Fire Control with Learning from Demonstration and Reinforcement Learning." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[83] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *arXiv preprint arXiv:1510.00149* (2015).

[84] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both weights and connections for efficient neural network." In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.

[85] Hado Philip van Hasselt. *Insights in reinforcement rearning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. Utrecht University, 2011.

[86] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. "Deep reinforcement learning with a natural language action space." In: *arXiv preprint arXiv:1511.04636* (2015).

[87] Kaiming He, Ross Girshick, and Piotr Dollár. "Rethinking imagenet pre-training." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4918–4927.

[88] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

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

[91] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep reinforcement learning that matters." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[92] Kevin Hernandez-Diaz, Fernando Alonso-Fernandez, and Josef Bigun. "Periocular recognition using CNN features off-the-shelf." In: *2018 International conference of the biometrics special interest group (BIOSIG)*. IEEE. 2018, pp. 1–5.

[93] J Fernando Hernandez-Garcia and Richard S Sutton. "Understanding Multi-Step Deep Reinforcement Learning: A Systematic Study of the DQN Target." In: *arXiv preprint arXiv:1901.07510* (2019).

[94] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. "On inductive biases in deep reinforcement learning." In: *arXiv preprint arXiv:1907.02908* (2019).

[95] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining improvements in deep reinforcement learning." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[96] Namgyu Ho and Yoon-Chul Kim. "Evaluation of transfer learning in deep convolutional neural network models for cardiac short axis slice classification." In: *Scientific reports* 11.1 (2021), pp. 1–11.

[97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[98] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799.

[99] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. "Searching for mobilenetv3." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324.

[100] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification." In: *arXiv preprint arXiv:1801.06146* (2018).

[101] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2. 2017.

[102] Rachel Huang, Jonathan Pedoeem, and Cuixian Chen. "YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers." In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 2503–2510.

[103] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. "What makes ImageNet good for transfer learning?" In: *arXiv preprint arXiv:1608.08614* (2016).

[104] Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. "Reinforcement learning algorithm for partially observable Markov decision problems." In: *Advances in neural information processing systems* (1995), pp. 345–352.

[105] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. "A survey of deep learning-based object detection." In: *IEEE access* 7 (2019), pp. 128837–128868.

[106] Ou Jin, Nathan N Liu, Kai Zhao, Yong Yu, and Qiang Yang. "Transferring topical knowledge from auxiliary long texts for short text clustering." In: *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2011, pp. 775–784.

[107] Philipp Kainz, Harald Burgsteiner, Martin Asslaber, and Helmut Ahammer. "Training echo state networks for rotation-invariant bone marrow cell classification." In: *Neural Computing and Applications* 28.6 (2017), pp. 1277–1292.

[108] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. "Model-based reinforcement learning for atari." In: *arXiv preprint arXiv:1903.00374* (2019).

[109] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation." In: *arXiv preprint arXiv:1806.10293* (2018).

[110] Ibrahem Kandel and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset." In: *ICT express* 6.4 (2020), pp. 312–315.

[111] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima." In: *arXiv preprint arXiv:1609.04836* (2016).

[112] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. "Deepmellow: removing the need for a target network in deep Q-learning." In: *Proceedings of the Twenty Eighth International Joint Conference on Artificial Intelligence*. 2019.

[113] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[114] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. "Overcoming catastrophic forgetting in neural networks." In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[115] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. "Panns: Large-scale pretrained audio neural networks for audio pattern recognition." In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), pp. 2880–2894.

[116] George Konidaris and Andrew Barto. "Autonomous shaping: Knowledge transfer in reinforcement learning." In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 489–496.

[117] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do Better ImageNet Models Transfer Better?" In: *arXiv preprint arXiv:1805.08974* (2018).

[118] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do better imagenet models transfer better?" In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2661–2671.

[119] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[120] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. "Stabilizing off-policy q-learning via bootstrapping error reduction." In: *arXiv preprint arXiv:1906.00949* (2019).

[121] Nicholas C Landolfi, Garrett Thomas, and Tengyu Ma. "A Model-based Approach for Sample-efficient Multi-task Reinforcement Learning." In: *arXiv preprint arXiv:1907.04964* (2019).

[122] Stephen H Lane, David A Handelman, and Jack J Gelfand. "Theory and development of higher-order CMAC neural networks." In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 23–30.

[123] Pat Langley. "Transfer of knowledge in cognitive systems." In: *Talk, workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning*. 2006.

[124] Romain Laroche and Merwan Barlier. "Transfer reinforcement learning with shared dynamics." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[125] Alessandro Lazaric. "Transfer in reinforcement learning: a framework and a survey." In: *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[126] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Reinforcement learning in continuous action spaces through sequential monte carlo methods." In: *Advances in neural information processing systems* 20 (2007), pp. 833–840.

[127] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Transfer of samples in batch reinforcement learning." In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 544–551.

[128] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[129] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[130] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/, 1994. 1996.

[131] Donghun Lee, Boris Defourny, and Warren B Powell. "Bias-corrected q-learning to control max-operator bias in q-learning." In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2013, pp. 93–99.

[132] Donghun Lee and Warren B Powell. "Bias-corrected Q-learning with multistate extension." In: *IEEE Transactions on Automatic Control* 64.10 (2019), pp. 4011–4023.

[133] Pascal Leroy, Damien Ernst, Pierre Geurts, Gilles Louppe, Jonathan Pisane, and Matthia Sabatelli. "QVMix and QVMix-Max: Extending the Deep Quality-Value Family of Algorithms to Cooperative Multi-Agent Reinforcement Learning." In: *Proceedings of the AAAI-21 Workshop on Reinforcement Learning in Games*, p. 8.

[134] Yuxi Li. "Deep reinforcement learning: An overview." In: *arXiv preprint arXiv:1701.07274* (2017).

[135] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015).

[136] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. "Runtime neural pruning." In: *Advances in Neural Information Processing Systems*. 2017, pp. 2181–2191.

[137] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching." In: *Machine learning* 8.3-4 (1992), pp. 293–321.

[138] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[139] Seppo Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors." In: *Master's Thesis (in Finnish), Univ. Helsinki* (1970), pp. 6–7.

[140] Jun S Liu and Rong Chen. "Sequential Monte Carlo methods for dynamic systems." In: *Journal of the American statistical association* 93.443 (1998), pp. 1032–1044.

[141] Libin Liu and Jessica Hodgins. "Learning to schedule control fragments for physics-based characters using deep q-learning." In: *ACM Transactions on Graphics (TOG)* 36.3 (2017), pp. 1–14.

[142] Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco Nadai. "Efficient Training of Visual Transformers with Small Datasets." In: *Advances in Neural Information Processing Systems* 34 (2021).

[143] Gilles Louppe. "Understanding random forests: From theory to practice." In: *arXiv preprint arXiv:1407.7502* (2014).

[144] David G Lowe. "Distinctive image features from scale-invariant keypoints." In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[145] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models." In: *Proc. icml*. Vol. 30. 1. Citeseer. 2013, p. 3.

[146] Gary Marcus. "Deep learning: A critical appraisal." In: *arXiv preprint arXiv:1801.00631* (2018).

[147] Raphaël Marée, Loïc Rollus, Benjamin Stévens, Renaud Hoyoux, Gilles Louppe, Rémy Vandaele, Jean-Michel Begon, Philipp Kainz, Pierre Geurts, and Louis Wehenkel. "Collaborative analysis of multi-gigapixel imaging data using Cytomine." In: *Bioinformatics* 32.9 (2016), pp. 1395–1401.

[148] Henrik Marklund, Suraj Nair, and Chelsea Finn. "Exact (Then Approximate) Dynamic Programming for Deep Reinforcement Learning." In: ().

[149] Dominic Masters and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks." In: *arXiv preprint arXiv:1804.07612* (2018).

[150] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[151] Neville Mehta, Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. "Transfer in variable-reward hierarchical reinforcement learning." In: *Machine Learning* 73.3 (2008), p. 289.

[152] Rahul Mehta. "Sparse Transfer Learning via Winning Lottery Tickets." In: *arXiv preprint arXiv:1905.07785* (2019).

[153] Thomas Mensink, Jasper Uijlings, Alina Kuznetsova, Michael Gygli, and Vittorio Ferrari. "Factors of Influence for Transfer Learning across Diverse Appearance Domains and Task Types." In: *arXiv preprint arXiv:2103.13318* (2021).

[154] Thomas Mensink and Jan Van Gemert. "The rijksmuseum challenge: Museum-centered visual recognition." In: *Proceedings of International Conference on Multimedia Retrieval*. 2014, pp. 451–454.

[155] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013).

[156] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[157] Tom M Mitchell et al. "Machine learning." In: (1997).

[158] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*. 2016, pp. 1928–1937.

[159] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. "Recurrent models of visual attention." In: *Advances in neural information processing systems*. 2014, pp. 2204–2212.

[160] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).

[161] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), p. 529.

[162] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. "Pruning convolutional neural networks for resource efficient inference." In: *arXiv preprint arXiv:1611.06440* (2016).

[163] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers." In: *Advances in Neural Information Processing Systems*. 2019, pp. 4933–4943.

[164] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Comparison of deep transfer learning strategies for digital pathology." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 2262–2271.

[165] Romain Mormont, Pierre Geurts, and Raphaël Marée. "Multi-task pre-training of deep neural networks for digital pathology." In: *IEEE journal of biomedical and health informatics* (2020).

[166] Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J Rezende. "Towards interpretable reinforcement learning using attention augmented agents." In: *arXiv preprint arXiv:1906.02500* (2019).

[167] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. "Language understanding for text-based games using deep reinforcement learning." In: *arXiv preprint arXiv:1506.08941* (2015).

[168] Yurii E Nesterov. "A method for solving the convex programming problem with convergence rate O (1/kˆ 2)." In: *Dokl. akad. nauk Sssr*. Vol. 269. 1983, pp. 543–547.

[169] Kien Nguyen, Clinton Fookes, Arun Ross, and Sridha Sridharan. "Iris recognition with off-the-shelf CNN features: A deep learning perspective." In: *IEEE Access* 6 (2017), pp. 18848–18855.

[170] Johan S Obando-Ceron and Pablo Samuel Castro. "Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research." In: *arXiv preprint arXiv:2011.14826* (2020).

[171] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. "Learning and transferring mid-level image representations using convolutional neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1717–1724.

[172] Michela Paganini and Jessica Zosa Forde. "Bespoke vs. Prêt-à-Porter Lottery Tickets: Exploiting Mask Similarity for Trainable Sub-Network Finding." In: *arXiv preprint arXiv:2007.04091* (2020).

[173] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning." In: *IEEE Transactions on knowledge and data engineering* 22.10 (2009), pp. 1345–1359.

[174] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning." In: *arXiv preprint arXiv:1511.06342* (2015).

[175] Jooyoung Park and Irwin W Sandberg. "Approximation and radial-basis-function networks." In: *Neural computation* 5.2 (1993), pp. 305–316.

[176] Ross Parry. "Digital heritage and the rise of theory in museum computing." In: *Museum management and Curatorship* (2005), pp. 333–348.

[177] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in pytorch." In: (2017).

[178] Devdhar Patel, Hananel Hazan, Daniel J Saunders, Hava T Siegelmann, and Robert Kozma. "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to Atari Breakout game." In: *Neural Networks* 120 (2019), pp. 108–115.

[179] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python." In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[180] Jing Peng and Ronald J Williams. "Incremental multi-step Q-learning." In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 226–232.

[181] Andreas Pentaliotis and Marco Wiering. "Variation-resistant Q-learning: Controlling and Utilizing Estimation Bias in Reinforcement Learning for Better Performance." In: (2021).

[182] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." In: *arXiv preprint arXiv:1802.05365* (2018).

[183] Matthew E Peters, Sebastian Ruder, and Noah A Smith. "To tune or not to tune? adapting pretrained representations to diverse tasks." In: *arXiv preprint arXiv:1903.05987* (2019).

[184] Fred Phillips and Brandy Mackintosh. "Wiki Art Gallery, Inc.: A case for critical thinking." In: *Issues in Accounting Education* 26.3 (2011), pp. 593–608.

[185] Alexandre Piché, Joseph Marino, Gian Maria Marconi, Christopher Pal, and Mohammad Emtiyaz Khan. "Beyond Target Networks: Improving Deep Q-learning with Functional Regularization." In: *arXiv preprint arXiv:2106.02613* (2021).

[186] Martin L Puterman. "Markov decision processes." In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[187] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[188] Pengjiang Qian, Yizhang Jiang, Zhaohong Deng, Lingzhi Hu, Shouwei Sun, Shitong Wang, and Raymond F Muzic. "Cluster prototypes and fuzzy memberships jointly leveraged cross-domain maximum entropy clustering." In: *IEEE transactions on cybernetics* 46.1 (2015), pp. 181–193.

[189] Pavlo M Radiuk et al. "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets." In: *Information Technology and Management Science* 20.1 (2017), pp. 20–24.

[190] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. "Continuous state-space models for optimal sepsis treatment: a deep reinforcement learning approach." In: *Machine Learning for Healthcare Conference*. PMLR. 2017, pp. 147–163.

[191] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[192] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[193] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement." In: *arXiv preprint arXiv:1804.02767* (2018).

[194] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems* 28 (2015), pp. 91–99.

[195] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[196] Corby Rosset, Chenyan Xiong, Minh Phan, Xia Song, Paul Bennett, and Saurabh Tiwary. "Knowledge-Aware Language Model Pretraining." In: *arXiv preprint arXiv:2007.00655* (2020).

[197] Sebastian Ruder. "An overview of gradient descent optimization algorithms." In: *arXiv preprint arXiv:1609.04747* (2016).

[198] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *nature* 323.6088 (1986), pp. 533–536.

[199] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[200] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[201] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. "Progressive neural networks." In: *arXiv preprint arXiv:1606.04671* (2016).

[202] Matthia Sabatelli. *Learning to Play Chess with Minimal Lookahead and Deep Value Neural Networks*. Tech. rep. Faculty of Science and Engineering, 2017.

[203] Matthia Sabatelli, Nikolay Banar, Marie Cocriamont, Eva Coudyzer, Karine Lasaracina, Walter Daelemans, Pierre Geurts, and Mike Kestemont. "Advances in Digital Music Iconography: Benchmarking the detection of musical instruments in unrestricted, non-photorealistic images from the artistic domain." In: *DHQ: Digital Humanities Quarterly* 15.1 (2021).

[204] Matthia Sabatelli, Francesco Bidoia, Valeriu Codreanu, and Marco Wiering. "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead." In: *7th International Conference on Pattern Recognition Applications and Methods*. 2018.

[205] Matthia Sabatelli and Pierre Geurts. "On The Transferability of Deep-Q Networks." In: *Deep Reinforcement Learning Workshop of the 35th Conference on Neural Information Processing Systems*. 2021.

[206] Matthia Sabatelli, Mike Kestemont, Walter Daelemans, and Pierre Geurts. "Deep transfer learning for art classification problems." In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 631–646.

[207] Matthia Sabatelli, Mike Kestemont, and Pierre Geurts. "On the transferability of winning tickets in non-natural image datasets." In: (2021), pp. 59–69.

[208] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. "Approximating two value functions instead of one: towards characterizing a new family of Deep Reinforcement Learning algorithms." In: *arXiv preprint arXiv:1909.01779* (2019).

[209] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A Wiering. "The deep quality-value family of deep reinforcement learning algorithms." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[210] Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco Wiering. "Deep Quality-Value (DQV) Learning." In: *Advances in Neural Information Processing Systems, Deep Reinforcement Learning Workshop*. Montreal. 2018.

[211] Iman Sajedian, Heon Lee, and Junsuk Rho. "Design of high transmission color filters for solar cells directed by deep Q-learning." In: *Solar Energy* 195 (2020), pp. 670–676.

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

[213] Adriel Saporta, Xiaotong Gui, Ashwin Agrawal, Anuj Pareek, Steven QH Truong, Chanh DT Nguyen, Van-Doan Ngo, Jayne Seekins, Francis G Blankenberg, Andrew Ng, et al. "Deep learning saliency maps do not accurately highlight diagnostically relevant regions for medical image interpretation." In: *medRxiv* (2021).

[214] Remo Sasso, Matthia Sabatelli, and Marco A Wiering. "Fractional Transfer Learning for Deep Model-Based Reinforcement Learning." In: *arXiv preprint arXiv:2108.06526* (2021).

[215] Stefan Schaal, Auke Jan Ijspeert, Aude Billard, Sethu Vijayakumar, and Jean-Arcady Meyer. "Estimating future reward in reinforcement learning animats using associative learning." In: (2004).

[216] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (2015).

[217] Juergen Schmidhuber. "Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions." In: *arXiv preprint arXiv:1912.02875* (2019).

[218] Robin M Schmidt, Frank Schneider, and Philipp Hennig. "Descending through a Crowded Valley–Benchmarking Deep Learning Optimizers." In: *arXiv preprint arXiv:2007.01547* (2020).

[219] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[220] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation." In: *arXiv preprint arXiv:1506.02438* (2015).

[221] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[222] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[223] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "Overfeat: Integrated recognition, localization and detection using convolutional networks." In: *arXiv preprint arXiv:1312.6229* (2013).

[224] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[225] Arjun Sharma et al. "Adapting off-the-shelf CNNs for word spotting & recognition." In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. 2015, pp. 986–990.

[226] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." In: *nature* 550.7676 (2017), pp. 354–359.

[227] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *arXiv preprint arXiv:1312.6034* (2013).

[228] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[229] Satinder Singh, Andrew G Barto, and Nuttapong Chentanez. *Intrinsically motivated reinforcement learning*. Tech. rep. MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE, 2005.

[230] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. "Convergence results for single-step on-policy reinforcement-learning algorithms." In: *Machine learning* 38.3 (2000), pp. 287–308.

[231] James E Smith and Robert L Winkler. "The optimizer's curse: Skepticism and postdecision surprise in decision analysis." In: *Management Science* 52.3 (2006), pp. 311–322.

[232] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[233] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. "Training agents using upside-down reinforcement learning." In: *arXiv preprint arXiv:1912.02877* (2019).

[234] Eric Van den Steen. "Rational overoptimism (and other biases)." In: *American Economic Review* 94.4 (2004), pp. 1141–1151.

[235] Gjorgji Strezoski and Marcel Worring. "Omniart: a large-scale artistic benchmark." In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14.4 (2018), pp. 1–21.

[236] Tianxiang Sun, Yunfan Shao, Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu, and Xuanjing Huang. "Learning Sparse Sharing Architectures for Multiple Tasks." In: *arXiv preprint arXiv:1911.05034* (2019).

[237] Richard S Sutton. "Learning to predict by the methods of temporal differences." In: *Machine learning* 3.1 (1988), pp. 9–44.

[238] Richard S Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding." In: *Advances in neural information processing systems*. 1996, pp. 1038–1044.

[239] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[240] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPs*. Vol. 99. Citeseer. 1999, pp. 1057–1063.

[241] Richard Stuart Sutton. "Temporal credit assignment in reinforcement learning." In: (1984).

[242] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[243] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[244] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[245] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE transactions on medical imaging* 35.5 (2016), pp. 1299–1312.

[246] Hong Hui Tan and King Hann Lim. "Review of second-order optimization techniques in artificial neural networks backpropagation." In: *IOP Conference Series: Materials Science and Engineering*. Vol. 495. 1. IOP Publishing. 2019, p. 012003.

[247] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114.

[248] Matthew E Taylor, Nicholas K Jong, and Peter Stone. "Transferring instances for model-based reinforcement learning." In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2008, pp. 488–505.

[249] Matthew E Taylor and Peter Stone. "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7 (2009).

[250] Gerald Tesauro. "TD-Gammon, a self-teaching backgammon program, achieves master-level play." In: *Neural computation* 6.2 (1994), pp. 215–219.

[251] Sebastian Thrun and Anton Schwartz. "Issues in using function approximation for reinforcement learning." In: *Proceedings of the Fourth Connectionist Models Summer School*. Hillsdale, NJ. 1993, pp. 255–263.

[252] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[253] Andrea Tirinzoni, Rafael Rodriguez Sanchez, and Marcello Restelli. "Transfer of value functions via variational methods." In: *Advances in Neural Information Processing Systems*. 2018, pp. 6179–6189.

[254] Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. "Importance weighted transfer of samples in reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4936–4945.

[255] Huan-Hsin Tseng, Yi Luo, Sunan Cui, Jen-Tzung Chien, Randall K Ten Haken, and Issam El Naqa. "Deep reinforcement learning for automated radiation adaptation in lung cancer." In: *Medical physics* 44.12 (2017), pp. 6690–6705.

[256] John N Tsitsiklis and Benjamin Van Roy. "An analysis of temporal-difference learning with function approximation." In: *IEEE transactions on automatic control* 42.5 (1997), pp. 674–690.

[257] Jacob Tyo and Zachary Lipton. "How transferable are the representations learned by deep q agents?" In: *arXiv preprint arXiv:2002.10021* (2020).

[258] Bram Van Ginneken, Arnaud AA Setio, Colin Jacobs, and Francesco Ciompi. "Off-the-shelf convolutional neural network features for pulmonary nodule detection in computed tomography scans." In: *2015 IEEE 12th International symposium on biomedical imaging (ISBI)*. IEEE. 2015, pp. 286–289.

[259] Hado Van Hasselt. "Double Q-learning." In: *Advances in Neural Information Processing Systems*. 2010, pp. 2613–2621.

[260] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. "Deep Reinforcement Learning and the Deadly Triad." In: *arXiv preprint arXiv:1812.02648* (2018).

[261] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double Q-learning." In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[262] Harm Van Seijen, Mehdi Fatemi, and Arash Tavakoli. "Using a logarithmic mapping to enable lower discount factors in reinforcement learning." In: *arXiv preprint arXiv:1906.00572* (2019).

[263] Ryan Van Soelen and John W Sheppard. "Using winning lottery tickets in transfer learning for convolutional neural networks." In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[264] Remy Vandaele, Sarah L Dance, and Varun Ojha. "Deep learning for automated river-level monitoring through river-camera images: an approach based on water segmentation and transfer learning." In: *Hydrology and Earth System Sciences* 25.8 (2021), pp. 4435–4453.

[265] Vladimir N Vapnik and A Ya Chervonenkis. "On the uniform convergence of relative frequencies of events to their probabilities." In: *Measures of complexity*. Springer, 2015, pp. 11–30.

[266] Vladimir Vapnik. "Principles of risk minimization for learning theory." In: *Advances in neural information processing systems*. 1992, pp. 831–838.

[267] Limin Wang, Yu Qiao, and Xiaoou Tang. "Action recognition and detection by combining motion and appearance features." In: *THUMOS14 Action Recognition Challenge* 1.2 (2014), p. 2.

[268] Zheng Wang, Yangqiu Song, and Changshui Zhang. "Transferred dimensionality reduction." In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2008, pp. 550–565.

[269] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay." In: *arXiv preprint arXiv:1611.01224* (2016).

[270] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando Freitas. "Dueling Network Architectures for Deep Reinforcement Learning." In: *International Conference on Machine Learning*. 2016, pp. 1995–2003.

[271] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[272] Qinglai Wei, Derong Liu, and Hanquan Lin. "Value iteration adaptive dynamic programming for optimal control of discrete-time nonlinear systems." In: *IEEE Transactions on cybernetics* 46.3 (2015), pp. 840–853.

[273] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. *Dublin core metadata for resource discovery*. Tech. rep. 1998.

[274] Marco A Wiering. "Explorations in efficient reinforcement learning." PhD thesis. University of Amsterdam, 1999.

[275] Marco A Wiering. "Convergence and divergence in standard and averaging reinforcement learning." In: *European Conference on Machine Learning*. Springer. 2004, pp. 477–488.

[276] Marco A Wiering. "QV (lambda)-learning: A new on-policy reinforcement learning algorithm." In: *Proceedings of the 7th European Workshop on Reinforcement Learning*. 2005, pp. 17–18.

[277] Marco A Wiering and Hado Van Hasselt. "The QV family compared to other reinforcement learning algorithms." In: *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*. IEEE. 2009, pp. 101–108.

[278] Marco Wiering and Jürgen Schmidhuber. "Speeding up Q (λ)-learning." In: *European Conference on Machine Learning*. Springer. 1998, pp. 352–363.

[279] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[280] Jos van de Wolfshaar. "Deep Reinforcement Learnig of Video Games." PhD thesis. Faculty of Science and Engineering, 2017.

[281] Jos van de Wolfshaar, Mahir F Karaaba, and Marco A Wiering. "Deep convolutional neural networks and support vector machines for gender recognition." In: *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE. 2015, pp. 188–195.

[282] Jos van de Wolfshaar, Marco A Wiering, and Lambert Schomaker. "Deep Learning Policy Quantization." In: 2018.

[283] R. Wollheim. *On Art and the Mind. Essays and Lectures*. Allen Lane, 1972.

[284] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." In: *arXiv preprint arXiv:1708.07747* (2017).

[285] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks." In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 5987–5995.

[286] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *arXiv preprint arXiv:1411.1792* (2014).

[287] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. "Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP." In: *arXiv preprint arXiv:1906.02768* (2019).

[288] Guido Zarrella and Amy Marsh. "Mitre at semeval-2016 task 6: Transfer learning for stance detection." In: *arXiv preprint arXiv:1606.03784* (2016).

[289] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[290] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. "Deep reinforcement learning with experience replay based on SARSA." In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2016, pp. 1–6.

[291] Yufan Zhao, Donglin Zeng, Mark A Socinski, and Michael R Kosorok. "Reinforcement learning strategies for clinical trials in nonsmall cell lung cancer." In: *Biometrics* 67.4 (2011), pp. 1422–1433.

[292] Sheng-hua Zhong, Xingsheng Huang, and Zhijiao Xiao. "Fine-art painting classification via two-channel dual path networks." In: *International Journal of Machine Learning and Cybernetics* 11.1 (2020), pp. 137–152.

[293] Yang Zhong, Josephine Sullivan, and Haibo Li. "Face attribute prediction using off-the-shelf CNN features." In: *2016 International Conference on Biometrics (ICB)*. IEEE. 2016, pp. 1–7.

[294] Yi-Tong Zhou and Rama Chellappa. "Computation of optical flow using a neural network." In: *ICNN*. 1988, pp. 71–78.

[295] Rong Zhu and Mattia Rigotti. "Self-correcting Q-Learning." In: *arXiv preprint arXiv:2012.01100* (2020).

[296] Xiaofeng Zhu, Zi Huang, Yang Yang, Heng Tao Shen, Changsheng Xu, and Jiebo Luo. "Self-taught dimensionality reduction on the high-dimensional small-sized data." In: *Pattern Recognition* 46.1 (2013), pp. 215–229.

[297] Xiaofeng Zhu, Xuelong Li, Shichao Zhang, Chunhua Ju, and Xindong Wu. "Robust joint graph sparse coding for unsupervised spectral feature selection." In: *IEEE transactions on neural networks and learning systems* 28.6 (2016), pp. 1263–1275.

[298] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven visual navigation in indoor scenes using deep reinforcement learning." In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3357–3364.

[299] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. "Transfer Learning in Deep Reinforcement Learning: A Survey." In: *arXiv preprint arXiv:2009.07888* (2020).

[300] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. "Discrimination-aware channel pruning for deep neural networks." In: *Advances in Neural Information Processing Systems*. 2018, pp. 875–886.