Department of Informatics, Bioengineering,
Robotics and Systems Engineering

Efficient machine learning with resources constraints

by
Paolo Didier Alfano

accepted on the recommendation of
Prof. Dr. Lorenzo Rosasco
Prof. Dr. Francesca Odone

Theses Series **DIBRIS-TH-2023**

DIBRIS, University of Genoa

Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

Abstract

Since machine learning techniques spread in the scientific community and in real-world scenarios, their usage has been justified by the impossibility of traditional techniques to deal with simple problems that require the retrieval of specific task-related information. In the beginning, neural networks were made of a very reduced amount of layers, with a limited capacity to solve complicated problems. However, in the last years, the set of methodologies we usually refer to as *deep learning* became the de-facto standard in a large variety of fields. Their astonishing ability to solve different kinds of problems has been proven, from very simple and specific tasks to more general problems, such as image recognition, object detection, video recognition, and natural language processing. In the last two years a new approach, referred to as transformers, has been proposed showing state-of-the-art performances in similar contexts to the ones covered by convolutional neural networks. The huge improvement in performances obtained by recent models came at a cost from different points of view. The number of learnable parameters involved moved from tens of millions to hundreds of billions in less than ten years coupled with an increase from a few hundred to millions of PFLOPS needed to train better models in terms of performance. Overall, the amount of energy needed to train the more recent architectures increased drastically in the last few years showing a problematic situation in terms of resources needed to obtain the next state-of-the-art performance. In this thesis, we will see different methodologies to alleviate the computational costs of some typical machine learning problems. First, we will focus on image classification, considering a simple transfer learning approach that exploits pre-trained convolutional features as input for a fast kernel method. By performing more than three thousand training processes, we will show that this fast-kernel approach provides comparable accuracy w.r.t. fine-tuning, with a training time that is between one and two orders of magnitude smaller. Then we will introduce and discuss an unsupervised pipeline that projects input images to a latent space with reduced dimension, making the clustering operation doable. We will show the pipeline effectiveness in a plankton monitoring context where operating in an unsupervised manner is crucial. Indeed, studying plankton population in situ is paramount to protect marine ecosystems as they can be regarded as biosensors. Lastly, we will discuss different methodologies to compare two or more image datasets. Indeed, each dataset can be seen as a set of points sampled by an unknown distribution that we can estimate and analyze. We will introduce different methodologies to study such distributions. We will show that, even on simple tasks involving images, the concept of dataset distance is elusive and very complicated to quantify. It is possible to obtain information on different image datasets, via good partitioning, as long as we analyze a small datasets subset. Overall, in this thesis, we will consider a set of techniques that can alleviate machine learning computational costs, in order to keep them computationally accessible to the scientific community.

Contents

List of Figures

List of Tables

Part I

Introduction & background

1 Introduction

Machine learning is a subfield of artificial intelligence developing algorithms that enable machines to improve their performance on a certain task, through experience. Machine learning models are usually designed to learn from data and make predictions or decisions without explicit instructions. A key hallmark of this algorithm family is the ability, when properly trained, to generalize to previously unseen examples.
The term machine learning was coined in 1959 by electrical engineer Arthur Samuel, proposing the *Samuel's Checkers Player*[268], one of the first impressive examples of self-learning programs. In the last sixty years, the machine learning field has seen astonishing improvements from different perspectives, and nowadays, the massive usage of machine learning techniques is related to the ability to deal with complex problems in an automatic way.

The artificial neural networks family is a popular model group that completely depends on the mechanism we have just described. Thanks to the increasing availability of data and computational resources, the use of neural networks became, in the last decade, a common practice. In the beginning, neural networks were made of a very reduced amount of components, with a limited capacity to solve complicated problems. This reduced ability was, at least partly, related to the small number of neurons in every layer and also to the limited amount of layers[283].
Instead, in the last decade, the complexity of neural networks, evaluated on the number of learnable parameters involved, increased exponentially. This behavior is coupled with the model's capacity to solve difficult problems.
This set of methodologies we usually refer to as *deep learning techniques*[81] became the de-facto standard in a large variety of fields. Their astonishing ability to solve different kinds of problems has been proven, from very simple and specific tasks to more general problems. After 2012 *convolutional neural networks* have been the most popular machine learning model. They focus on a small portion of the data at-a-time via the convolution operation. This approach fits the image context as the model's unit can convolve over the image to grasp useful information.
The neural model's success is mostly due to the great performances obtained in the ImageNet large-scale visual recognition challenge(ILSVRC)[246]. In this challenge, given an image as input, every model must distinguish between one thousand different categories, providing as an answer the most probable category contained in the input image. This operation is performed after a training process over a set of images composed of approximately one million images.
The extraordinary improvements made via these new models in the ILSVRC competition can be shown in Figure 1.1. After the introduction of convolutional neural networks between 2011 and 2012, the ILSVRC classification error dropped from 26% to 16% and even to 2%-3% in the next five years when the challenge ended.

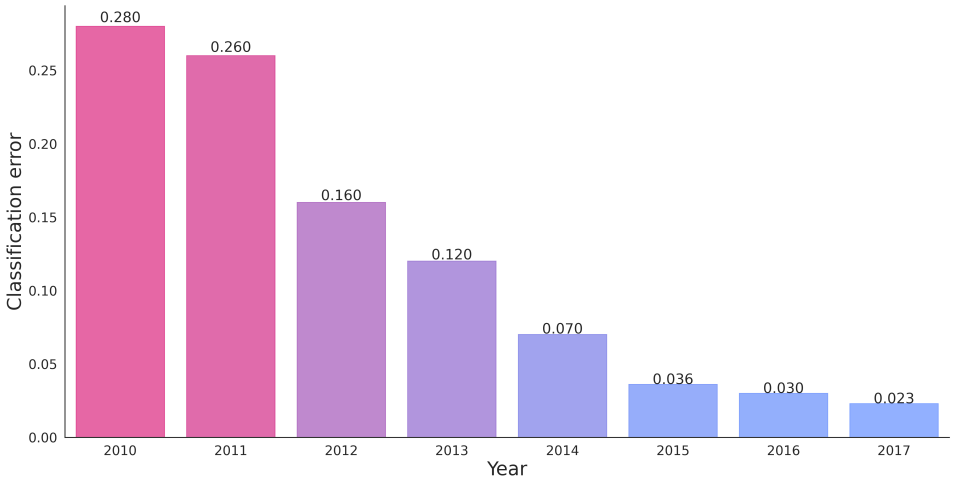


Figure 1.1: The classification error obtained by winning models in the ImageNet large-scale visual recognition challenge(ILSVRC) over the last decade. In 2012 the first convolutional model was introduced, with a massive error reduction w.r.t. the previous year.

Since their first appearance, convolutional neural networks have been applied to a large set of research fields including, but not limited to, image recognition, speech recognition, object detection of images, video recognition, and natural language processing.
Moreover, in the last two years, a new approach referred to as *transformers* has been proposed, showing state-of-the-art performances in similar contexts to the ones covered by convolutional neural networks such as computer vision tasks and image analysis in general.

The huge improvement in performances obtained by recent models came at a cost, from different points of view. Modern machine learning architectures are often referred to as *data hungry* models. Indeed, the amount of labeled and unlabeled data needed by modern architectures increased, and datasets containing tens or even hundreds of millions of images are used to train neural models. Such a huge amount of information is needed to set up properly all the parameters defining modern models. The convolutional model introduced in 2012 which won the ILSVRC competition, i.e. AlexNet, was made by 62.3 million learnable parameters with a training time of between five and six days on two GPUs. Since 2012 a plethora of different architectures has been proposed with an increasing number of parameters. Some model families such as ResNet, EfficientNet, and InceptionResNet stuck to a similar amount of parameters. Some others, like MobileNet, decided to reduce the number of parameters involved to be run on embedded/mobile devices. Nonetheless, architectures such as VGG moved in the opposite direction with more than 100 million learnable parameters.
Since 2017 with the introduction of the attention mechanism and the transformer architecture for natural language processing (NLP) problems, the number of parameters involved in the training process increased exponentially. Some of the first introduced models had comparable parameters number w.r.t. convolutional models. Indeed, models such as ELMo and BERT are made by 94 million and 340 million learnable parameters, respectively. Nonetheless, many recent models

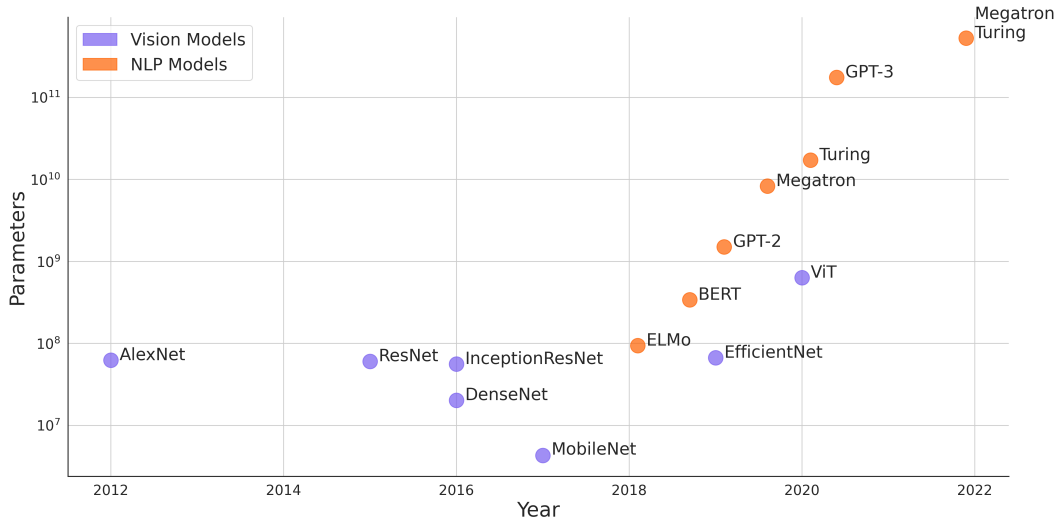


Figure 1.2: Models developed in the last decade with the corresponding amount of learnable parameters.

pushed forward the number of learnable parameters involved. For instance, The GPT-2 model and the Megatron-LM, introduced in 2019, involved 1.5 billion and 8.3 billion learnable parameters, respectively. The GPT-3 model and the Megatron-Turing NLG models, introduced in 2020, involve more than a hundred billion parameters. Even if these models are used nowadays to deal with natural language problems, different models such as Vision Transformer(ViT) have been proposed to apply the same NLP principles to image classification and analysis in general. A representation of parameters involved in some of the most famous machine learning models is shown in Figure 1.2.

The massive increase in the number of parameters involved in the process is, at least partly, coherent with the computational resources involved in the process. The amount of computation needed to train a modern neural architecture is usually expressed with FLoating point Operations Per Second(FLOPS). In the last decade, such quantity moved approximately from $4 \cdot 10^2$ PFLOPS of AlexNet to $3 \cdot 10^8$ PFLOPS of GPT-3. Considering that the amount of FLOPS is directly correlated with energy consumption, the amount of energy needed to train the more recent architecture increased drastically in the last few years, showing a problematic situation in terms of resources needed to obtain the next state-of-the-art performance.

This thesis examines various methodologies that aim to mitigate the computational expenses associated with common machine learning problems. These challenges can be approached from multiple perspectives, given that modern training processes require different resources, including training time, physical memory, and the quantity of available data. Therefore, this thesis concentrates on these three crucial factors in an effort to diminish the impact of machine learning costs. First, we will consider a simple transfer learning approach that exploits pre-trained convolutional features as input for a fast kernel method. By performing more than 3000 training processes, involving 32 target datasets and 99 different settings, we will show that this fast-kernel approach provides comparable accuracy w.r.t. fine-tuning, with a training time that is between one and two

orders of magnitude smaller. We will provide results suggesting that the fast-kernel approach is indeed a useful alternative to fine-tuning in small/medium datasets, especially when training efficiency is crucial. This is typical of robotics devices and autonomous systems, where multiple training may need to be done on the fly. Moreover, our results will show that the marginal benefit of fine-tuning is low dependent on the neural network architecture used as a pre-trained model. On the other hand, the choice of an appropriate pre-training dataset has a significant impact on the obtained accuracy, particularly for the fast-kernel methodology.
After the analysis in terms of training time, we will focus on representation efficiency. We will introduce and discuss the impossibility of a clustering algorithm to deal directly with images. Indeed, even with a low resolution, such images can have tens of thousands of dimensions. At the same time, the feature output size of modern architectures is usually between a thousand to tens of thousands of elements. We will show how we implemented an unsupervised pipeline that projects the input to a latent space with reduced dimension, making the clustering operation doable. We will show results testing our pipeline effectiveness in the plankton monitoring context where operating in an unsupervised manner is crucial. Indeed, detecting and studying plankton populations in situ is paramount to protecting marine ecosystems as they can be regarded as biosensors, reflecting the overall health of the oceans. We will show how we leveraged pre-trained neural network models to extract expressive feature maps efficiently, without fine-tuning. We then use an encoder-decoder network architecture to perform dimensionality reduction, producing low-dimensional embedded features that can then be fed to a clustering algorithm. We will assess our methodology on three plankton datasets with different characteristics and increasing complexity.
Lastly, we will introduce and discuss different methodologies to compare two or more image datasets. We will show that each dataset can be seen as a set of points sampled by an unknown distribution that we can estimate and analyze. As we mentioned, often a key hallmark of deep learning models is the lack of good, labeled data. To overcome this process, different research fields focus on transferring knowledge from related but different data distributions. In this sense, estimating such distributions can be important to provide a better comprehension of our data and whether a set of data is suitable for transferring information to a new setting.
We will introduce different methodologies to estimate image distributions such as histograms and Random Fourier Features, coupled with various methodologies to compute distances between them such as integral probability metrics and f-divergences. Our results will show that, even on simple tasks involving images, the concept of dataset distance is elusive and very complicated to quantify. In particular, two different approaches will be tested providing similar results. We will show that it is possible to obtain information on different image datasets, via good partitioning, as long as we analyze a small dataset subset.
Overall, in this thesis, we will consider a set of techniques that can alleviate machine learning computational costs. In our work, we will focus on three different efficiency aspects: training time, compressed data representation, and datasets distance. Each one of them focuses on a distinct efficiency aspect of machine learning. For all the aforementioned reasons, aiming for computational efficiency in machine learning tasks is necessary to reduce the greater budget required nowadays by modern models. Reducing the cost of such models is going to be one of the greatest challenges we will face in the future, to keep them computationally accessible to the scientific community.

The structure of the thesis is the following. In chapter 2 we will introduce the basic ingredients

needed in the machine learning context. We will introduce different machine learning settings, such as supervised and unsupervised learning or predictive and descriptive models. We will provide details about the optimization process involved. In chapter 3 we will focus on the training time efficiency by comparing two machine learning approaches on the image classification problems. In chapter 4 we will consider the representation efficiency aspect, by considering an unsupervised problem applied to plankton images. In chapter 5 we will focus on the datasets distance problem, applied to the image context to compare two or more datasets. Lastly, in chapter 6 we will furnish a set of final remarks and conclusive considerations to summarize our findings and provide an overall assessment of our research.

2 Machine learning Frameworks

In this thesis, we selected different tools to analyze the bottlenecks in some common pipelines related to our studies. In this chapter, we are going to introduce some instruments that were adopted in all the research we performed. We start by introducing machine learning and its main components such as tasks, performance measure and experience. We conclude by introducing some fundamental machine learning models such as neural networks, convolutional neural networks and transformers.

2.1 Introduction

Machine learning is a field of artificial intelligence based on statistical techniques to give computer systems the ability to "learn" over a certain problem. Many books and papers provide a good presentation of the basic machine learning concepts. In this thesis we are going to introduce them relying mostly on two books by Tom Mitchell[194] and Peter Flach[68], respectively.

2.1.1 An intuitive definition

Different definition were provided in the last decades. A simple and precise definition of the learning process was presented by Tom Mitchell in its book *Machine Learning*[194]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

For instance in a machine learning program that learns to play chess, the performance could be evaluated by considering the percentage of won games against another program or against a human being. In general, to have a well-posed learning problem we must define three elements: a task T, a performance measure P and the experience E.

2.1.2 Tasks

Determining precisely the task, i.e. the problem we want to solve, is essential for a good start. Moreover, the whole development steps can be guided by the task itself. The task can be more or less intuitive to be defined, depending on different factors. Examples of tasks can be the following:

- In a chess learning program the task could be simply playing or winning chess games.
- In a handwriting machine learning application the task could be to recognize and classify handwritten words.

- In a medical imaging context the task could be to determine either the presence or absence of malign tumors in the input image.

- In a driving learning problem the task could be to drive an automated vehicle on a public four-lane highway using vision sensors.

2.1.3 Performance measure

One of the crucial aspects in the machine learning field is how to determine whether the program is improving at a task. The performance P is the measure we use to evaluate such enhancement. The used metric determines how the performance of machine learning algorithms is measured and compared, with respect to previous and future works.

A widespread metric adopted in machine learning problems is the so-called *loss*. This is a general term used to indicate functions that measure the error committed by the program. Losses are also used during learning to guide the algorithm. In these cases, continuous functions are preferable. Formally, consider a set of n input-output pairs $\{(x_i, y_i)_{i=1}^n\} \subset \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the input domain and $\mathcal{Y}$ the output domain. We are going to call these points *training samples* or, equivalently, *training set*. The input is generated according to an unknown distribution P and labeled by a function F that maps the inputs to the outputs. In this sense $f : \mathcal{X} \to \mathcal{Y}$.

Regression: consider a program h that learns to map the input to the output. In this sense, we can see this program as a function such that $h : \mathcal{X} \to \mathcal{Y}$. The performance of h can be measured by computing the probability of getting a random instance x according to the unknown input distribution $\mathcal{D}$ where $p(x) \neq f(x)$.

Formally this performance measure l with $l : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ can be expressed by the following equation:

$$l(h, f) \stackrel{\text{def}}{=} \mathbb{P}_{x \sim P}[h(x) \neq f(x)] \tag{2.1}$$

More in general, our end goal would be to predict effectively the output starting from input on the whole distribution and in this sense, we can compute the *expected risk*:

$$L_P(h, f) \stackrel{\text{def}}{=} \mathbb{E}_{x \sim P}[l(h, f)] = \int_{\mathcal{X} \times \mathcal{Y}} l(h, f) d\mathcal{P} \tag{2.2}$$

As we were mentioning the input points x are distributed accordingly to the unknown distribution P and labeled by f. Since we do not know either P or f we need to consider only the n points in the training set to estimate them. From this idea, we can define the *expected empirical risk*

$$\hat{L}_P(h, f) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} l(h(x_i), f(x_i)) \tag{2.3}$$

At the core of these definitions we have the loss function itself. Given a point x_i it computes the error measure obtained by comparing the real label $f(x_i)$ with the output obtained by our

program $h(x_i)$. Assuming that both $f(x_i) \in \mathbb{R}$ and $h(x_i) \in \mathbb{R}$, we are considering a *regression* setting, we can compute different loss functions. Some common choices are:

- *Square loss*: this kind of loss is probably the simplest and most common in machine learning problems. It just calculates the difference between the real value and the predicted output and squares it:

$$l_s(y, y') = (y - y')^2 \tag{2.4}$$

  where $y = f(x)$ and $y' = h(x)$. The square loss function is both convex and smooth. Moreover, this loss is simple and prevents large errors by making them costly. At the same time in many real scenarios, we could have misleading input data for many reasons: systematic error in the instruments or in the data pre-processing. Such inputs, called *outliers* are usually discarded and instead can have a deep impact when using square loss as they contribute massively to the error computation.

- *Absolute loss:* this function is very similar to the previous one:

$$l_a(y, y') = |y - y'| \tag{2.5}$$

  In this situation, the loss function can be considered complementary to the square loss. The advantage is that dealing with an outlier will not deeply influence the training but at the same time when handling incorrect answers from the program they will have a limited impact.

- *ϵ-insensitive loss*: this function defines a margin of tolerance where no penalty is given to committed errors. Its definition is partly similar to the absolute one:

$$l_\epsilon(y, y') = \max(|y - y'| - \epsilon, 0) \tag{2.6}$$

In Figure 2.1 we show all the aforementioned regression loss functions.

Classification: so far we assumed that both $f(x_i) \in \mathbb{R}$ and $h(x_i) \in \mathbb{R}$. As we were saying this setting is usually known as a regression setting. A different scenario we could deal with is the *classification* setting. In this situation, the label of our data is not a real number anymore. It is, instead, an instance of a categorical variable. This situation is very common and some simple examples of classification problems were given in subsection 2.1.2: when dealing with handwritten characters, selecting (i.e. classifying the character) the right character is a classification problem with a number of categories equal to the number of valid characters. When dealing with medical images, determining the presence or the absence of a tumor is a classification problem with two categories.

Similarly to what we have seen previously, we can have an expected empirical error $\hat{L}_P(h, f)$ that depends on the loss function. Suppose to consider a very simple binary classification problem. From this setting we can define different loss functions:

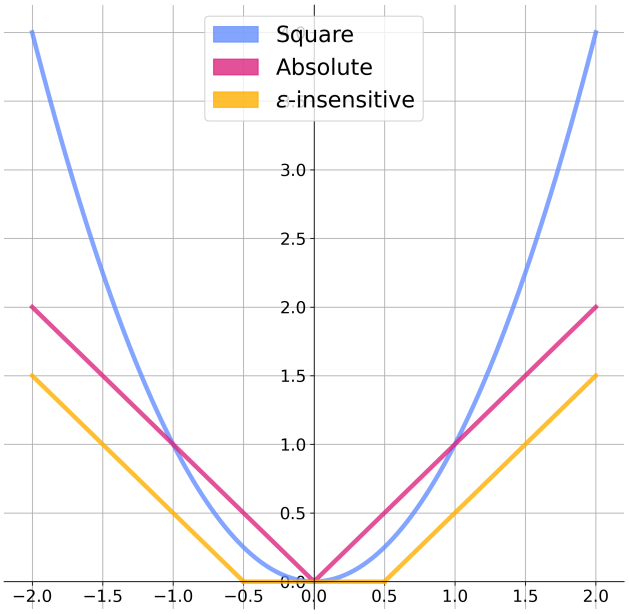


Figure 2.1: The regression losses as presented in Equation 2.4, Equation 2.5 and Equation 2.6

.

- *0-1 loss*: this is the simplest loss function for classification problems. It returns as output a 1 every time an element is misclassified, and a 0 otherwise

$$l_{0-1}(y, y') = \begin{cases} 1 & \text{if} \quad -yy' \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

- *Square loss*: even if it is more common in regression setting, the square loss can be rewritten to fit the classification scenarios

$$l_{sc}(y, y') = (1 - yy')^2 \tag{2.8}$$

  Even if we can use such a loss function for classification problems this is not a very common choice.

- *Hinge loss*: this kind of loss function is common and used in different scenarios. One of the most typical uses of hinge loss is for the *Support Vector Machine* model (SVM). This function incorporates a margin from the classification boundary into the cost itself

$$l_h(y, y') = \max(1 - yy', 0) \tag{2.9}$$

  The intuition is that even if a data point is correctly classified it can incur in a penalization. Vice-versa a misclassified point closer to the margin will receive a smaller penalty.

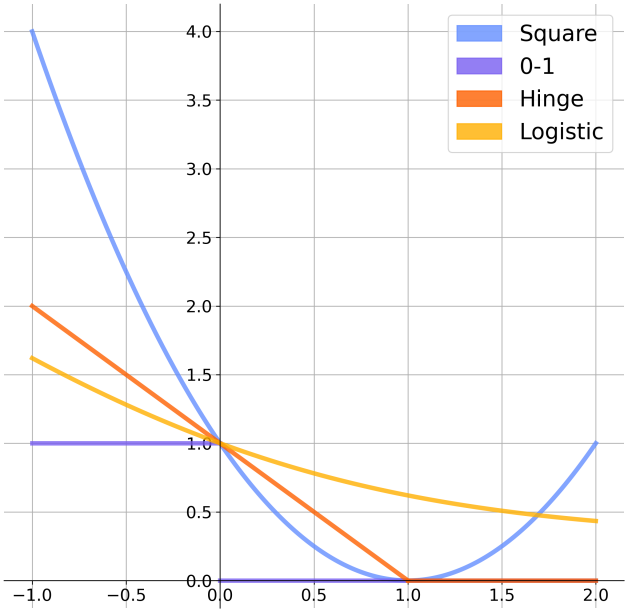


Figure 2.2: The classification losses as presented in Equation 2.7, Equation 2.8, Equation 2.9 and Equation 2.10

.

- Logistic loss: this function is the most important one based on probabilities. Instead of classifying directly a probability output is given

$$l_{log}(y, y') = \log(1 + e^{-yy'}) \tag{2.10}$$

  The intuition is: the more the probability value diverges from the real one, the higher the logistic loss value is.

In Figure 2.2 we show all the aforementioned classification loss functions.

It is quite common to exploit a different metric to evaluate our model. All the classification losses we saw, aim to be minimized. Instead, we can define a new metric, named *accuracy* as:

$$acc \stackrel{\text{def}}{=} 1 - \frac{1}{n}\sum_{i=1}^{n} l_{0-1}(y, y') \tag{2.11}$$

Intuitively the accuracy simply measures the percentage of data points classified correctly. Accuracy can be rewritten accordingly to this intuitive definition:

$$acc \stackrel{\text{def}}{=} \frac{c}{p} \tag{2.12}$$

where c is the number of correct predictions while p is the total number of predictions.

The relationship between loss and accuracy is peculiar because usually in a machine learning model we can use the first one easier to improve our model. At the same time, the second one is what we usually care about in real-life scenarios.

Train, validation and test sets: we saw that, given a bag of data points called a training set, we can teach our model to improve its on a certain task. In addition to the training set, we can introduce a *validation set* and a *test set*. To explain why, suppose we are studying for a university test. To improve our performance, It is usually a good idea to study tests provided previously by the teacher. In this sense, this set of tests we can study and rely on is the training set. The test we are going to take at the end of our training phase is called *test set*. In real-world scenarios, the training set can be composed of millions or even billions of samples. Also, the test set is usually large to verify the effective ability of the model to generalize to unseen examples. The most important thing about the test set is that, as the name states, it should never be used until the very final test phase. Such a set of samples should be used only to assess the performance of a fully-trained model.
Another set of data that we need to introduce is the *validation set*. While the test set was used only to assess the performance on unseen data, the validation set can be used to tune the hyper-parameters of the function we are learning. We will see in the next pages that a model can learn to map the input to the output via a set of parameters. Such parameters can be millions or even billions but they are learned automatically. At the same time, each model has a set of hyper-parameters that must be set up by a human before we start the training procedure. The validation set can be used precisely to tune this set of hyper-parameters.

When we are provided a set of data we should split them into these three sets. This way we can learn automatically the parameters via the train set, tune the hyper-parameters manually via the validation set and verify the model's behavior on previously unseen samples via the test set. So far there are no clear guidelines on how to split optimally between train validation and test sets. Different studies focus on the split ratio between train and test sets. Some works underlined that a ratio around $[70 : 30]$ is reasonable[54, 222]. Others works[4, 223] stated that this ratio should be between $[75 : 25]$ and $[50 : 50]$. Moreover, different works[150] stated via an asymptotic study that the ratio should tend to $[0 : 100]$ when the dataset size increases.

2.1.4 Experience

According to the English Oxford Dictionary:

> **Ex·pe·ri·ence** [ɪkˈspɪə.ri.əns], noun: the knowledge and skill that you have gained through doing something for a period of time.

in this sense, the "knowledge and skill" collected by the program is very similar to one accumulated by a human being. Usually, the source of experience corresponds to the data used to train the system and can have a significant impact on the results over the task. The source of data can be *direct* or *undirect*. For instance, in a machine learning program about chess, a direct form of experience could be chess board configurations associated with the best move to do for each configuration. An undirect form of experience could be a sequence of moves made by a human

player. Of course, this second approach presents an additional problem about "how good" the moves are by considering the final result of the match. Considering that the experience is directly tied to the input data is important to consider in advance the kind of input as it is going to influence deeply the model used to solve the task. One common distinction is between *labeled* and *unlabeled* training data:

- Labeled data: in this situation, the data are given with their "real" label. For a regression problem this could be the value of the $y \in \mathcal{Y}$ variable. Instead, in a classification setting could be the class the $x \in \mathcal{X}$ belongs to. This kind of information can be used to guide our program to produce better outcomes $\hat{y}$.
- Unlabeled data: in this situation, we have data that has not been annotated by human experts. In practice, we are removing the important $\mathcal{Y}$ information.

Such separation between labeled and unlabeld data marks a distinction between two categories of learning problems: *supervised* and *unsupervised*, respectively.
Usually, the knowledge is accumulated by the program over a period of time. In this sense, there are different quantities that measure the time spent by a program to learn about a task. Different types of programs rely on different "time" quantities:

- Epochs: when the program analyzes the entire available set of data and updates its parameters accordingly.
- Steps: usually, with the modern dataset increasing in size, it is not possible nor suitable to update the program's parameter at the end of the epoch. Commonly the set of data is split in *batches* and the program updates its parameter after analyzing a single batch. So we can see an epoch as a sequence of steps.

2.2 Model

So far we referred to the function that determines what type of knowledge will be learned as a program. This kind of function h, as defined in subsection 2.1.3 maps the elements from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$ as:

$$h : \mathcal{X} \to \mathcal{Y} \tag{2.13}$$

From now on we are going to refer to this function as the *model*. Usually, the model function needs a mathematical representation related to the relevant information about the problem. These pieces of information are called *features* and they are usually indicated as $x_1, \ldots, x_n$.
Once we determined the features, a very simple mathematical representation of the target function could be:

$$\hat{h}(x) = w_0 + w_1 x_1 + \ldots + w_n x_n \tag{2.14}$$

where $w_0, \ldots, w_n$ is a set of weights that the model must determine.

So far in this section, we provided a general definition of the model as a target function, mapping the input space to the output space. There are numerous ways to provide a more fine-grained definition. A common one consists in dividing models into two categories:

- *Supervised*: the model is presented with example inputs and their desired outputs. The goal is to learn a general function Φ that maps inputs to outputs. These outputs, usually called *labels* are used to "teach" the program the correct answers for every training example. Without noticing this is what we did in the whole subsection 2.1.3. In this context we have a set of input-output pairs $\{(x_i, y_i)_{i=1}^n\} \subset \mathcal{X} \times \mathcal{Y}$. The needed ingredients of the supervised setting are:
  - The set of input-output pairs
  - The model itself, mapping the input to the outputs: $h : \mathcal{X} \to \mathcal{Y}$
  - A loss function $L(h, f)$ chosen between the ones presented in subsection 2.1.3, depending on the nature of the problem itself, e.g. regression or classification.

  With these elements, the supervised learning task can be defined and solved as an optimization problem such that:
  $$\min_{h:\mathcal{X}\to\mathcal{Y}} L(h, f) \tag{2.15}$$
- *Unsupervised*: the model is presented with example inputs without their real labels. The goal is to find relations within the data that help to understand better the data. This is done by inferring the properties of the data probability distribution without the help of the labels.
  In this context, we would like our model to learn a good representation of the input data. In this sense we want to learn a general function Φ as:
  $$\Phi : \mathcal{X} \to \mathcal{F} \tag{2.16}$$
  where $\mathcal{F}$ is the feature space we want to learn.

Aside from these two precise categories, different research field about "intermediate" situations has been explored. For instance, is possible to find in literature other terms such as *weakly supervised learning*[316, 324] and its subset *semi-supervised learning*[36, 291]. Another different approach is the one proposed by *self-supervised learning*[119, 169]. A distinct modern categorization between machine learning models is between *predictive* and *descriptive* models. As presented by Peter Flach[68]:

- Predictive model: this type of model try to predict the value of a variable. Usually, a predictive model uses statistical techniques and forecast methodologies to give a prediction.
- Descriptive model: this group of models wants to understand something about the data without doing any prediction. Usually, a descriptive model uses data aggregation and data mining to provide insights and give an explanation to data.

By considering the two aforementioned categorizations (supervised versus unsupervised and predictive versus descriptive) we can compile Table 2.1 where an overview of different machine learning settings is given along with examples from every setting.

	Predictive model	Descriptive model
Supervised	classification, regression	subgroup discovery
Unsupervised	predictive clustering	association rule discovery

Table 2.1: Overview of different machine learning settings

One last distinction we can underline about machine learning models is related to the task a machine learning program wants to solve. By taking a second look at Table 2.1 we can notice that machine learning models solve a large variety of problems: classification, subgroup discovery, predictive clustering. . . Yet another way to look at machine learning, concerns the kind of models it is able to build, most of the models can be classified as it follows:

- Geometric: a model that aims to collect information about data in a geometric space like $\mathbb{R}^n$. The aim of the model is to infer some geometric properties regarding the data. This group of models can be divided into two subgroups: *linear* models and *distance based* models.

  - Linear: this group of models operates directly in space and is based on geometrical entities like lines or planes, and, most commonly, hyper-planes. Usually, these models try to separate A very simple example of a linear model is the one presented in Equation 2.14. By setting:
    $$\hat{h}(x) = 0$$
    we are essentially defining an hyper-plane in the $\mathbb{R}^{n+1}$ space.
    This kind of model is widely used because, even with its simplicity, can represent some real-world scenarios. Different models belong to this category, for instance, support vector machines, and least-square methods.

  - Distance based: in machine learning a useful concept is the one of *distance*. When two instances coming from the same dataset are close in their features space, they could be similar and this is inherently tied to how the space is defined in terms of the features. In the cartesian space, it can be useful to measure the distance between two points x and x' in terms of the euclidean distance $\sqrt{\sum_{i=1}^{d}(x_i - x'_i)^2}$ where d is the space dimensionality. We will also see that the euclidean distance is not always suitable for every feature space. Different types of distance can be used e.g. cosine distance, Mahalanobis distance, and city-block distance.

- Probabilistic: a model that is probabilistic in nature, considering that randomness plays a role in predicting future events. The main idea here is to use the grounding of probability theory to make inferences about a given variable, given the data at hand. In this sense, when some $x \in \mathcal{X}$ is given we would like to predict the outcome, i.e. the $y \in \mathcal{Y}$. For this reason, probabilistic models focus mainly on the $P(Y|X)$ relation. Commonly this probability is studied by using the Bayes rule:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{2.17}$$

With Bayes rule we can compute the *maximum posterior probability*

$$Y = \arg\max_Y P(Y|X) = \arg\max_Y \frac{P(X|Y)P(Y)}{P(X)} \propto \arg\max_Y P(X|Y)P(Y) \tag{2.18}$$

Some examples of probabilistic models could be the naive Bayes model, the Bayes model, the logistic regression model, and Hidden Markov Model.

- Logical: these types of models are called logical because they can be easily translated into a set of rules easily understandable by humans. The set of rules can be also organized as a tree called *feature tree*. The intuition behind this approach is to partition the instance space by applying part of the rule included in the set. By doing so the instance space is divided into a set of hyper-rectangles with decision boundaries corresponding to the edges of the hyper-rectangles. Some instances of logical models are ordered/unordered rule sets and decision trees.

2.3 Kernel methods

As we have seen in the previous section, machine learning models can be studied from different perspectives. Machine learning models can address different questions, depending on their purpose and their aim. In this section we are going to provide details about *kernel methods*.
We have seen that in the geometric models family, the linear models operates directly in the input space. Despite of their simplicity, they are widespreaded as can be easily interpretable and can describe many real-world scenarios. Usually, if we want to deal with a classification problem via a linear model, we would like to separate the input in a meaningful way. Such situation is usually interpreted by a *Support Vector Machine*, aiming to separate the input and maximizing the margin between different classes.
We have seen previously that, training a linear support vector classifier involves an optimization procedure. The goal is to maximize the margin, which is the distance between the closest pair of data points belonging to opposite classes. These data points are known as support vectors, as they determine the decision boundary. In order to train the support vector classifier, we must identify the hyperplane with the maximal margin or optimal separating hyperplane, which effectively separates the classes and allows us to generalize to new data and make accurate predictions.
When the data cannot be separated linearly in their original space, we can transform them using mappings $\phi(x)$ into a feature space with a higher dimension, aiming to achieve linear separability between the classes in the new space. By fitting a decision boundary in this higher dimensional feature space, we can separate the classes and make predictions. While there are different transformations that can produce data linearly separable in higher dimensions, not all of these functions qualify as kernels. The kernel function possesses a unique property that makes it highly advantageous for training support vector models. The technique of exploiting this property to optimize

non-linear support vector classifiers is commonly known as the *kernel trick*.
The kernel trick involves representing the data through a series of pairwise similarity comparisons between original data observations x, which retain their original coordinates in the lower dimensional space. This approach avoids the explicit application of transformations $\phi(x)$ and representation of the data using the transformed coordinates in the higher dimensional feature space. Formally, given two data samples $x, x' \in X$ and a map $\phi : X \mapsto \mathbb{R}^n$, we can define a *kernel function* as

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{2.19}$$

The kernel trick's main advantage is that the objective function we optimize to fit the higher dimensional decision boundary involves solely the dot product of the transformed feature vectors. This allows us to easily substitute these dot product terms with the kernel function, eliminating the need for the $\phi(x)$ transformation.
Now we provide details on the kernel trick. In Equation 2.14 we presented a very simple model called *linear model*. We can describe it in a more compact way as $h_W(x) = \sum_{j=1}^{d} w_j x_j = W^\top x$. Previously in this chapter we have seen also a set of possible performance measures, introducing the square loss in Equation 2.4. Usually when dealing with a loss function for real-world scenario we need another term called *regularization term*, preventing the model to fit perfectly the training data. In section 2.7 we will provide all the necessary details on this subject. The regularized version of the square loss can be rewritten as

$$l(X, y, W, \lambda) = \sum_{i=1}^{n} (y_i - W^\top x_i)^2 + \lambda \sum_{j=1}^{d} W_j^2 \tag{2.20}$$

where λ must be set a-priori, determining the importance of the regularization term. The current framework provides us a closed-form solution for the parameters W as

$$W = (X^\top X + \lambda I)^{-1} X^\top y \tag{2.21}$$

As we were descrbing, we can introduce a non-linear feature map $\phi(x)$ and apply it to the input X, obtaining:

$$h_W(x) = W^\top \phi(X) \tag{2.22}$$

we can notice that by using a non-linear function ϕ the resulting model is non-linear in the input X but is it still linear in W. We can express the ϕ transformation in terms of the associated matrix Φ:

$$\Phi = \begin{bmatrix} \phi(x_{11}) & \dots & \phi(x_{1p}) \\ \phi(x_{21}) & \dots & \phi(x_{2p}) \\ \vdots & & \vdots \\ \phi(x_{n1}) & \dots & \phi(x_{np}) \end{bmatrix}. \tag{2.23}$$

By introducing Φ matrix we can provide a closed form solution to obtain the best parameters with the new attributes as

$$W = (\Phi^\top \Phi + \lambda I)^{-1} \Phi^\top y. \tag{2.24}$$

showing that we can obtain a solution for the optimal parameters also when our input are transformed via ϕ. We can notice that finding the optimal solution cost us $\mathcal{O}(p^3)$ as we have to invert the $\Phi^\top \Phi$ matrix. Via the push-through matrix identity equation we can rewrite our problem in the following way, usually called the *dual form*

$$W = \Phi^\top (\Phi \Phi^\top + \lambda I)^{-1} y \tag{2.25}$$

We can notice that finding the optimal solution cost us $\mathcal{O}(n^3)$ as we have to invert the $\Phi^\top \Phi$ matrix. Solving this equivalent problem can be preferable whether $p > n$.
Another even more important observation is about the dual form we just introduced. We can notice that the output of $(\Phi \Phi^\top + \lambda I)^{-1} y$ is a vector composed by n positions. We are going to refer to such vector, as α

$$W = \Phi^\top \alpha \tag{2.26}$$

Indeed, by also noticing that $\alpha_i = \sum_{j=1}^{p} L_{ij} y_j$ where $L = (\Phi \Phi^\top + \lambda I)^{-1}$, we can rewrite the optimal solution of the n training points as

$$W = \sum_{i=1}^{n} \alpha_i \phi(x_i) \tag{2.27}$$

This is a very important observation as once we receive a new input x' if we want to compute a prediction about it we can apply the optimal weights expressed by Equation 2.27 to it:

$$\phi(x')^\top W = \sum_{i=1}^{n} \alpha_i \phi(x')^\top \phi(x_i) \tag{2.28}$$

stating that the model prediction according to the computed set of weights W is a linear combination of $\phi(x')$, our new input after we applied the ϕ transformation to it, and the featurized version of every other transformed training examples $\phi(x_i)$ weighted by the corresponding value α_i.
The crucial observation here is that to determine the prediction of a new input we do not need the transformation of the inputs $\phi(x_i)$ or $\phi(x')$ if we know directly their dot product. This is the key idea behind the kernel trick we introduced in Equation 2.19. It is worth noticing that we just need the dot product between $\phi(x_i)$ and $\phi(x')$ even at training time. Indeed, by recalling that each row i of Φ is the i-th featurized input $\phi(x_i)^\top$, we can define a matrix $K = \Phi \Phi^\top$ as the matrix of all dot products between all the $\phi(x_i)$ where

$$K_{ij} = \phi(x_i)^\top \phi(x_j). \tag{2.29}$$

Now, by recalling and updating the definition of $\alpha = (K + \lambda I)^{-1} y$ we can notice that also in this definition we only have K where we only have the dot product $\phi(x_i)^\top \phi(x_j)$ directly. Indeed, we can notice that at training time we just need to compute the K matrix usually referred to as

Kernel name	Product result
Polynomial	$(x^\top x' + c)^d$
Sigmoid	$\tanh(\gamma x^\top x' + c)$
Gaussian	$\exp(\frac{-\|\|x-x'\|\|^2}{\sigma^2})$
Laplacian	$\exp(-\gamma\|\|x - x'\|\|_1)$
χ^2	$\exp(-\gamma \sum_i \frac{(x_i - x_i')^2}{x_i + x_i'})$

Table 2.2: Different types of kernel

kernel matrix. Then, we can plug values of K in the α vector that can be used later at test time to predict outcome of unseen samples. The main advantage about the kernel trick is that with some specific function, computing the product $\phi(x_i)^\top \phi(x_j)$ is equivalent to another formulation that is cheaper to compute. We can show some examples of well-known kernels in Table 2.2

2.4 Neural networks

A very popular model family in machine learning is *artificial neural networks*. These models are vaguely inspired by the biological neural networks that constitute animal brains.
Neural networks were introduced for the first time in 1943 by McCulloch and Pitts[186]. After many modifications and improvements[193, 245], neural networks are used nowadays to solve specific tasks, both in a supervised and unsupervised way.
By now, neural networks have been used in a large variety of fields for over thirty years[8, 226]. Some applications are: handwritten character recognition[152, 171], speech recognition[52, 148, 200] and object detection of images[117, 168, 320], for instance faces recognition[69, 183, 300] or object classification[143, 278].

A neural network is made by *neurons*, a generalization of *perceptrons*. In the simplest neural network model a neuron computes a function called *transfer function* and checks through an *activation function* whether the output of the transfer function is greater than a certain threshold. If so, the neurons "fires" providing a continuous value as output. The neuron model structure is shown in Figure 2.3.

Usually the transfer function computes the following quantity:

$$\sum_{i=0}^{n} w_i x_i + t = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + t \tag{2.30}$$

The activation function checks whether the neurons must activate by verifying a logical condition. As we were saying this condition could be, for example, that the output is greater than a certain value. In that case the activation function is called *step function*.
We can have different activation function types. We can show below some of them:

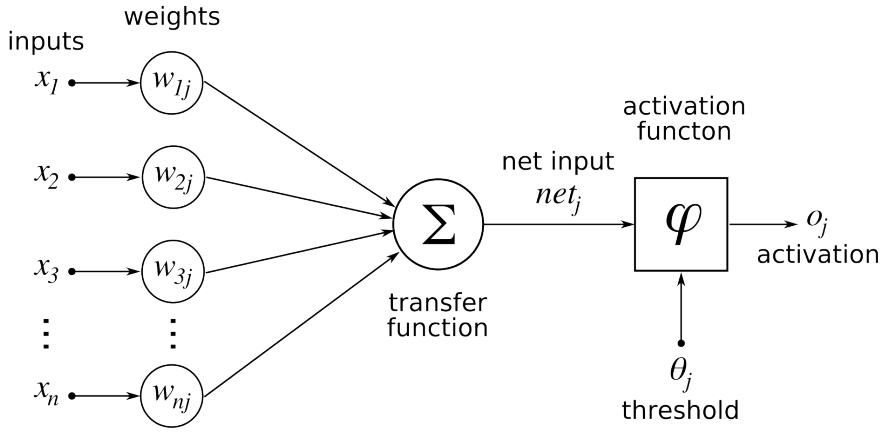


Figure 2.3: Artificial neuron model

- Step function:

$$\phi(v) = \begin{cases} a, & \text{when } v < 0 \\ b, & \text{when } v > 0 \end{cases} \tag{2.31}$$

In its most simple version, the activation function can fire a value b when the input is greater than 0. Mathematically, the step function can be written as a finite linear combination of indicator functions of intervals. A simple example of a step function is the sign function $sgn(x)$, assuming -1 for negative numbers and $+1$ for positive numbers. The most biologically-intuitive interpretation is the one where the output of the step function is 1 when the neuron "fires" and 0 otherwise, meaning a turned-off neuron. To such a special scenario is granted the name *Heaviside step function*. It is equivalent to the sign function, up to a shift and scale of range. Indeed, $H(v) = (sgn(v) + 1)/2$.

- Logistic sigmoid function:

$$\phi(v) = \frac{1}{1 + e^{-v}} \tag{2.32}$$

the sigmoid function is bounded, monotonic, and differentiable, defined for all real input values with a non-negative derivative on the whole real interval. Moreover, the output of the logistic sigmoid function is bounded between 0 and 1 and for this reason, it is often used in the neural network field to predict the probability as output.

- Gaussian function:

$$\phi(v) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{v-\mu}{\sigma})^2} \tag{2.33}$$

The Gaussian function is an alternative to the previous activation functions. Differentiable and easy to approximate to $\phi(v) = e^{-v^2}$. In the neural network field, it is used when dealing with image processing, where two-dimensional Gaussians are used for Gaussian blurs.

- Rectified linear unit function(ReLU):

$$\phi(v) = \begin{cases} 0, & \text{when } v < 0 \\ v, & \text{when } v \geq 0 \end{cases} \tag{2.34}$$

A general problem with some of the previous functions is saturation. For instance, the sigmoid function with large values tends to 1 while tending to 0 for small values as input. Further, some functions are only really sensitive to changes around their mid-point. To address this problem another activation function called Rectified Linear Unit is proposed. Nowadays, especially in deep networks, this function and its alterations are the most used ones to compute the output of a neuron. This function family was selected as de facto standard because of some advantages. First, they are easy and fast to compute. Moreover, they provide a sparse representation, being able to produce a real zero as output. Lastly, it is mostly linear and easier to optimize.

- Gaussian error linear unit(Gelu):

$$\phi(v) = v\varphi(v) \tag{2.35}$$

where $\varphi(v)$ is the cumulative distribution function of a gaussian distribution with $\mu = 0$ and $\sigma = 1$. Originally proposed to merge dropout functionality with ReLU activation function. An intuitive way to interpret the Gelu activation function is by noticing that for negative values the function tends to 0 as the cumulative function is closer to 0. At the same time for positive values the Gelu function tends to the identity as the cumulative is closer to 1. In practical scenarios, the Gelu function is often approximated as

$$\phi(v) = \frac{v}{2}\, tanh\left(\sqrt{\frac{2}{\pi}}(v + 0.045v^3)\right) \tag{2.36}$$

Such approximation is faster to compute and does not rely on the cumulative distribution tables associated with the gaussian distribution function.

We can show all the aforementioned activation functions in Figure 2.4.

Once we decide which activation function to use, we can build the "real" neural network by constructing a *layer* of the network that is made by a certain number of neurons. By considering a basic version of a neural network as the one we introduced so far, we can show it in Figure 2.5. Clearly, in real-world scenarios, networks are made by more than one layer, processing the information one layer after another. Formally, a neural model can be defined as:

$$f(x) = W\Phi(x) \tag{2.37}$$

that is, for $W = (w^1, \ldots, w^T)$,

$$f^t(x) = \langle w^t, \Phi(x) \rangle \tag{2.38}$$

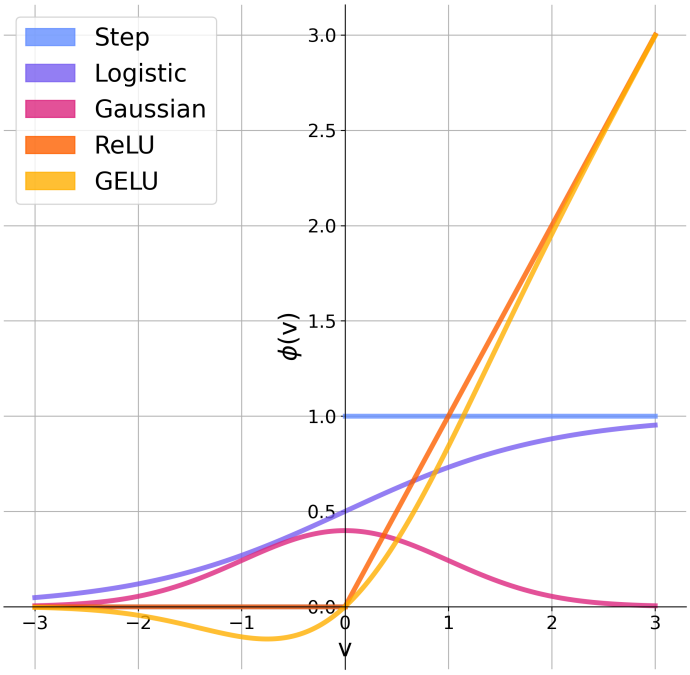


Figure 2.4: Common activation functions in neural networks field.

where the Φ is obtained composing L representations of the input

$$z \in \mathbb{R}^{u_{\ell-1}} \mapsto \Phi(x) = \Phi_L \circ \Phi_{L-1} \circ \ldots \circ \Phi_1(x) \in \mathbb{R}^{u_\ell}, \tag{2.39}$$

where u_ℓ is the number of hidden units in the ℓ-th layer (with $u_0 = d$) .
Each map Φ_ℓ corresponds to the ℓ-th layer and is further parameterized.
Fully connected layers are of the form

$$\Phi_\ell(z) = \sigma(B_\ell z + b_\ell) = (\sigma(\langle z, B_\ell^1 \rangle + b_\ell), \ldots, \sigma(\langle z, B_\ell^{u_\ell} \rangle + b_\ell^{u_\ell})) \tag{2.40}$$

where B_ℓ is a $u_{\ell-1}$ times u_ℓ matrix with rows $B_\ell^1, \ldots, B_\ell^{u_\ell}$, and σ is a component-wise non linearity, e.g. $\sigma(a) = \text{ReLU}(a) = \max\{0, a\}$ for $a \in \mathbb{R}$.

Another important part of the neural model, providing some interpretability to the problem is the *softmax* component. Such part of the network it is usually used at the end of the network. It implements the softmax function that, given a vector z with length i, is defined as

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}. \tag{2.41}$$

The softmax layer has two purposes. The first one is to normalize the results coming from the network. Indeed the output of the layer before the softmax one is going to be a set of numbers related to the activation level of the previous neurons: the greater the value, the more the neuron

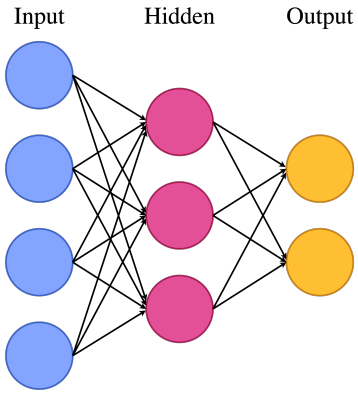


Figure 2.5: An example of neural network

activeness. In this sense, the softmax output is instead a normalized version of the previous layer output summing up to one. Similarly to probability distributions, this aspect provides the user a slight form of interpretability as we can see every neuron firing proportionally to how probable is to activate it. In particular, when the softmax layer is used in classification contexts, in the last layer we have a neuron for each class. When coupled with the softmax layer we have precisely the probability for each class associated with each neuron. The second interesting aspect is related to the previous one and is that softmax is a smoothed version of the argmax function. It is worth noticing that all the nodes in a single layer are linked with all the nodes of the next layer. These types of neural networks are called *fully connected networks*. In another context, a neural network node could be linked with just part of the neurons of the next layer. These other type of networks are called *sparse neural networks*.

The main reason neural networks have always been popular, particularly in the last two decades, is because they are able to learn how to solve a specific complex problem.
But how a neural network can learn?
To understand it we must reconsider what we said previously. In a supervised context, the neural network tries to learn the parameters of a function h as defined in Equation 2.13 that maps the input to the output. The key idea is that, while the network is learning, every time produces an output we can verify it by comparing it with its "true" output.
By comparing the label produced by the network with the real label we can tell how good was the network prediction. To have a mathematical definition of "good" we need to refer to a cost function l as defined previously in Equation 2.3. Ideally, we would like to set the parameters of the network to minimize the value of the cost function. In other words, given n parameters we would like to find the optimal point of $\mathbb{R}^n$ that minimize the cost given by l.
Once we obtain a value from the cost function l we would like to know how to change the parameters of h in order to reduce the value of l.
Different methods have been proposed in the last century to deal with this matter. Indeed, this is a very general problem called *optimization problem*[116, 149] where we try to minimize a specific function. The most popular method used nowadays is the *gradient descent*[157] technique and

its variations.
In the simplest model we can think of, the perceptron weights update rule should be:

$$\vec{w'} = \vec{w} + \Delta\vec{w} \tag{2.42}$$

where:

$$\Delta\vec{w} = -\eta\nabla h(\vec{w}), \tag{2.43}$$

where η is a positive constant called *learning rate* which determines the size of the step in the gradient descent search. The value of $\nabla h(\vec{w})$ is the following:

$$\nabla h(\vec{w}) \stackrel{\text{def}}{=} \left[\frac{\partial h}{\partial w_0}, \frac{\partial h}{\partial w_1}, \ldots, \frac{\partial h}{\partial w_n}\right]. \tag{2.44}$$

As we can see, to use the gradient descent technique, the cost function h must be differentiable. Moreover, its output must depend only on the neural network output. By having a differentiable cost function we can determine the right way to change the weights to obtain a performance improvement.
The gradient descent technique is widely used in many machine-learning models. It is usually combined with the *backpropagation algorithm*[81, 242, 247], a typical technique that computes how the weights of the network should be changed and how to propagate backward these updates in order to get an improvement. When the gradient descent technique is combined with the backpropagation algorithm they are considered the standard technique to train a neural network. In modern architectures, a different version of gradient descent is used, called *Mini-Batch Gradient Descent*[98, 130]. Usually, with the modern dataset increasing in size, it is not possible nor suitable to update the program's parameter via the gradient descent technique. This is mostly related to the impossibility of loading an entire dataset inside a modern GPU's VRAM. The solution is to consider a small portion of the dataset called batches. Given a single batch, we perform a forward pass through the network and compute the corresponding loss. Then we compute the gradient for this small amount of samples and run the backpropagation algorithm. This approach has different benefits:

- Feasibility: even if we have to compute the backpropagation algorithm for every batch, it is now possible to fit one or more batches in the GPU's memory.
- Regularization: in section 2.7 we will see that a machine learning training process can be problematic when the model starts to fit the noise in the samples we have. Computing the gradient for the whole dataset provides the right direction to move considering also such noise. Instead, with the stochastic gradient descent we are computing an approximation of the gradient, reducing the probability of getting stuck in local minima.

An important aspect of mini-batch gradient descent is the batch size itself. It is natural to ask ourselves what is a sufficient small batch size. There is no precise answer to this question as stated in different works[128, 160]. In modern architectures, this value can vary between a few to hundreds of samples. Nonetheless, due to all the aforementioned reasons, we know that training time per epoch and until convergence increases as the batch size increases. At the same time, the resulting model quality usually gets worse as we enlarge the batch size. An extreme scenario is one

where we compute the gradient and update the weights after every sample. This technique is usually referred to as *Stochastic Gradient Descent*(SGD)[11, 27, 241].

Even if we cannot focus on optimization algorithms it is worth noticing that many of them have been proposed. Their usage is related to the available information about the problem solved via the function to optimize. An important piece of information is whether the objective function can be differentiated. In this sense we can consider the set of first-order optimization algorithms, based on using the first derivative, corresponding to the gradient computed in Equation 2.44 to move in the parameters space. To this category belong: gradient descent[241], momentum gradient descent[229, 276], AdaGrad[61, 175], RMSProp[98, 285] and Adam[115, 133]. Different approaches can be used, such as exploiting the second-order derivative in second-order algorithms, or bracketing algorithms where we know that the optima belongs within a specific range. Choosing an optimization algorithm is fundamental to solving a specific task. It determines how we are going to update the parameters of the function h, providing a different solution that depends on the algorithm and its parameters.

So far we have seen that if we consider a set of layers each one made by a certain number of neurons, the model is able to learn with respect to a certain metric and get better performances. We have seen that such improvement is accomplished by automatically updating the model's parameters. What we did not talk about is how to set up the network itself. Indeed some aspects of the model must be defined apriori. The needed amount of layers and the number of neurons per layer are fundamental to determining the function complexity a model can learn about. Moreover, the weights initialization that can be uniform or random can influence the training process while the activation function determines different values that are passed to the backpropagation algorithm. Also, the learning rate determines the step size in the optimization algorithm and is fundamental in the parameters update procedure. This is a well-known problem called *hyper-parameters tuning*. In subsection 2.1.3 we saw that a dedicated part of the data, called validation set is used to tune the hyper-parameters.
In general, we do not have a precise rule to set such hyper-parameters before the training begins. We usually have rule-of-thumb indications about how to initialize and adjust them subsequently. However, a good way to characterize these aspects is through the concept of *capacity*[16, 48]. As stated in previous works[81]:

> Informally, a model's capacity is its ability to fit a wide variety of functions.

Usually, it is used to describe the complexity of a model. In statistical learning theory, we can find various ways to quantify precisely the capacity of a model. A famous one is the *Vapnik-Chervonenkis dimension*[26, 295], measuring the capacity of a binary classifier. Intuitively the VC-dimension measures the largest possible value e such that a set of e training points can be labeled arbitrarily by the model. Some important results in statistical learning theory[26, 293, 294] show that a classifier error is bounded from above by a quantity that grows with the model capacity and decreases with an increasing amount of training samples.
With such considerations in mind, we can understand how the above-described hyper-parameters influence the model's capacity:

- Layers number: as we increase the number of layers involved in the process, we are increasing the amount of trainable parameters involved in the training process. Therefore, we are increasing the capacity.
- Nodes number: as we increase the number of nodes involved in a layer, we are increasing the amount of trainable parameters involved in the training process. Therefore, we are increasing the capacity.

We can summarize the above information about the relationship between hyper-parameters and the model complexity in Table 2.3

Hyper-parameter	Increases capacity when
#Layers	Increased
#Nodes	Increased

Table 2.3: Relation between different hyper-parameters and capacity

2.5 Convolutional Neural Networks

In the previous section, we introduced a tool that can be used to deal with data by learning the parameters of a transfer function and an activation function. In this section we are going to show a specialized version of a neural network called *convolutional neural network*(CNN)[6, 84, 153, 162]. Historically, the earliest precursor of the Convolutional Neural Network was the Neocognitron[75]. Was introduced in 1982 along with some fundamental concepts that we are going to show in the next few pages such as shared connections and pooling. In the 90s LeNet[154] was introduced for handwritten digit recognition tasks. It was probably the first convolutional architecture as we know them nowadays. The convolutional approach effectiveness was finally proven in 2012[144] by winning one of the most important competitions in the world about image classification, the *ImageNet large scale visual recognition challenge*(ILSVRC).
Nowadays this kind of network is mainly used for visual input and has a grid-like topology. The main difference with the neural network showed previously is that convolutional neural networks replace the general matrix multiplication with a convolution operation[58, 62, 82] in one of the network layers. Such a model is biologically inspired by the visual cortex[166, 231] that does not focus on the whole image. Instead, when we look at our surroundings, we focus on a small portion centered around our pupils. In the last decade, they have been exploited in many different applied fields such as image recognition[95, 237, 288], video recognition[73, 123, 224], medical images[14, 279, 309], recommendation systems[2, 264, 313], image segmentation[15, 190, 273], natural language processing[106, 301]. In all these fields they turned out to be extremely effective.

We can briefly focus on the mathematical convolution operation. In its most general formulation, convolution is an operation on two functions f and g of real values. Intuitively it expresses how the shape of one is modified by the other. It is usually denoted by the symbol $*$ as $f * g$. It is defined as the integral of the product of the two functions after one is reflected on the y-axis and shifted. Formally

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau, \tag{2.45}$$

where t is the amount we shifted our function f. The convolution operation is commutative and the above equation can be rewritten as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau \tag{2.46}$$

meaning that the choice of which function is reflected and shifted before the integral does not change the integral result. Intuitively we are sliding the function $g(t-\tau)$ over the function $f(\tau)$. Indeed, we can consider the two functions $f(\tau)$ and $g(\tau)$, reflect the function $g(\tau)$ on the y-axis, obtaining $g(-\tau)$. Then we can add the offset t and let it start at $-\infty$, sliding all the way to $+\infty$. Wherever the two functions intersect, find the integral of their product. In other words, at value t, we can compute the area under the function $f(\tau)$ weighted by $g(t-\tau)$ In the following, where we work with images, we are going to use the discrete version of the convolution operation, defined as

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{2.47}$$

We saw previously that we are letting the function g slide over the function f. In this sense, when we are dealing with images we are sliding a matrix made by learnable parameters over the input image, computing a point-wise product.

In the previous section, we saw that a neural network can be trained, given a set of input-output pairs. In this sense, an image that is made of pixels with its own value can be considered as a vector of input values. This fact underlines that we could just re-shape our input from a two-dimensional matrix, i.e. the image, to a new kind of input that is simply a one-dimensional vector obtained by concatenating all the rows of the original matrix. This new kind of input could be given to a neural network as showed in the previous section, without the need of introducing a new model like the convolutional one. Such an approach can work with very small and simple images. It can be easily tested that a simple feed-forward fully connected neural network is able to obtain very good results when dealing with simple problems like hand-written character recognition. for instance when working with the MNIST[155] dataset. However, this is explained by two simple shreds of evidence:

- Small image size: the dataset content is made by 28×28 pixel images. The image size plays an important role in the computational cost of the training process. As images are organized as bidimensional matrices, when dealing with the larger size the computational cost scales quadratically.

- Dataset simpleness: the image content for the MNIST dataset is very simple. Every image contains a hand-written digit and the model must classify between the ten different digits. Moreover, because of their simplicity, images do not present complex spatial dependencies and they are composed of simple shapes. We show a sample for every digit in Figure 2.6.

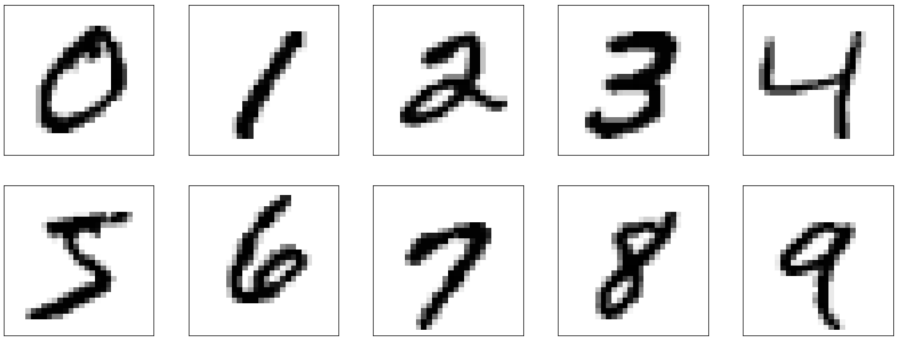


Figure 2.6: Digits samples from the MNIST dataset

More in general, when dealing with a larger input both in terms of image size and data amount, it is nearly impossible to treat efficiently such a volume of data with classical neural networks. With convolutional neural networks, the aim is to reduce drastically the amount of parameters involved. they are mostly based on the shared-weight architecture of the convolution kernels.
A convolutional neural network is made of different layers, and part of them are made by components we saw previously, like the activation function. In the following, we are going to introduce the parts that make the convolutional model more efficient w.r.t. a classical neural network.

CONVOLUTIONAL FILTER: also referred to as *convolutional kernel*, it is defined by a set of learnable weights arranged in a matrix form. The filter is usually smaller than the input size and it convolves, i.e. shifts, over the input image. At every step of the discrete convolution, we perform a Hadamard product[104, 191] between the filter and the input at the current location. After the product is computed we can shift the filter over the image to the next position. The amount of pixels we shift before computing a new Hadamard product is usually referred to as *stride*. Formally, given a matrix K with shape (n, m) where $n, m \in \mathbb{N}$, and a stride $s = 1$, the bi-dimensional discrete convolution output O given by the Hadamard product with the input image I is defined as:

$$O[a, b] = \sum_{i=1}^{n} \sum_{j=1}^{m} K[i, j] I[a - i, b - j] \tag{2.48}$$

where a and b are the output number of rows and columns, respectively. As we were mentioning, the K size is usually smaller than the input size. Typical values for kernel size are $n = 3$ and $m = 3$. Similar values can be also used. We can show in Figure 2.7 a graphical explanation of a 3×3 kernel convolving over a 4×4 image, resulting in a 2×2 output.

Usually, the convolution kernel is coupled with an activation function like the ones described in the previous pages. One thing that is worth noticing is that the size of the output is smaller w.r.t. the input image. We can already guess that by iterating such a process the information amount involved is going to decrease along the network's subsequent layers.
As in fully-connected neural networks, the convolutional ones are usually made by the compo-

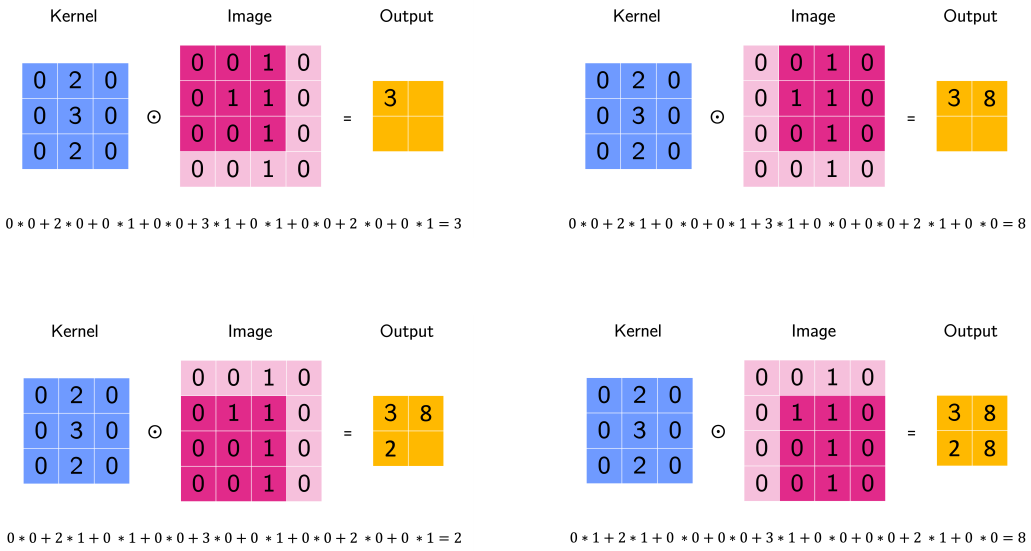


Figure 2.7: A 3×3 kernel convolving over a 4×4 image.

sition of subsequent features map based on linear and non-linear operations, where the linear operations are now convolutions:

$$\Phi_\ell(z) = (\sigma(B_\ell^1 \star z), \ldots, \sigma(B_\ell^{u_l} \star z)) \tag{2.49}$$

and the nonlinear operations include pooling, for instance max pooling, where if $a \in \mathbb{R}^{u_\ell}$, we have

$$\sigma(a) = \max\{\mathrm{ReLU}(a^1), \ldots, \mathrm{ReLU}(a^{u_l})\}. \tag{2.50}$$

In a typical architecture for image classification, a number of convolutional layers are followed by fully connected layers,

$$\Phi(x) = \underbrace{\Phi_L \circ \Phi_{L-1} \circ \ldots \Phi_{C+1}}_{\text{Fully connected layers}} \circ \underbrace{\Phi_C \circ \ldots \circ \Phi_1(x)}_{\text{Convolutional layers}}. \tag{2.51}$$

As we already seen in the previous section, an important concern about the practical usage of convolutional neural networks is related to the hyperparameters of the model. While the parameters of the convolutional model are learnable, some more general features of the network must be defined apriori. One hyper-parameter is the kernel size K. We said previously that a common value for the size of the filter n, m is 3. However, we could choose among a plethora of different values as long as they are smaller than the input size. Something similar can be said for the amount of filters used in every layer or the number of layers involved in the model. At the same time, even if we did not focus on the stride, it is another hyper-parameter that can influence the learning process.

- Filter size: as we increase the filter size in the nodes, we are increasing the area where the filter is looking for visual dependencies. Taking into account a larger portion of the image

increases the ability of the model to find such dependencies. Therefore, it increases the capacity.

- Stride: as we increase the stride of the model, we are shifting "faster" on the input image. Such operation force the model to perform a lower number of analysis step on the input image. Therefore it decreases the capacity.

Again we can summarize the above information about the relationship between hyper-parameters and the model complexity in Table 2.4

Hyper-parameter	Increases capacity when
Filter size	Increased
Stride	Decreased

Table 2.4: Relation between different hyper-parameters and capacity

POOLING NODE: the second component that reduces massively the number of trainable parameters involved in the learning process is the pooling node[79, 144, 323]. It is not made by parameters that we can learn during the training process, but simply reduces the dimension of data by combining small square clusters into one value. A hyperparameter involved in the process is the tile dimension to analyze. Formally, given $t_s \in \mathbb{N}$ and an input with size $d_1, d_2 \in \mathbb{N}$ the pooling node divide its input into $(\lceil \frac{d_1}{t_s} \rceil, \lceil \frac{d_2}{t_s} \rceil)$ tiles, computing an output for each of them. The output value can be produced in two different ways depending on the pooling type we use:

- Max pooling: given a tile T of dimension (t_s, t_s) the max pooling operation is computed as
$$p_{max}(T) = \max(T) \tag{2.52}$$
selecting the maximum value in the tile as node output.

- Average pooling: given a tile T of dimension (t_s, t_s) the average pooling operation is computed as
$$p_{avg} = \frac{1}{t_s^2} \sum_{i=1}^{t_s} \sum_{j=1}^{t_s} T_{ij} \tag{2.53}$$
selecting as node output the average value between the ones included in the tile.

We show these two approaches in Figure 2.8. Using a pooling node presents at least two main advantages. First, as we were mentioning, the input size decreases drastically. Even with the minimum pooling tile size, i.e. 2, every time the input passes through a pooling node its dimension is decreased by 4 times. Even if the typical size for the pooling tile size is precisely 2, with a larger value the reduction could be even greater. The second important advantage of pooling is the increase in the model invariance to small perturbations. Both max pooling and average pooling provide similar outputs when the input change slightly. That makes the model more robust to small perturbations in the data.

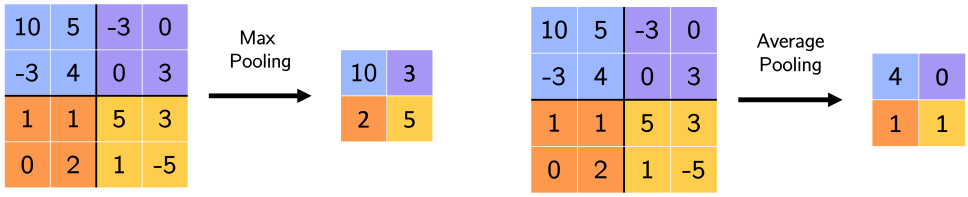


Figure 2.8: Max pooling and average pooling examples.

Similarly to previous considerations, also the pooling node involves hyper-parameters that must be set before the training, such as the tile dimension and the pooling type. As we were mentioning, the tile size is usually $t_s = 2$. On the pooling type the most common choice is the use of max pooling that is more perturbation resistant w.r.t. the average pooling node.
Different studies in the last few years tried to propose strategies and improvements to the pooling node[42, 274] and its role inside the neural model[253, 314].

2.6 Transformers

In the previous section, we introduced a popular deep learning model, named Convolutional Neural Networks, that have been the most common tool to deal with images in the last decade. Their success has been proven in various scenarios, outperforming previous approaches. Many variations and modifications have been proposed and a new methodology is emerging as state-of-the-art. Even if we are going to analyze the transformers for the computer vision context, historically they have been strongly tied to the Natural Language Processing(NLP) context.
Natural Language Processing is a subfield of linguistics, computer science, and artificial intelligence. Its main purpose is to study how programs can compute and analyze large amounts of natural language data. To deal with this problem many deep models have been introduced in the last twenty years. The first models were probably Recurrent Neural Network(RNN)[167, 244, 249] along with Hopfield networks[103] and Long Short-Term Memory(LSTM)[83, 101] networks. Between 2017 and 2018 some cornerstone papers were published introducing the concept of *attention*[297] and *Transformer*[251]. Between 2018 and 2020 many implementations and improvements have been proposed such as ELMo[220], BERT[125], GPT[232] and GPT3[30] to cope with NLP problems. Since 2020 transformers have been applied also to vision classification tasks via the Vision Transformer(ViT)[60] and broader vision tasks in general[129, 210]. Different variations have been proposed[23, 170]. More in general, transformers have been applied to different research fields such as chemistry[259], life sciences[240] and audio processing[59]. Moreover, different studies tried to embed a transformer backend to previous convolutional models[308].

Transformers for NLP tasks: to understand how a transformer works we first need to introduce the aforementioned concept of attention. Previous models such as LSTM tried to implement techniques to detect long-term dependencies. With the attention mechanism, we can catch extremely long and effective dependencies. Indeed the reference window with the attention

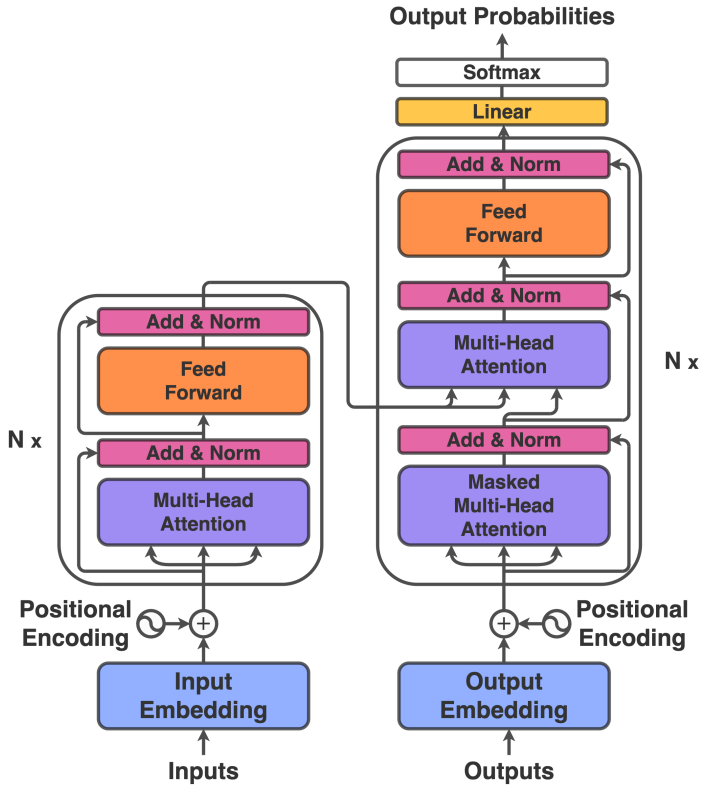


Figure 2.9: Transformer architecture as presented by Vaswani et al.[297].

mechanism is potentially infinite, allowing the model to detect dependencies from the whole context. In a nutshell, the attention mechanism analyzes an input sequence and evaluates at each step which other parts of the sequence are relevant to it.
To intuitively get what the attention module does, suppose to have the n-th new input that after some steps are encoded into a vector v_n. Now we can compute all the scalar products $v_1 \cdot v_n$, $v_2 \cdot v_n, \ldots, v_{n-1} \cdot v_n$ providing us n scalars encoding how similar, i.e. how relevant, each previous input was w.r.t. the new one.

The attention component is inserted in a larger model that we show in Figure 2.9. Such a model is composed of two different parts: an encoder and a decoder. Each component both in the encoder and in the decoder can be stacked multiple times on the same component to increase the model capacity.
The encoder's duty is to map each input to a representation containing the whole information about the sequence is analyzing. First, the n inputs are passed through an input embedding layer. The purpose is to use a pre-train embedding layer to map them to a vector of dimension d. Then, a positional encoding layer must be introduced as transformers do not have recurrence or convolution. To accomplish this task, information about the position is injected via positional encoding.

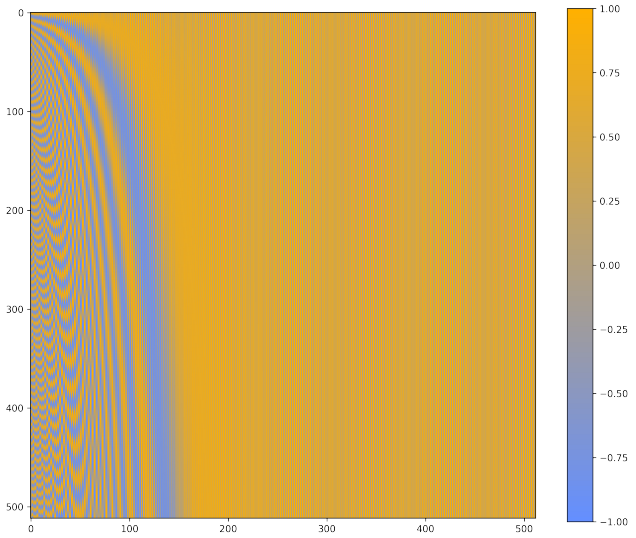


Figure 2.10: The positional encoding matrix with size $d = 512$, the same as in the original paper

In the original work about transformers sine and cosine functions with different frequencies have been used to define a positional encoding matrix PE:

$$PE_{(p,2i)} = sin\Big(\frac{p}{10000^{2i/d}}\Big) \tag{2.54}$$

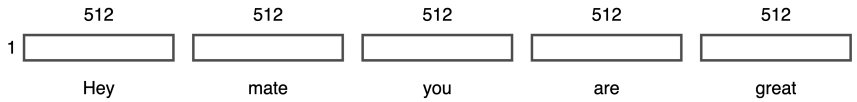
$$PE_{(p,2i+1)} = cos\Big(\frac{p}{10000^{2i/d}}\Big) \tag{2.55}$$

where p is the p-th row and i is the i-th column of the matrix. However, only the first d columns are considered as the p-th input token from the input embedding is summed up to the p-th row. The sine and cosine wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$ and, as for any fixed offset o the $PE_{p,o}$ can be represented as a linear combination of PE_p the relative position encoding is easy to learn for the model. We can show the positional encoding matrix content in Figure 2.10.

After both embedding and positional encoding are applied to the inputs, we can pass them through the multi-head attention layer. The name multi-head is related to the presence of h distinct "heads" computing in parallel. Before feeding the input to the multi-head attention module three matrices are introduced: the *Query* Q, the *Key* K, and the *Value* V. Such matrices are made by learnable parameters and all of them have dimension $d \times d$. Then we multiply every token for the three matrices, obtaining 3 vectors of dimension $1 \times d$. Now we split every vector into h parts and pass every part to one of the heads. So far we projected the n split embedding into h subspaces. We can focus on just one of them. Each head receives $3 \cdot n$ input vectors, one for every key, query, and value result for every input. Each vector has dimension $1 \times d/h$. Now we compute the attention by computing a dot product between each query vector and each key vector received by the head. We end up with a measure of similarity between each query and each key. Therefore the output of the attention module is just a $n \times n$ matrix QK encoding how related each query

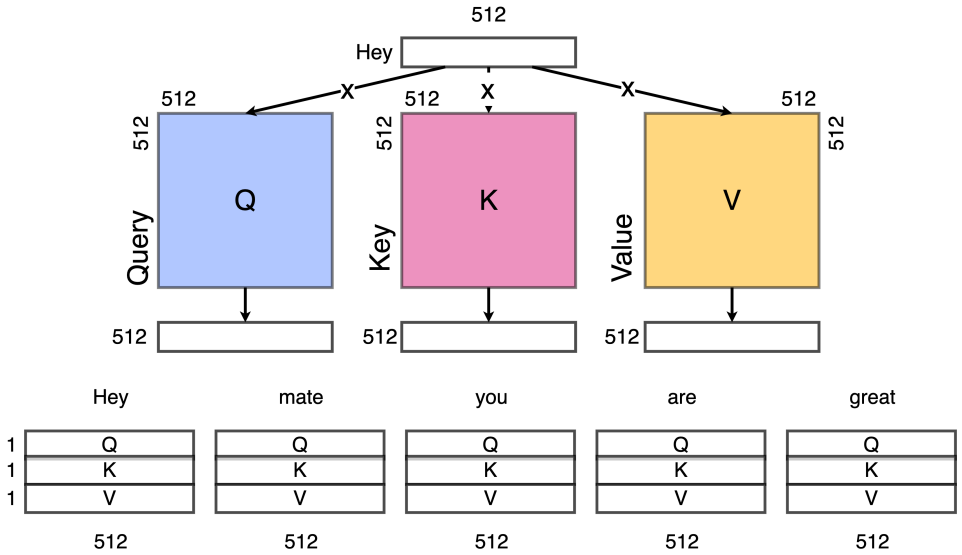
is to each other key. Then we multiply each value vector for the corresponding result in the QK matrix. Then we can normalize each output, dividing by $\sqrt{d/h}$ and passing each matrix row to a softmax layer. We can then sum up such results and obtain a $1 \times d/h$ vector and concatenate it with the output of every head, obtaining a $1 \times d$ vector. At this point, we have n vectors with size $1 \times d$. Then we can use the *output* matrix with dimension $d \times d$ learnable parameters. By going for a point-wise multiplication of each vector for the output value we obtain again n vectors with size $1 \times d$. This is the output coming from the multi-head attention module. To such output, we add the input-positioned embedding and we normalize it. Then it is given as input to a feed-forward model that is just a fully connected network as shown in section 2.4.
It is worth noticing that in the encoder part more than one subsequent encoder can be stacked. After one or more encoders, the input can be passed to the decoder.
The decoder is similar to the encoder except for the first module which is a masked multi-head attention. In this module all the multiplication between a query and its subsequent values are set to $-\infty$ so that softmax will provide a 0 as output, meaning that a token cannot use information about the subsequent tokens because is what the transformer is going to predict at test time. The remaining part of the model is made by modules we already analyzed.

To better understand the transformer functioning we provide an example of going through the encoder module. Suppose that we want to translate the following sentence "Hey mate you are great". The first step is to go through the input embedding layer. In the original paper, the embedding output size is 512. The second step consists in adding a marker related to the position via the positional encoding module. After we pass the five tokens through the input embedding and the positional encoding we obtain 5 tokens with 1×512 size.

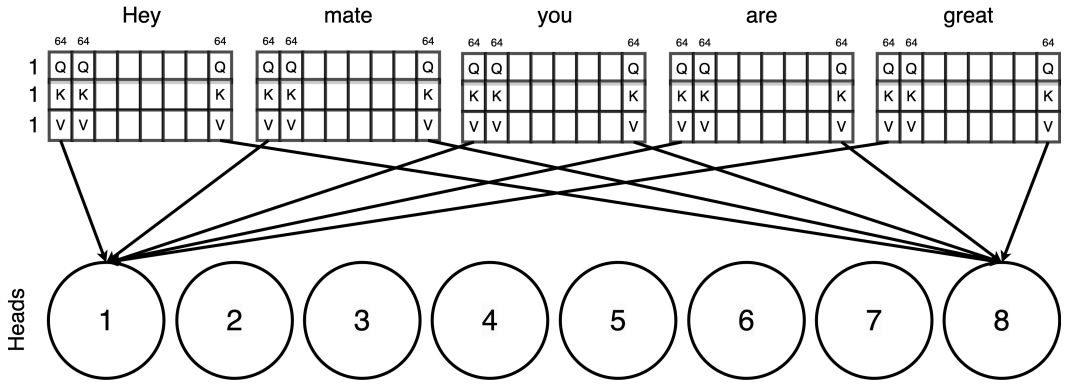


Then we introduce the three learnable matrices, query, key, and value. In order to provide an intuition on why we need such matrices, we have to understand that such terminology is coming from the information retrieval context. Usually, when we interact with a program looking up information like a search engine we provide a set of keywords we are interested in, corresponding to the query. Then we map the query to the set of possible information, corresponding to the keys. Lastly, we are provided a set of results, corresponding to the values. In the original paper about transformers, each matrix has a size equal to the input dimension on both axes, i.e. 512×512. We can now compute a point-wise multiplication between a token and each matrix Q, K, and V. For every point-wise multiplication, we obtain a 1×512
We introduced a set of 3 learnable matrices, query key, and value, and we multiplied each token for each matrix. In the original sentence, we had 5 in different tokens. This way the output will be a set of $3 \cdot 5 = 15$ vectors having each one 1×512 size.

Now we can split each vector into h parts to pass them properly to the h heads of the multi-head attention part. In the original paper $h = 8$. This way we obtain a new subset of 120 vectors each one with size $1 \times 512/8 = 1 \times 64$. The first head receives all the first segments of every token. The second head receives all the second segments of every token. In general, the i-th head receives

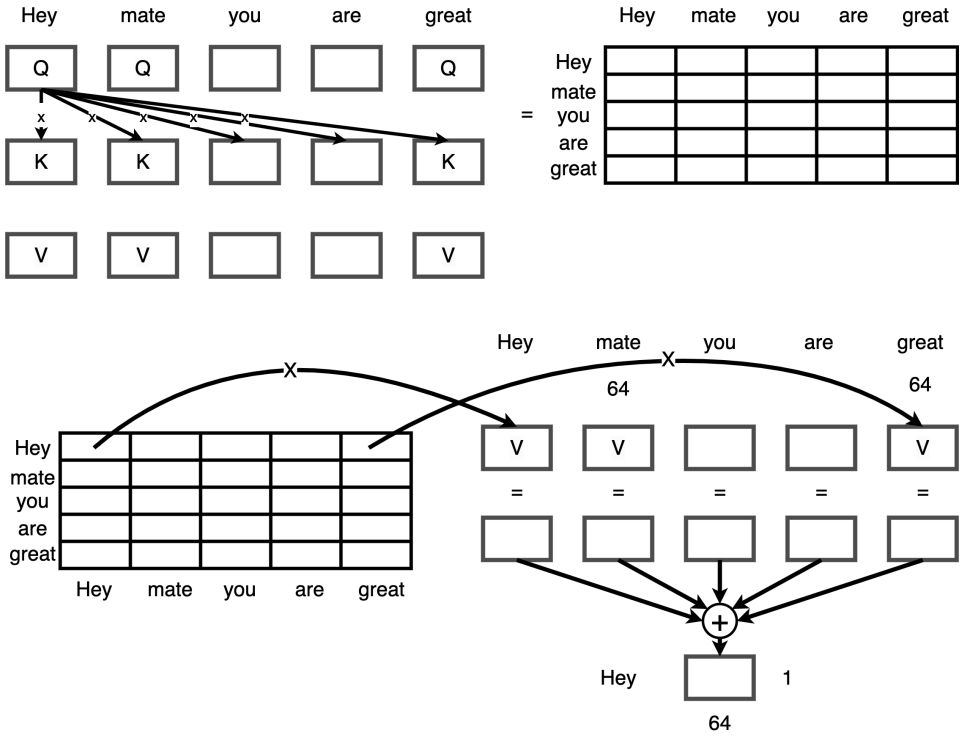


all the i-th subpart of the original tokens multiplied by Q, K, and V. This way the computation about each token can be parallelized, providing in addition, more stable results.

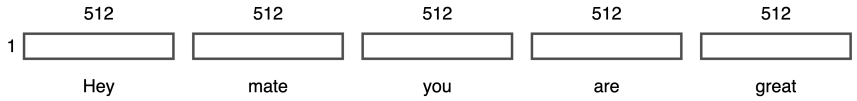


Now we can focus on the first head, keeping in mind that the process we are going to show is identical in every head of the model. Every head receives, for every token, 3 vectors related to Q, K and V with size 1×64. Now we can compute the dot product between every query vector and every key vector. As we have one query vector and one key vector for every token the result is the QK matrix with size 5×5. Next we can normalize each element in the QK matrix, dividing by $\sqrt{d/h} = \sqrt{512/8} = 8$. Then we can pass each row of the QK matrix to a softmax layer. As we worked on every cell or on every row without creating new matrices, the matrix size output is still 5×5.

Given the QK matrix, if we consider the i-th token we can compute the product between all the elements in the i-th of the QK matrix with the corresponding value tokens. Then we can sum them up, resulting in how much the i-th token is related to the other tokens. The output for every token is a vector with size 1×64.



As we do this for every token we obtain 5 tokens with size 1×64. This is going to be the output of a single-head in the multi-head attention. Now we can remember that in the original paper, the multi-head attention was composed of 8 distinct heads. Therefore we can zoom out from the single head and, for every token, we concatenate again the 8 output with 1×64 coming from the 8 different heads. The result for every token is again a 1×512 vector.



The last step consists in introducing the output matrix. In the original paper such a matrix has dimension 512×512. Then we can compute the product between each vector related to the tokens obtained in the previous step and the output matrix. The output is again, for every token, a 1×512 vector.

This is the final output of the multi-head attention and it is ready to be passed to the addition and normalize module before being passed to the subsequent feed-forward module.
As we were mentioning, the output of the encoder module can be either passed to the decoder or go through another encoder module. In the original paper, 6 encoder layers are stacked. Also in the decoder, several layers can be inserted one after another. In the original paper, 6 decoder layers are stacked.

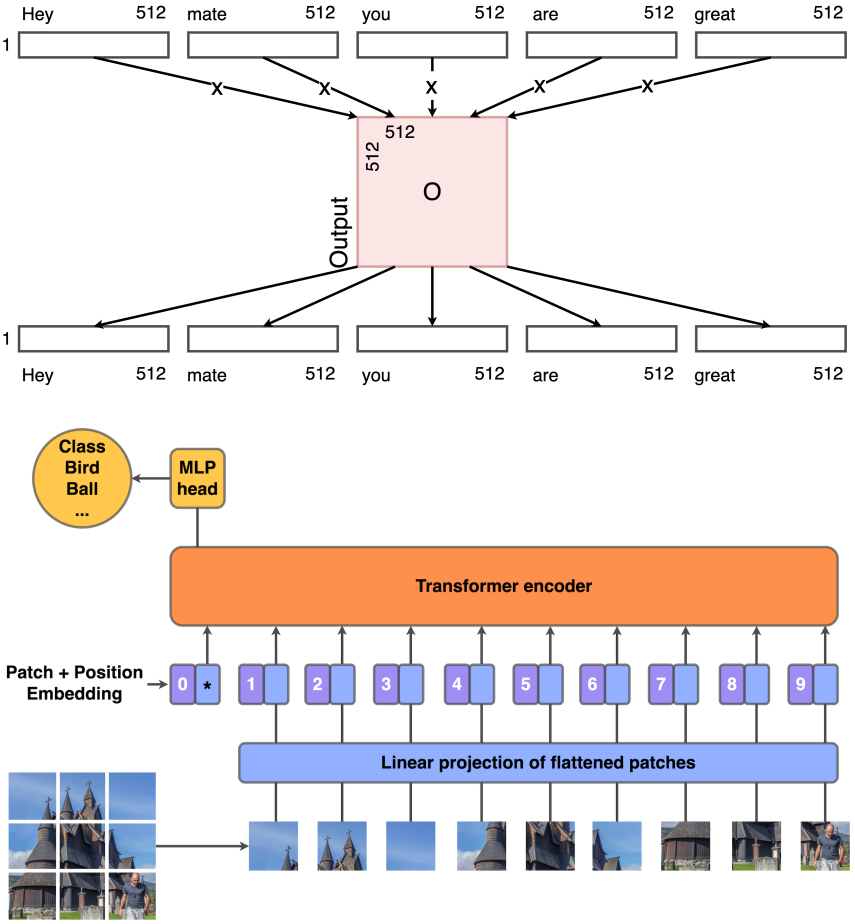


Figure 2.11: The overall architecture of the Vision Transformer.

TRANSFORMERS FOR VISION TASKS: in the last few pages we saw how a transformer can be used to deal with Natural Language Processing problems. Since 2020 the transformer's idea has been applied also to computer vision tasks. In the crucial work from Dosovitskiy et al[60] proved that the transformer architecture can be applied to image classification, paving the way for more recent advances in the computer vision community.
We saw that the basic idea about training a transformer is to consider a phrase as a set of words and evaluate their relationship via the attention mechanism. Indeed, with the vision transformer, we have a similar process. Instead of having a phrase we have an image and we can split it into patches, where each patch represents a single word. Once we have the "words" we can process them as we showed for the NLP context. We can show the overall architecture in Figure 2.11

A single image is split into patches and proceeds through the layer producing its flattened version. It is worth noticing that the "image" input is now indistinguishable from a word encoded through the input embedding layer of the original transformer implementation for NLP. The only main difference with the original paper about the transformer is in the extra learnable token that is used as the first token in the procedure. The main purpose of such tokens, originally intro-

duced by Devlin et al[125] is to aggregate the information learned from the other tokens. Indeed we can see that is then used in the multi-layer perceptron head to provide classification results. Also in the vision transformer model, more encoders can be stacked one after another. Three different versions are proposed in the original paper, base, large and huge, depending on how many encoder layers are used and how many heads are inside every encoder.

2.7 Fitting & Regularization

A common problem in machine learning we have to deal with is about *regularization*. To better understand this key concept we must introduce the notion of *overfitting*. According to the English Oxford dictionary, the statistical definition of overfitting is

> **O·ver·fit·ting**, ['əʊvər'fɪtɪŋ]noun: in statistics, the production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.

This situation can happen when a set of points is fitted with a model that has more parameters than needed. Intuitively, when the model is fitting precisely the training set, it will not be able to treat correctly new samples as it lacking on the ability to generalize. This situation is the opposite w.r.t. *underfitting*. When a model is underfitting over a set of data means that it does not have enough parameters to adapt itself to the problem. In that context, a more complex model is needed to understand the relationship between data. The two-folded problem of underfitting/overfitting is strictly related to the capacity concept we introduced in the previous pages.
A simple problem to better understand the overfitting concept is about fitting a set of points with a function. Suppose we are provided a set of m points as we show them in Figure 2.12(Top-Left). We can intuitively see that such points are generated accordingly to a parabola with some noise. We show a good fitting function in Figure 2.12(Top-Right). Notice that usually, we do not know this information. We could fit the training points with a polynomial, defined as:

$$p_n(x) = \sum_{k=0}^{n} a_k x^k + b \tag{2.56}$$

where k is the polynomial degree. If we fit the training points with a polynomial, a hyper-parameter we must set is the polynomial degree. The two opposite scenarios of underfitting and overfitting are the following. We could try to fit our training samples with a 0 degree polynomial: $p_0(x) = \sum_{k=0}^{0} a_k x^k + b = b$, corresponding to the horizontal line at height b. Clearly, this is a poor interpretation of our data as we are trying to model a set of points intuitively distributed as a parabola with a straight, horizontal line. This situation corresponds to the underfitting scenario where our model has only one parameter to set, i.e. b, while it should need more. We show this situation in Figure 2.12(Bottom-Left). The opposite situation is about using a polynomial with degree m, equal to the number of points, resulting in $p_m(x) = \sum_{k=0}^{m} a_k x^k + b$. Such a situation, where the polynomial is going to fit perfectly the training set, corresponds to the overfitting scenario. The model is too complex as it is trying to interpret the data with a polynomial with a high degree while we know that such points are generated according to a parabola. The problem

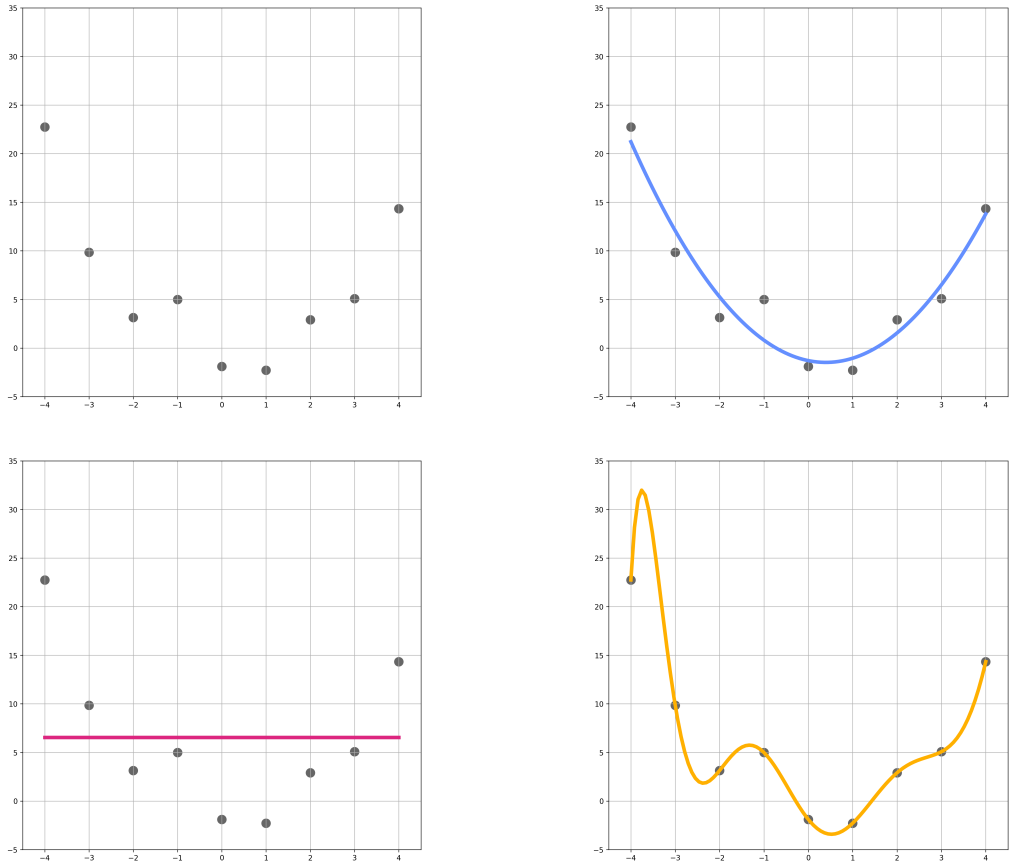


Figure 2.12: (Top-Left): training set points generated according to a noisy polynomial of degree 2 function. (Top-Right): polynomial of degree 2 fitting the training set. (Bottom-Left): polynomial of degree 0 fitting the training set, incurring in an underfitting situation (Bottom-Right): polynomial of degree 9 fitting the training set, incurring in an overfitting situation

with overfitting is that while the model will fit perfectly the training set when dealing with new points will provide a wrong interpretation given by the too-complex model that was trained with. We show this scenario in Figure 2.12(Bottom-Right).

Finding a good compromise about model capacity can be a hard task to accomplish and in general, we saw that too few tunable parameters in the model lead to an underfitting situation. Instead, having too many parameters can result in an overfitting scenario. This problem, well known in machine learning, it is usually referred to as bias-variance tradeoff or *bias-variance dilemma*[18, 57, 78]. Bias and variance are two out of three prediction error sources, while the last one is the irreducible error due to inherent randomness or feature set incompleteness. Bias usually measures how far off, in general, predictions made by a model are distant from the correct value. A simple machine learning model will have a large bias error as it does not matter how much data we collect, the model does not have enough flexibility to deal with them. That is the case of Figure 2.12(Bottom-Left) where the simple polynomial could only move the bias, and will never

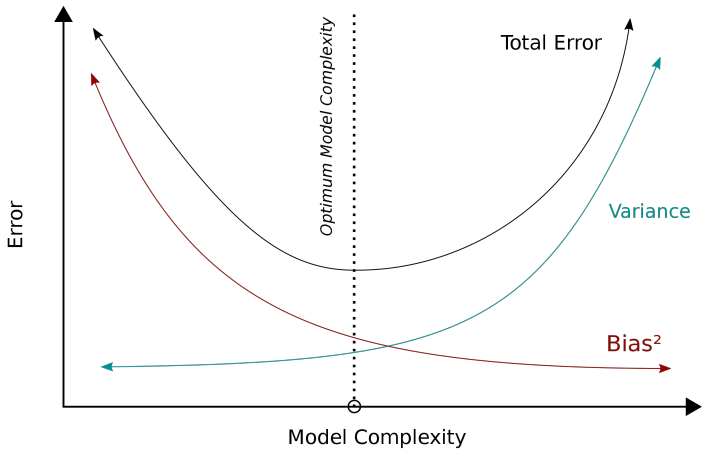


Figure 2.13: The bias-variance dilemma. A too-simple function will provide a high bias error while a too-complex function will provide a high variance error

adapt to new input. Instead, the variance error is the variability of a model prediction for a given data point. Usually, we have a large variance term when we overfit our data. Indeed in this situation, a model with many parameters can adapt precisely to the training set, fitting also the noise coming from the data. That is the case of Figure 2.12(Bottom-Right) where the complex model adapts perfectly to the training set.
However, it is worth noticing that these two kinds of error are mathematically tied. Suppose to have the function f, data generation source, and its estimate $\hat{f}$. In this situation, we have that:

$$
\begin{aligned}
&E[(f(x) - \hat{f}(x))^2] = E[f(x)^2 - 2f(x)\hat{f}(x) + \hat{f}(x)^2] = \\
&= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + E[\hat{f}(x)]^2 = \\
&= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + f(x)^2 - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)]^2 = \\
&= E[(\hat{f}(x) - E[\hat{f}(x)])]^2 + (f(x) - E[\hat{f}(x)])^2 = Var(\hat{f}(x)) + Bias^2(\hat{f}(x))
\end{aligned}
$$

resulting in:

$$E[(f(x) - \hat{f}(x))^2] = Var(\hat{f}(x)) + Bias^2(\hat{f}(x)) \tag{2.57}$$

Stating that on average the error committed by estimating f with $\hat{f}$ is the sum of variance and bias squared. We can show such important results in Figure 2.13. The total error, represented by the black line, is the summation of variance and squared bias. While we increase the model complexity/capacity on the x-axis the bias error decreases while the variance increases. Finding a good trade-off between bias and variance via an "optimal" model complexity results in the minimization of the total error.

In the last few pages, we saw that both overfitting and underfitting can be problematic to treat and so far the only tool we introduce to tune our model capacity is the number of parameters in-

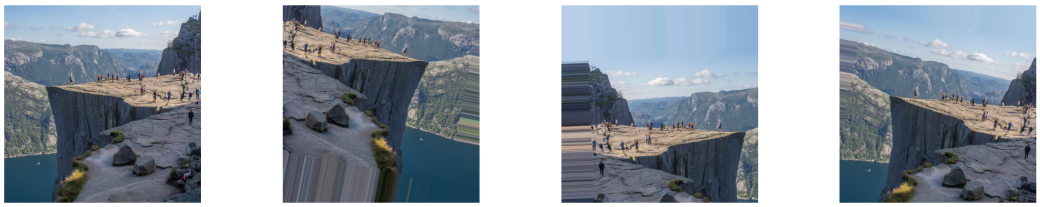


Figure 2.14: Instance of image augmentation. On the left, we have the original image. The other three images are possible augmented images.

volved in the model. In the next pages we are going to introduce different ways to tackle the most problematic situation, the overfitting one:

- Having more data: remembering what we said before, a model that is overfitting, has a parameter number that is excessive w.r.t. the available amount of data. In contexts where we can retrieve more data easily, like the ones when data are synthetically generated, a simple solution to alleviate the overfitting is to collect new data about the problem we are studying. Every data point can provide new information about the problem, forcing the model to adapt and learn a more general mapping function. This is particularly true when dealing with images as every single image carries a lot of information via its three-channel inputs.

- Data augmentation: in some scenarios, it is not possible to retrieve new data for our model. A plausible solution to such a problem is about augmenting synthetically the available set of data. The idea of this process is to apply an affine transformation to our input data in order to generate new data that could be useful for the model to learn. Given an input x and an affine transformation $\phi(\cdot)$:

$$x_{aug} = \phi(x) \tag{2.58}$$

  In the image context, some examples of possible $\phi(\cdot)$ functions can be: reflection, rotation, scaling, and translation. More than one transformation can be applied at the same time to increase the variability of the input. Moreover, we can have other image modifications such as brightness and shear. We can show some typical augmentation examples in Figure 2.14

- Regularization: suppose that we have a set of points X and we want to map them to the label space Y. We have seen that, given a model with parameters $\hat{w}$ and a performance measures function like the square loss defined in Equation 2.4, we can minimize the distance between the proposed solution and the real one. In practice, we want to minimize the distance between Y and the input multiplied by the available parameters $X\hat{w}$. Formally:

$$\hat{w} = \min_{\hat{w}} ||X\hat{w} - Y||_2^2 \tag{2.59}$$

  by finding the set of parameters $\hat{w}$ such that the distance between Y and $X\hat{w}$ is minimized, we are looking for the "best" model that can fit our data. The regularization technique

consists in introducing a slight but fundamental modification to the above minimization problem:

$$\hat{w} = \min_{\hat{w}} ||X\hat{w} - Y||_2^2 + \lambda ||\hat{w}||_k^2 \tag{2.60}$$

in this new minimization problem, we are also trying to minimize the norm of the parameters vector. Depending on the norm k we use, different results and names are given to the optimization problem:

$$\hat{w} = \min_{\hat{w}} ||X\hat{w} - Y||_2^2 + \lambda ||\hat{w}||_1^2 \tag{2.61}$$

$$\hat{w} = \min_{\hat{w}} ||X\hat{w} - Y||_2^2 + \lambda ||\hat{w}||_2^2 \tag{2.62}$$

The first optimization problem is usually referred to as *Lasso regularization* technique. It was originally introduced for geophysics problems[250] and named officially after a regression study by Tibshirani[284]. The second one was introduced in the 1970s and it was named *Ridge regularization*[96, 102] or *Tikhonov regularization*. The reason to minimize the norm of the vector $\hat{w}$ is two-folded. Philosophically, models with smaller weights correspond to simpler solutions. Forcing the model's weights to be in a smaller range does not allow to have complicated solutions perfectly fitting the training data. Mathematically, our data inherently do not represent perfectly the training distribution and they can be prone to a small error E. In this sense, we have that

$$(X + E)\hat{w} = X\hat{w} + E\hat{w} \tag{2.63}$$

we can notice that having a smaller $\hat{w}$ will also reduce the propagation of the error E.
Lasso and Ridge regularization techniques are different one from another and we can understand intuitively what is the effect of using one technique or the other. Both methodologies try to minimize the value of the weights. The main difference is that the Ridge regression technique, when the value of the weights is between 0 and 1 does not have huge benefits to force them going to zero. Instead, the lasso regularization cost is linear even when the values are between 0 and 1. In this sense Lasso will try to force as most as possible weights to 0, providing more sparse solutions.

- Early stopping: when defining overfitting, we saw that such a situation can happen whether our model m starts to fit the noise in the n data points. We also saw that usually this means a perfect fit on our training data as shown in Figure 2.12(Bottom-Right). It is easy to understand that fitting perfectly the training points correspond to a training error:
$\hat{L}(m, f) = \frac{1}{n}\sum_{i=1}^{n} l(m(x_i), f(x_i)) = 0$
In a more complex model, the convergence to zero error on the training set happens gradually. Usually, to evaluate the goodness of our model we test it on a different set of data called *validation set*. This way we understand how well the model generalizes to new data. The *early stopping* technique[228, 312] consists in halting the training procedure when the error committed on the validation set starts to grow. Indeed we cannot use the training loss to evaluate the overfitting situation as the model will learn more and more about the training data trying to minimize the loss of such data.

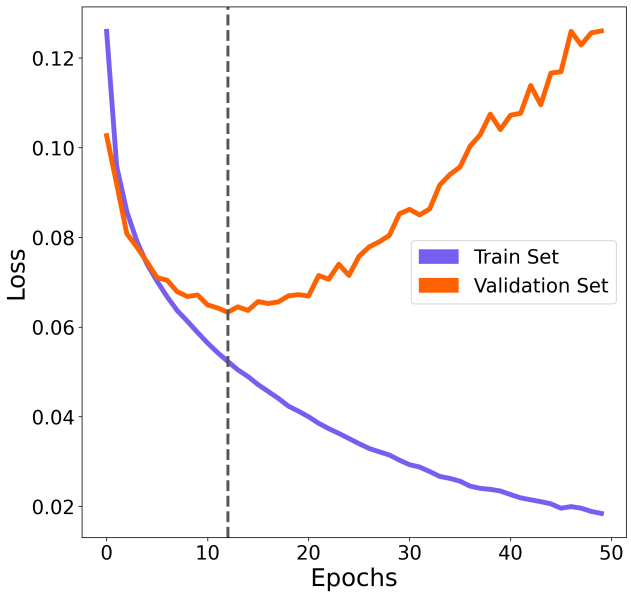


Figure 2.15: A common overfitting situation. The dataset used in this training process is the CIFAR10 dataset. The training loss is decreasing over epochs while the validation one reaches a minimum before starting to increase. According to early stopping, the model's parameters stored are the ones computed at epoch 13 where the validation error reaches its minimum.

From a practical point of view, most of the models used nowadays such as neural networks and transformers learn iteratively over time. The early stopping force the training to stop the training and save the model's parameters when a promising local minimum is reached. This situation is shown in Figure 2.15. The image shows the loss obtained by training a convolutional model on the popular dataset CIFAR10[142].

We can see that the training process through the epochs leads to a monotonic error decrease for the training set. Instead, the validation set error is at first comparable with the training set error. At epoch 13 the validation error reaches a minimum, marked by the dotted line in Figure 2.15. After epoch 13 the validation error starts increasing over the epochs. This is a clear overfitting phenomenon. The early stopping criterium would store the model's parameter computed during epoch 13. Such parameters are apparently the best ones we can use to generalize on unseen samples.

The regularization techniques we have seen in the previous pages like Lasso and Ridge operate directly on loss function to encourage the model to provide simpler solutions. Such approaches are referred to as *explicit regularization*. The early stopping technique does not influence directly the considered loss function. Instead, it studies the model's behavior during the training and selects a-posteriori the best parameters configuration. Such technique is usually referred to as *implicit regularization*.

- Dropout: the last technique we introduce to prevent overfitting is called *dropout*. Introduced in 2012 by Hinton et al.[100], it is a very popular solution to regularize neural models by preventing complex co-adaptations on training data. Indeed a common issue with large neural networks is co-adaptation where all the weights are learned together and it some connections will have more predictive capability than others. Usually, as the network is trained iteratively, connections with more predictive capability are learned more and more while the weaker ones become less important at each iteration.
  The main idea about dropout is to ignore some nodes in a layer during the training process. The hyper-parameter we need to set to perform such operations is usually the amount, i.e. the percentage p_d, of nodes we want to shut down during the training, along with all their incoming and outgoing connections. This operation in neural models involves a single layer. Therefore we can perform it differently on each layer. Usually, epoch after epoch, the set of neurons that are shut down, changes. This way, by not considering the output coming from a certain amount of neurons, i.e. by "dropping out" a subset of neurons, we are making the model more robust. Dropout simulates a sparse activation in the model coming from a certain layer, encouraging the network to learn a sparse representation as a side effect. This should be guaranteed by the fact that the model is presented with a different piece of information about the data in every epoch. Formally, given a dataset X with labels Y, a simple single layer network having n neurons and a p_d amount of dropout minimizes the following quantity:

$$\hat{L}_{dr}(X, Y) = l\left(Y, \sum_{i=1}^{n} \delta_i (1 - p_d) w_i X_i\right) \tag{2.64}$$

  where $\delta_i(1-p_d)$ is a Bernoulli function with parameter p_d, meaning that its value is 0 with probability p_d, and 1 otherwise. If we compute the gradient of $\hat{L}$ along every w_i direction we have:

$$\frac{\partial \hat{L}_{dr}}{\partial w_i} = -Y \delta_i X_i + w_i \delta_i^2 X_i^2 + \sum_{j=1, j \neq i}^{n} w_j \delta_i \delta_j X_i X_j \tag{2.65}$$

  It is possible to show that the empirical loss $\hat{L}_{dr}$ of such network with dropout has a relationship with the loss $\hat{L}$ of a neural network without dropout as:

$$\mathbb{E}\left(\frac{\partial \hat{L}_{dr}}{\partial w_i}\right) = \frac{\partial \hat{L}}{\partial w_i} + w_i p_d (1 - p_d) X_i^2 \tag{2.66}$$

  stating that the expectation of the gradient when we introduce dropout is equal to the gradient of the regularized regular network when $w' = (1 - p_d)w$. We saw that an overfitting situation happens when, given a set of data, we try to map the input to the output with a model that has too much capacity. In this sense, when a layer is forced to drop some of its neurons, its computation capacity is reduced.
  The dropout capacity reduction is applied only at training time. Once the model is done

with learning, we can restore normal connectivity between nodes in different layers and test the model on unseen data with its full capacity.

Part II

Machine learning efficiency

3 Training time efficiency

In the previous chapter, we introduced a set of important machine learning notions. Through the basic ingredients of machine learning such as performance measure, task, and experience we were able to properly define what a model is. We also saw that in the machine learning field, we can have a wide plethora of different models. We introduced some of them such as neural networks and transformers. The last decade has seen a massive increase in neural model usage, both in academic and company contexts, showing outstanding results. In this sense, the impressive performances of deep learning architectures are associated with a massive increase in model complexity. Millions of parameters need to be tuned, with training and inference time scaling accordingly. But is massive fine-tuning necessary?
In this chapter, we are going to focus on image classification, considering a simple transfer learning approach that exploits pre-trained convolutional features as input for a fast kernel method.
We refer to this approach as *top-tuning* since only the kernel classifier is trained on the target dataset. By performing more than 3000 training processes involving 32 different target datasets, we show that this top-tuning approach provides comparable accuracy w.r.t. fine-tuning, with a training time that is between one and two orders of magnitude smaller.
These results suggest that top-tuning provides a useful alternative to fine-tuning in small/medium datasets, especially when training efficiency is crucial.
This chapter is organized as follows: section 3.3 presents an account of the relevant background while section 3.4 reports details on our methodology. We also present the hyper-parameters configuration and provide details on the datasets involved in the experiments. In section 3.5, we illustrate the results of our empirical analysis. Finally, section 3.6 is left to concluding remarks.

3.1 Motivations

In the last decade, deep learning has led to unprecedented successes in computer vision, at par with human performances in several tasks[144, 207, 299]. In particular, Convolutional Neural Networks (CNNs)[84, 237] proved successful in a wide range of domains[162], from medical images[14, 161, 195, 279, 310] to robotics [7, 31, 146, 199] and cyber-security[163, 322], to name a few examples.
These advances are related to a frenetic increase in model complexity[87, 151, 278, 297, 318], demand for data[84] and corresponding growth of computations[33]. Several solutions have been proposed to alleviate the need for labeled data, from few-shot learning[266, 302] to self-supervision techniques[55, 80, 315], and also to reduce the inference time, e.g. via pruning techniques [159, 179, 238].
Instead, on the computational resources needed during the training process, we are observing a visible tendency towards deeper and wider models. Such a trend can be problematic when re-

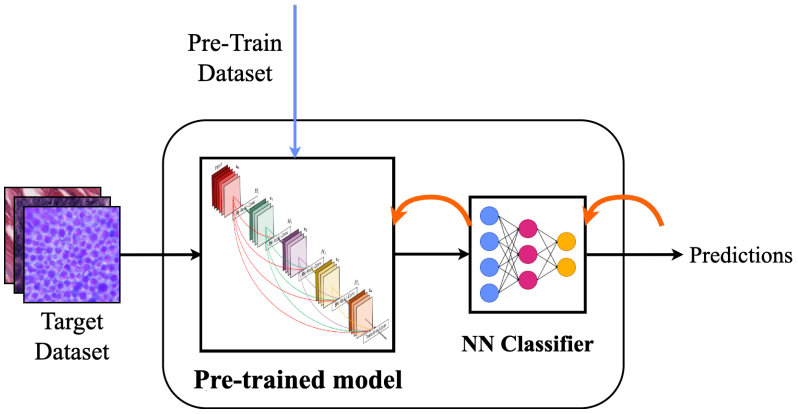


Figure 3.1: The *Fine-Tuning* pipeline. All the model weights are updated.

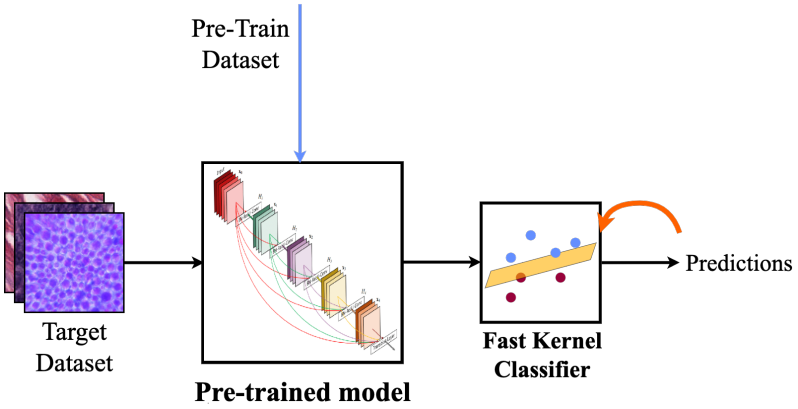


Figure 3.2: The *Top-tuning* pipeline. Only the fast kernel weights are updated.

sources are budgeted, as training models from scratch can be prohibitive[158, 272].
Different contributions can be found in literature, from replacing the fully connected layers with a more efficient component[311] to combining convolutional networks with gaussian process[287] or with kernels[306].
Another solution is transfer learning[113, 208, 260, 261, 304, 325], an approach that can tackle both the issues of data scarcity[114] and long training times, by leveraging pre-trained models to address new problems.
The knowledge learned on a problem is stored in the weights of the model. In many practical scenarios with limited availability of data and computational resources, these pre-trained weights represent a good starting configuration to obtain a more refined knowledge of the new task.
In particular, a very common approach is the so-called *fine-tuning*[162, 317] strategy, where the weights of the network are initialized with pre-trained models and only (fine) tuned rather than being trained from scratch — see Figure 3.1.

Instead, in the following pages we consider an approach that adopts convolutional features[262] produced by a state-of-the-art model pre-trained on ImageNet [51], with no further tuning. These

features are used as input to train anew a fast and scalable kernel classifier [188, 243]. Such a model, based on Nyström approach to reduce the problem size and on a preconditioned gradient solver for kernel methods, has a remarkable impact on performances, including training time and scalability. We refer to such approach as *top-tuning* and we show it in Figure 3.2.
This is a simple idea [3, 76, 85, 120], that we re-examine in the light of common practices for transfer learning using deep nets and of fast and scalable kernel methods. Indeed, our analysis consider a set of pre-trained features as input to a fast kernel methodology.
Although we do not focus on a specific application, we have in mind scenarios where computational resources are budgeted and fast training is relevant. This is typical of robotics devices and autonomous systems, especially when operating in unconstrained, quickly changing scenarios, where multiple training may need to be done on the fly [177, 211].
Our study shows that top-tuning is a promising alternative to fine-tuning for small-medium-sized datasets. Indeed, top-tuning provides comparable accuracy w.r.t. fine-tuning (sometimes slightly worse, and sometimes slightly better), with training times between one and two orders of magnitude smaller.

To assess the potential of the proposed methodology, in our experimental analysis we focus on three different aspects:

- Target dataset: to ensure the generality of our empirical observations, we consider 32 target datasets, showing that the top-tuning approach is $\simeq$ 85 times faster on average w.r.t. the fine-tuning one. To provide more robust results we confirm our findings with two additional head classifiers: a naive shallow net and a ridge regressor, showing a smaller still significant speed-up.

- Pre-trained model: to evaluate the influence of a specific model, we include seven different state-of-the-art pre-trained models. We consider complex architectures such as Xception and Vision Transformers as smaller models aimed at embedded devices such as MobileNetV2. Our findings are confirmed showing that despite different pre-trained models, the top-tuning approach is $\simeq$ 70 times faster on average w.r.t. the fine-tuning one.

- Pre-training dataset: to assess dependency on the source dataset, we consider four distinct datasets for pre-training, taking into account different factors such as the amount of images and classes, and image size. We show the influence of the pre-training dataset on the downstream task, identifying the number of classes as a crucial factor in the pre-training dataset.

3.2 Background

In this section, we review some fundamental algorithmic ideas to establish the notation and highlight a few key points. We start by defining the multiclass classification problem. Then, we consider deep learning models with fine-tuning procedures. We conclude with a focus on the no-tuning approach, by considering the fast kernel classifier trained on deep features.

MULTICLASS LEARNING: given a set of input-output pairs $(x_1, y_1), \ldots, (x_n, y_n)$ the goal of supervised classification is to find a classification rule c to predict the outputs corresponding to new inputs. Here, the inputs are high dimensional vectors $x_i \in \mathbb{R}^d$, for instance images, and the outputs are discrete labels $y_i \in \{1, \ldots, T\}$, for instance different semantic classes.
Each label y_i in the training set is mapped into a vectorial one-hot encoding
$\mathbf{y}_i \in \{e_1, e_2, \ldots, e_T\} \subset \mathbb{R}^T$
The search for a classification rule is formulated as the search for a vector function $f : \mathbb{R}^d \to \mathbb{R}^T$, where for all x:

$$f(x) = (f^1(x), f^2(x), \ldots, f^T(x)) \tag{3.1}$$

DEEP LEARNING & FINE-TUNING: we can recall the class of neural networks we described in section 2.4. We have seen that a neural network can be described as a sequence of feature maps Φ organized in subsequent layers as

$$\Phi(x) = \Phi_L \circ \Phi_{L-1} \circ \ldots \circ \Phi_1(x) \in \mathbb{R}^{u_\ell}.$$

We have seen that also convolutional layers are based on composing linear and non-linear operations, with linear operations as convolutions. In this sense, the convolutional neural network can be described as

$$\Phi(x) = \underbrace{\Phi_L \circ \Phi_{L-1} \circ \ldots \Phi_{C+1}}_{\text{Fully connected layers}} \circ \underbrace{\Phi_C \circ \ldots \circ \Phi_1(x)}_{\text{Convolutional layers}}$$

In the following, we view the convolutional layers as providing the data representation, through the feature map $\Phi_C \circ \ldots \circ \Phi_1(x)$, while the fully connected layers as defining a classifier. This distinction is useful in our discussion, although it may be blurred in practice. All layers are trained in an end-to-end fashion minimizing an empirical error via back-propagation, provided a suitable initialization. The training can be made more or less aggressive at each layer by choosing different learning rates. We note that the vast majority of parameters in an architecture such as Equation 2.51 are in the fully connected layers, and hence their training is the most computationally demanding part. Training from scratch often requires large datasets, and using previously trained weights provides an interesting initialization strategy when data are not massive. This is the basic idea of transfer learning by *fine-tuning* [162, 317]. Previously learned weights are fine-tuned for the task at hand. This can amount to updating only the last layer(s), which is necessary since the labels might be different, but also a *deeper* tuning where convolutional layers are also updated. In the following, we give up end-to-end training and replace the fully connected layers with a kernel classifier. In this view, only the very last layer is tuned/updated.

TOP-TUNING VIA FAST KERNEL CLASSIFIERS WITH PRE-TRAINED DEEP FEATURES: the strategy used here can be described with the same notation introduced before. Indeed, we are still considering functions of the form of Equation(2.37), Equation(2.38), but now the feature map Φ is given by

$$\Phi = \underbrace{\Psi}_{\text{Kernel feature map}} \circ \underbrace{\Phi_C \circ \ldots \circ \Phi_1}_{\text{Convolutional layers}}. \tag{3.2}$$

where Ψ is a typically infinite-dimensional feature map corresponding to a kernel on the pre-trained features, like the one described in Equation 2.19. In our experiments, we consider the Gaussian kernel $k(z, z') = e^{-\frac{|z-z'|^2}{2\gamma^2}}$, where γ is the kernel width. Unlike before, here both the convolutional features $\Phi_C, \ldots, \Phi_1$ and the kernel features Ψ are assumed to be fixed and are not trained/tuned. The only free parameters are computed by a ridge regression minimizing

$$\sum_{i=1}^{n} |W\Phi(x_i) - \mathbf{y}_i|^2 + \lambda |W|_F^2 \tag{3.3}$$

where $|\cdot|_F$ if the Frobenius norm. Indeed, the above problem can also be rewritten as:

$$\sum_{i=1}^{n}\sum_{t=1}^{T}(\langle w^t, \Phi(x_i)\rangle - \mathbf{y}_i^t)^2 + \lambda \sum_{t=1}^{T} |w^t|^2 = \sum_{t=1}^{T}(\sum_{i=1}^{n}(\langle w^t, \Phi(x_i)\rangle - \mathbf{y}_i^t)^2 + \lambda |w^t|^2) \tag{3.4}$$

which is simply a one versus all classifier, based on solving each binary classification problem by ridge regression.
From the above expression, it is clear that only the choice of the hyperparameters λ, γ couples the training of the one-vs-all classifiers that are otherwise independent.
Before discussing our experiments we briefly recall some ideas that allow us to massively speed up the training of kernel ridge regression classifiers [188]. Indeed, using the representer theorem [255] we have that

$$f(x) = W\Phi(x) = \sum_{i=1}^{n} k(x, x_i)\mathbf{c}_i, \qquad \mathbf{c}_i \in \mathbb{R}^T, \tag{3.5}$$

where the coefficients satisfy the linear system

$$(\mathbf{K} + \lambda I)\mathbf{C} = \mathbf{Y}, \tag{3.6}$$

where $\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_n), \mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n) \in \mathbb{R}^{nt}$, and $\mathbf{K} \in \mathbb{R}^{nt}$ with $\mathbf{K}_{ij} = k(x_i, x_j)$. However, since the solution to the above system is typically costly, we can consider a Nÿstrom approximation[204] taking a smaller set of $m < n$ randomly selected points $\tilde{x}_1, \ldots, \tilde{x}_m$ in the representer theorem

$$\tilde{f}(x) = \sum_{i=1}^{m} k(x, \tilde{x}_i)\mathbf{c}_i, \qquad \tilde{\mathbf{c}}_i \in \mathbb{R}^T \tag{3.7}$$

so that the coefficients now satisfy the linear system

$$(\mathbf{K}_{nm}^\top \mathbf{K}_{nm} + \lambda \mathbf{K}_m)\tilde{\mathbf{C}} = \mathbf{K}_{nm}^\top \mathbf{Y}, \tag{3.8}$$

where $\tilde{\mathbf{C}} = (\tilde{\mathbf{c}}_1, \dots, \tilde{\mathbf{c}}_n), \mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_n) \in \mathbb{R}^{mt}$, $\mathbf{K}_{nm} \in \mathbb{R}^{nm}$ with $(\mathbf{K}_{nm})_{ij} = k(x_i, \tilde{x}_j)$, $\mathbf{K}_m \in \mathbb{R}^{mm}$ with $(\mathbf{K}_m)_{ij} = k(\tilde{x}_i, \tilde{x}_j)$. Finally, following [188] the linear system in Equation 3.8 is solved by conjugate gradient using an approximate preconditioner given by $(\mathbf{K}_m^2 + \lambda \mathbf{K}_m)$. This latter approach yields a time/memory complexity which is $O(nmT)/O(nm)$ as opposed to the complexity of exact kernel ridge regression (3.6) which is $O(n^3)/O(n^2)$.

CONJUGATE GRADIENT: now we briefly talk about the *conjugate gradient* technique, used in the top-tuning classifier. Aside from that, it is a useful methodology to deal with large linear systems of equations and can be adapted to solve nonlinear problems.
Originally introduced by Hestensen and Stiefel[94] is an iterative method for solving a linear system. Can be used as an alternative to gaussian elimination, as a suitable methodology for large linear systems. Its performance depends on the eigenvalues of the coefficient matrix.
Suppose we want to solve a linear system defined as

$$Ax = b \tag{3.9}$$

where A is a $n \times n$ symmetric positive definite matrix. Solving the linear system is equivalent to solving the following optimization problem:

$$\min \phi(x) = \frac{1}{2} x^T A x - b^T x \tag{3.10}$$

Indeed, this can be immediately seen as:

$$\nabla \phi(x) = Ax - b \stackrel{\text{def}}{=} r(x) \tag{3.11}$$

The main idea about the conjugate gradient method is to produce a set of vectors called *conjugate vectors* having a *conjugacy property*. Indeed, a set $\{p_0, p_1, \dots, p_l\}$ is said to be conjugate w.r.t. a matrix A whether $\forall i \neq j$

$$p_i^T A \, p_j = 0 \tag{3.12}$$

We can notice that vectors in a conjugate set are also linearly independent.
The intuition about the conjugate gradient method is minimizing the $\phi(\cdot)$ problem in n steps by first determining the conjugate directions, i.e. the "important" directions, and then by successively minimizing along every conjugate direction.
Given a starting point $x_0 \in \mathbb{R}^n$ and a conjugate set $\{p_0, p_1, \dots, p_l\}$ we can generate a sequence of steps $\{x_1, \dots, x_k\}$ where

$$x_{k+1} = x_k + \alpha_k p_k \tag{3.13}$$

having step length $\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k}$. It is possible to prove that under such conditions the sequence $\{x_k\}$ converges to the solution x^* of the linear system defined in Equation 3.9 in at most n steps. We can provide a simple geometric interpretation of the above problem. When the A matrix is diagonal, the contours of the function $\phi(\cdot)$ are ellipses with axes aligned with the carte-

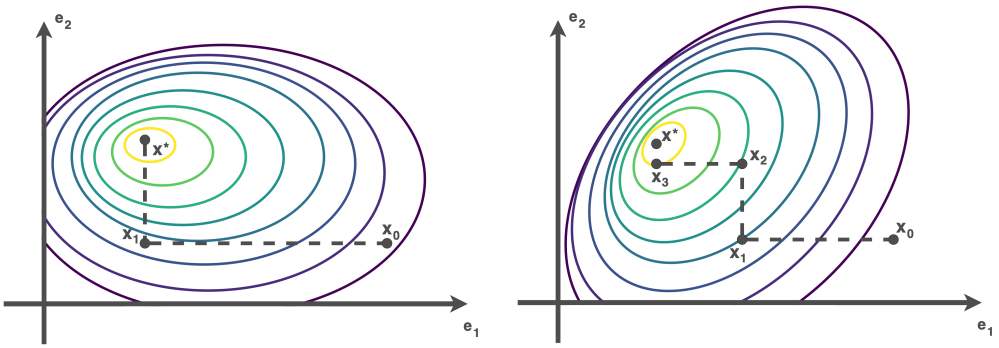


Figure 3.3: Geometric interpretation of conjugate gradient technique. (Left) When the A matrix is diagonal, the contours of the function $\phi(\cdot)$ are ellipses with axes aligned with the cartesian coordinate direction e_1, e_2. The optimal point is reached within n steps. (Right) When the A matrix is not diagonal w.r.t. e_1, e_2. More than n steps are needed.

sian coordinate direction e_1, e_2. We can show such a situation in Figure 3.3(left). We can find the minimum by performing a one-dimensional optimization along each coordinate.
Instead, this is not true when A is not diagonal. In that situation, the algorithm will not move along the e_1, e_2 directions.

However, we can recover the good situation shown in Figure 3.3(Left) by transforming the problem and making A diagonal along e_1, e_2 directions. We transform the problem by defining a new input $\hat{x}$

$$\hat{x} = S^{-1}x \tag{3.14}$$

where S is a $n \times n$ matrix defined as

$$S = [p_0, p_1, \ldots, p_{n-1}] \tag{3.15}$$

the set of conjugate directions w.r.t. A. The quadratic problem we defined in Equation 3.10 now becomes:

$$\hat{\phi}(\hat{x}) \overset{\text{def}}{=} \phi(S\hat{x}) = \frac{1}{2}\hat{x}^T(S^TAS)\hat{x} - (S^Tb)^T\hat{x} \tag{3.16}$$

where $S^TAS)$ is diagonal by conjugancy property. Now we can minimize along the $\hat{x}$ coordinates. Performing the input transformation is important as at every step we determine one component of the solution x^*. On this we have a theorem, stating that for any $x_0 \in \mathbb{R}^n$ we consider the sequence $\{x_k\}$, then $r_k^T p_i = 0$ for every step $i \in [0, 1, \ldots, k-1]$. Moreover, x_k is a minimizer of Equation 3.10 over the set

$$\{x | x = x_0 + span\{p_0, p_1, \ldots, p_{k-1}\}\} \tag{3.17}$$

Clearly, the whole procedure relies on the direction we select to transform our input. Different solutions have been tested in the past such as computing the eigenvectors which is a good but expensive methodology. Also modifying the Gram-Schmidt orthogonalization is possible but incurring in the problem of storing the entire direction set. Instead, the conjugate gradient method produces the subsequent direction based only on the previous one. That means that to compute the new conjugate vector p_k it only needs p_{k-1} without needing $p_0, \dots, p_{k-2}$. Indeed, each direction p_k is a linear combination of the negative residual $-r_k$

$$p_k = -r_k + \beta_k p_{k-1} \tag{3.18}$$

where $\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}}$.
Overall, the conjugate gradient algorithm is shown in Algorithm 1. With all these considerations in mind is possible to prove that

- $p_0, p_1, \dots, p_{n-1}$ are conjugate gradient directions, implying the convergence in n steps
- The residuals $\{r_k\}$ are mutually orthogonal
- Each direction p_k and the residual r_k are contained in the Krylov subspace[164, 263] of degree k for r_0

$$\mathcal{K}(r_0, k) \stackrel{\text{def}}{=} span\{r_0, Ar_0, \dots, A^k r_0\} \tag{3.19}$$

Algorithm 1 Conjugate gradient algorithm

Require: x_0
  $r_0 \leftarrow Ax_0$
  $p_0 \leftarrow -r_0$
  $k \leftarrow 0$
  while $r_k \neq 0$ **do**
    $\alpha_k \leftarrow -\frac{r_k^T r_k}{p_k^T A p_k}$
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
    $r_{k+1} \leftarrow Ax_{k+1} - b$
    $\beta_{k+1} \leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k}$
    $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$
    $k \leftarrow k + 1$
  end while

To accelerate the conjugate gradient method, we usually couple it with a preconditioning technique[22]. Similarly to the transformation we saw previously, we transform the linear system $\hat{x} = Cx$ obtaining

$$\hat{\phi}(\hat{x}) = \frac{1}{2}\hat{x}^T (C^{-T} A C^{-1})\hat{x} - (C^{-T} b)^T \hat{x} \tag{3.20}$$

and using the conjugate gradient algorithm to solve the linear system

$$(C^{-T}AC^{-1})\hat{x} = C^{-T}b \tag{3.21}$$

Selecting a good preconditioner is fundamental to obtaining good performances on the problem we want to solve. There is no general best solution as knowledge about the structure and origin of the problem is the key. However, different practical preconditioners have been proposed in the past such as symmetric successive overrelaxation(SSOR)[86], incomplete Cholesky[127] and banded preconditioners[182].

So far we focused on linear scenarios. The conjugate gradient can be used with a general convex function f thanks to the extension provided by Fletcher and Reeves[70] and its Polak-Ribiere variant[225]. Such approaches are based on two simple changes

- In place of $\alpha_k \leftarrow -\frac{r_k^T r_k}{p_k^T A p_k}$ we perform a line search identifying an approximated minimum of the ϕ non-linear function along p_k
- The residual r,is replaced by gradient of non-linear objective function f

3.3 Related works

The investigation of transfer learning settings and how different parameters influence performances in terms of accuracy and training time is not itself a novelty. Recent works focus on the properties of specific neural network architectures on transfer-learning accuracy. In [137], the authors assess whether models that achieve the best performances when trained on ImageNet show the same trend also on other vision tasks. They find that a model accuracy on ImageNet correlates with the performances of the same model fine-tuned on target tasks. Another recent work [201] suggests that additional factors are involved in determining the transferability of models on top of ImageNet pre-training accuracy. The authors show that a high degree of diversity in the features learned brings better transferable models. The usage of ImageNet as a source dataset in transfer-learning frameworks has become a standard for different computer vision tasks [38, 89, 109, 111, 185, 216]. Recent works focus exactly on investigating what are the features of ImageNet responsible for the high descriptive quality of a pre-trained ImageNet model learned features. In [111] the authors perform an empirical investigation on ImageNet properties, training a CNN on different subsets of ImageNet, varying the number of samples, classes, and granularity to assess their effect in terms of accuracy.

At the same time, several works focus on the problem of speeding-up training process, evaluating potential influencing factors, and proposing faster training strategies [90, 118, 136, 181].
In [90] the authors focus on the impact of a CNN complexity on the training time, evaluating the accuracy with constrained time cost, considering the influence of different factors including depth, number of filters, and filter size.
In [136], the authors investigate the paradigm of pre-training deep models on large supervised datasets in a transfer-learning framework for different vision tasks. They introduce a cheap fine-tuning protocol that avoids expensive hyperparameters search, which is replaced by a custom

heuristic procedure. In [181] and [118], different strategies to modify convolutional filters are proposed to reduce the training and inference time for CNNs. Another relevant area of research includes the compression of deep neural networks to reduce their storage and computational cost (e.g., parameters pruning and quantization) [39, 159].
The cited works focus on investigating transfer-learning parameters that influence performances in terms of accuracy or strategies to reduce training time, with no specific comparison between different approaches nor evaluating the trade-off between accuracy and training resources.
In the following pages, instead, we perform an extensive experimental analysis comparing different top-tuning and fine-tuning approaches on a large ensemble from small to medium size datasets. We show that convolution features pre-trained on rich datasets as ImageNet, provide general-purpose features which can be transferred to a new task simply by training an external classifier, with a significant training process speed-up.
The speed-up we achieve is also boosted by the choice of a fast kernel method as a head classifier. In the last few years, the fast kernel approach we are adopting was tested in different scenarios. In [188, 197] theoretical guarantees for the Nyström approximation technique is provided and empirically examined. Its effectiveness in terms of speed is shown on classical numerical feature datasets, not dealing with computer vision datasets.
In [34, 177] the fast kernel is tested on specific robotic datasets for detection and pose-estimation tasks.
In [12, 19, 187] Nyström approximation is tested on basic image datasets such as MNIST and CIFAR10 while in [305] it is used for 3D surface reconstruction. In our work we instead present an extensive analysis of more than thirty general-purpose image datasets, proving the effectiveness of the fast kernel approach on a wide set of image classification tasks, and showing its potential in the computer vision domain.

3.4 Methodology

Given the above discussion, we first describe the two basic approaches we consider for transfer learning. Then, we present technical details about the conducted experiments. We consider the hyper-parameters tuning, both for the fine-tuning and top-tuning procedures. Lastly, we report details about the datasets used in the empirical analysis.

Pipelines: we assume we are given a state-of-the-art deep learning model pre-trained on a source dataset (e.g. ImageNet), and a target dataset of input-output pairs $\{(x_i, y_i)\}_{i=1}^{n}$ to address a new image classification problem. In our study we compare two alternative transfer approaches:

1. *Fine-tuning*: we add fully connected layers on top of the pre-trained architecture. Then we fine-tune the model for the new task. Notice that this procedure updates both the parameters of the fully connected and of the pre-trained part.

2. *No-tuning*: we use convolutional features produced by a state-of-the-art model pre-trained on ImageNet, with no further tuning. These features are then used as input to a fast kernel classifier. It is worth noticing that this procedure uses the pre-trained part of the network

only once, without updating its weights. Only the parameters of the classifier are updated.

EXPERIMENTS DETAILS: a model can be tuned in several ways involving numerous hyper-parameters. In our analysis, we do not focus on an exhaustive hyper-parameters exploration. Instead, we consider a set of plausible configurations, both for the fine-tuning and the top-tuning pipelines, according to guidelines in previous works. It is worth noticing that, for each pipeline, the overall training time is computed as the summation of every considered configuration training time.

- Fine-tuning: we perform a five-fold cross-validation analysis involving the following hyper-parameters:
  - Training steps: inspired by previous studies on a similar context[137] we limited this quantity to 20.000 training steps, coupled with an early stopping criterion.
  - Early stopping: we monitor the validation loss with a patience parameter equal to 10.
  - Weights update: we only consider fine-tuning the whole convolutional part, making the model more adaptable to the downstream task w.r.t. the top-tuning approach. That is to avoid a combinatorial explosion in the number of configurations, that would result in excessive and unfair training time for the fine-tuning pipeline.
  - Batch size: taking into account previous studies about batch size[21, 88, 137, 184] we compute it as: $b = \lfloor 2^{2 \cdot log_{10}(n)-1} \rfloor$ where n is the number of points in the dataset.
  - Optimizer: we use default Stochastic Gradient Descent (SGD). As suggested by[81], we use two different learning rates: $l = \{0.1, 0.01\}$

  All the hyper-parameters for our neural network model are fixed but the learning rate. Hence we run one training instance per each considered learning rate value, taking the best-performing one for the results.
  The training time reported in the results is the sum of the computational time requested for both training instances.

- Top-tuning: also for the top-tuning approach we perform a five-fold cross-validation analysis involving the following hyper-parameters:
  - Kernel: the approximated kernel ridge regression is based on a Gaussian kernel $k(z, z') = e^{-\frac{|z-z'|^2}{2\gamma^2}}$, with γ kernel width. We use two values of $\gamma = \{10^2, 10^3\}$.
  - Regularization: we consider two values for the regularization term: $\lambda = \{10^{-5}, 10^{-6}\}$

  With two possible values for both kernel width and regularization, we run four different training instances, taking the best-performing one for the results. The training time reported in the results is the sum of the computational time requested for the four training instances.

DATASETS DETAILS: We include 32 datasets in our experiments, from a wide range of contexts and scenarios (see Table 3.1). Our collection includes popular datasets in the computer vision

Dataset name	#images (Tr/Te)	Img. size mean	#classes
AFHQ (AF)[40]	13.167/1.463	512 × 512	3
Beans (BE)[147]	1.167/128	500 × 500	3
Best artworks (BA)[281]	7.896/878	980 × 921	50
Boat types (BT)[46]	1.315/147	905 × 1234	9
Caltech-101 (C101)[66]	3.060/6.084	251 × 282	102
Cassava (CSV)[198]	7.545/1.885	573 × 611	5
Cats vs Dogs (CVSD) [65]	20.935/2.327	365 × 410	2
Chest xray (CXRAY) [126]	4.708/524	968 × 1321	2
CIFAR10 (CIF10) [142]	50.000/10.000	32 × 32	10
CIFAR100 (CIF100) [142]	50.000/10.000	32 × 32	100
Citrus leaves (CLV) [236]	534/60	256 × 256	4
Colorectal hist (COL) [124]	4.500/500	150 × 150	8
Deep weeds (DW) [205]	15.758/1.751	256 × 256	9
DTD (DTD)[43]	3.760/1.880	453 × 500	47
EuroSAT (ES) [92]	24.300/2.700	64 × 64	10
FGVC Aircraft (AIR)[178]	6.667/3.333	353 × 1056	100
Footb vs Rugby (FVSR) [77]	2.203/245	618 × 788	2
Gemstones (GEM) [37]	2.571/286	330 × 335	87
Hors or Hum (HVSH) [196]	1.027/256	300 × 300	2
iCubWorld subset (ICUB)[211]	86.400/9.600	256 × 256	10
Indian Food (IF) [227]	3.600/400	550 × 610	80
Make No Make(MVSN)[275]	1.355/151	211 × 246	2
Malaria (MAL) [234]	24.802/2.756	133 × 132	2
Meat quality (MQA) [290]	1.706/190	720 × 1280	2
Oxford Flowers (OF) [203]	2.040/6.149	538 × 624	102
Oxford-IIIT Pets (OP) [209]	3.680/3.669	383 × 431	37
Plankton (PL) [213]	4.500/500	106 × 120	10
Sars Covid (SCOV) [267]	2.232/249	260 × 350	2
Stanford Cars (SC) [141]	8.144/8.041	308 × 573	196
Stanford Dogs (SD) [131]	12.000/8.580	386 × 443	120
Tensorflow Flowers(TFF) [282]	3.303/367	272 × 365	5
Weather (MW) [271]	1.012/113	335 × 506	4

Table 3.1: The datasets adopted in our analysis. For every dataset, we provide the amount of images (train and test split, respectively), the mean image size of the dataset and the number of classes.

community, like CIFAR10/100 and Caltech101, as well as more challenging datasets where the amount of data is limited with respect to the number of classes and complexity of the task (e.g., Stanford Cars and FGVC aircraft).
Finally, we include datasets with a significantly limited amount of images, to consider practical cases where transfer learning procedures may be fundamental (e.g., Beans and Citrus leaves).
From Table 3.1 we can notice that both number of images and classes can vary deeply from one task to another. This can influence the task's hardness. On average each dataset has 11746.46 images and 35.21 classes. The average number of images per classes is 1780.24. The dataset with most images per class is the Malaria dataset with 12401 images per class. The dataset with few images per class is Oxford Flowers with 20 images per class.

3.5 Experiments

We now describe the details of the empirical analysis. We compare fine-tuning and top-tuning in terms of accuracy and training time. To provide robust results we also replace the fast-kernel head classifier with a vanilla fully connected network and a ridge regressor. Then, we consider seven different pre-trained neural networks to replace the pre-trained architecture used by both fine-tuning and top-tuning, to weigh the dependency of the results w.r.t. the pre-trained architecture. Lastly, we evaluate the importance of pre-training by considering four different datasets as pre-train for our model. All the experiments have been carried out on a single Quadro RTX 6000 GPU, 24Gb VRAM.

3.5.1 Top-tuning is highly faster with limited accuracy drop

To compare fine-tuning and top-tuning approaches, we first fix the neural network model and the pre-training dataset, considering an ImageNet pre-trained DenseNet201[108] architecture. This model represents a good compromise between predictive power and size in terms of number of parameters.
We consider the 6 different configurations defined in section 3.4 averaging the results of a 5-folded procedure over the 32 datasets, resulting in $6 \cdot 5 \cdot 32 = 960$ distinct training processes.

In Figure 3.4 we show the overall accuracy results. Each point represents a different dataset. Its position is given by the accuracy obtained by the best fine-tuning configuration on the x-axis, and by the best top-tuning configuration on the y-axis. The diagonal is marked for readability purposes. Intuitively, when a point is lying below the diagonal, the fine-tuning model is performing better w.r.t the top-tuning one, and vice versa.
Most of the datasets lie around the diagonal, showing similar accuracy between fine-tuning and top-tuning methods. Indeed,

$$\Delta Acc = Acc_{\text{top-tuning}} - Acc_{\text{fine-tuning}} \in [-2.5\%, +2.5\%]$$

in 60% of our experiments.
For these datasets, the benefits coming from the fine-tuning procedure are either absent or typically marginal. Indeed, on half of the considered datasets, the top-tuning procedure is performing better than the fine-tuning one.

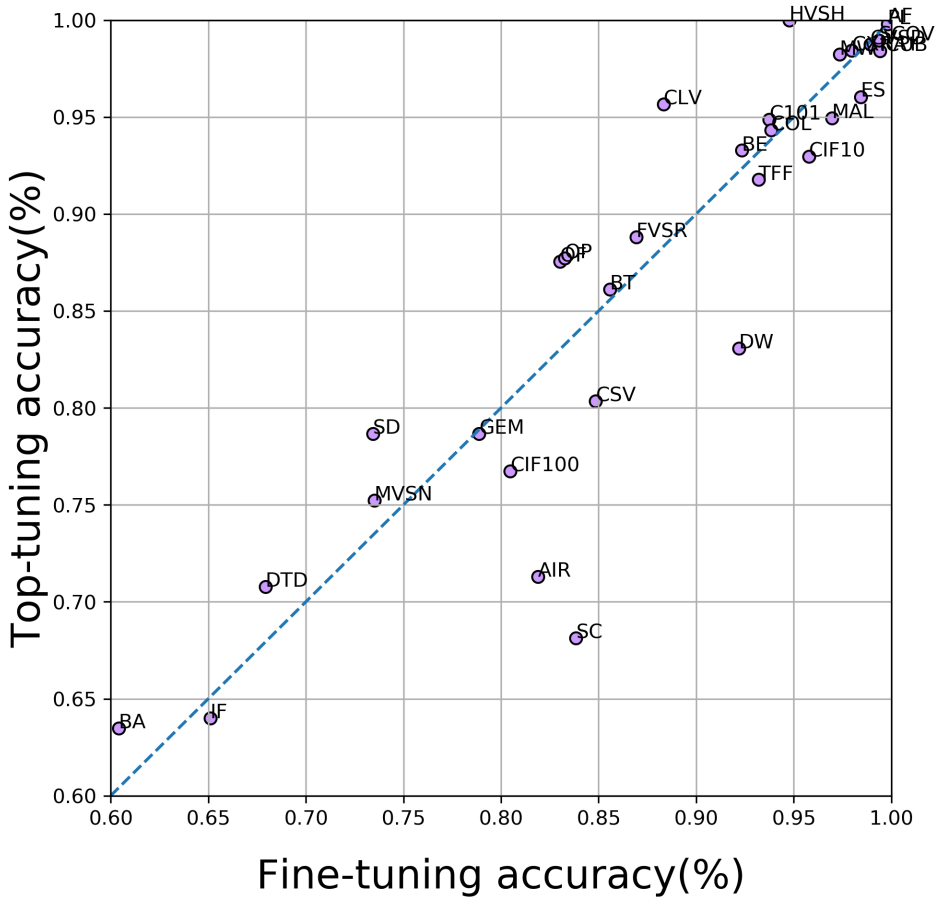


Figure 3.4: Overall accuracies on different target datasets. Each point represents a dataset. The accuracy was obtained by the best fine-tuning model on the x-axis, and by the best top-tuning model on the y-axis. When a point is lying below the diagonal, the fine-tuning model is performing better w.r.t the top-tuning one, and vice versa.

Only on few datasets, e.g. FGVC aircraft and Stanford Cars(SC), fine-tuning provides a benefit. This behaviour could be related to two factors. (i) Representation in ImageNet: as pointed out by [137], in ImageNet cars and planes are represented at a coarse-grained level. For instance, ImageNet contains only two plane classes; (ii) Dataset hardness in terms of the number of classes, granularity, and the amount of images per class.

In Figure 3.5 we show the overall results in terms of training time. Each column refers to a different dataset, reporting the speed-up obtained by the top-tuning w.r.t. fine-tuning. For instance, on the AFHQ dataset, the top-tuning training was $\sim$ 95 times faster. The top-tuning model is always highly faster to train than the fine-tuning one. The speed-up is in the range [10,150] with mean 84.64 ± 38.97 across the datasets.
On larger datasets, e.g. CIFAR100, the training time was reduced from $\sim$ 2 hours to $\sim$ 10

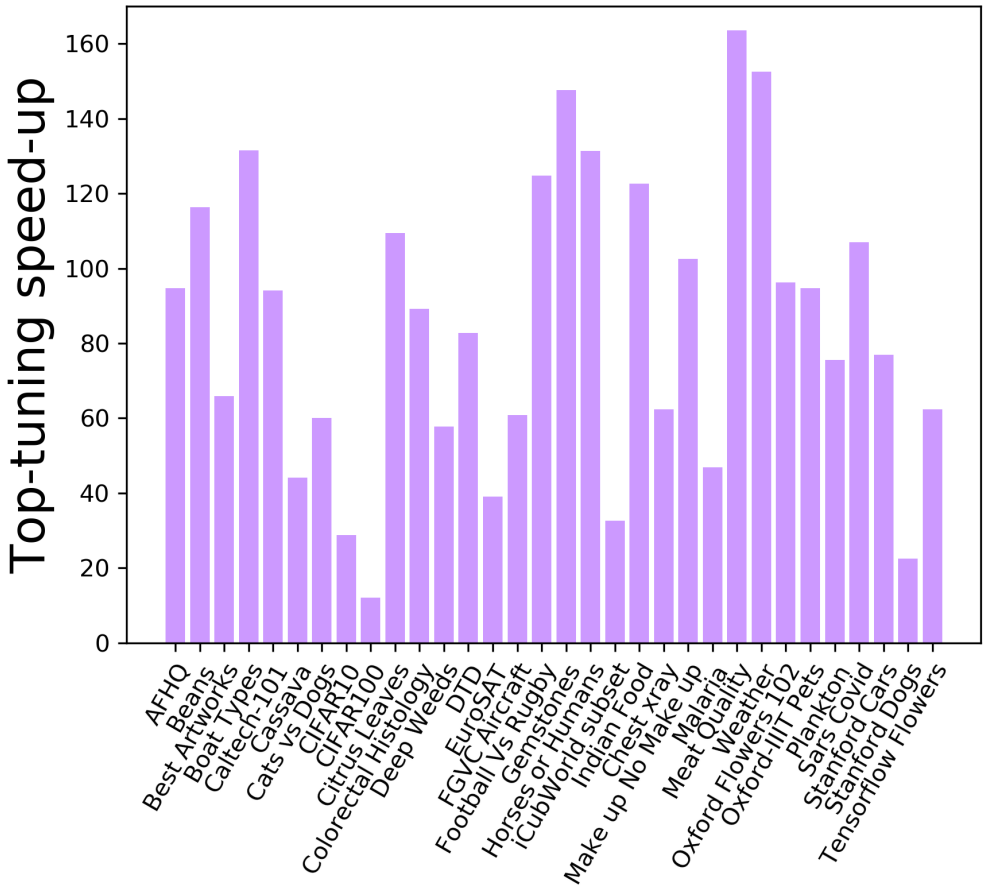


Figure 3.5: Overall speed-up on different target datasets obtained by the top-tuning model w.r.t. the fine-tuning one. Each column represents a dataset. The corresponding height represents the speed-up factor of the top-tuning model w.r.t. the fine-tuning one(e.g. on the AFHQ dataset the top-tuning training was $\sim$ 95 times faster). All the experiments have been carried out on a single Quadro RTX 6000 GPU, 24Gb VRAM.

minutes; computed as the training time sum of the two fine-tuning and four top-tuning configurations, respectively.
We can relate the faster training time to (i) number of parameters: the top-tuning model have a number of parameters on average two orders of magnitude lower than the fine-tuning ones; (ii) impact of backpropagation: every layer needs to wait for the subsequent layer computation.

In Figure 3.6 and Figure 3.7 we show an in-depth analysis reporting accuracy and training time for every dataset.

Table 3.2 summarizes the quantitative results obtained with our experiments both in terms of accuracy and training time speed-up. For every dataset, we report the $\Delta Acc = Acc_{\text{top-tuning}} - Acc_{\text{fine-tuning}}$ and the corresponding speed-up.

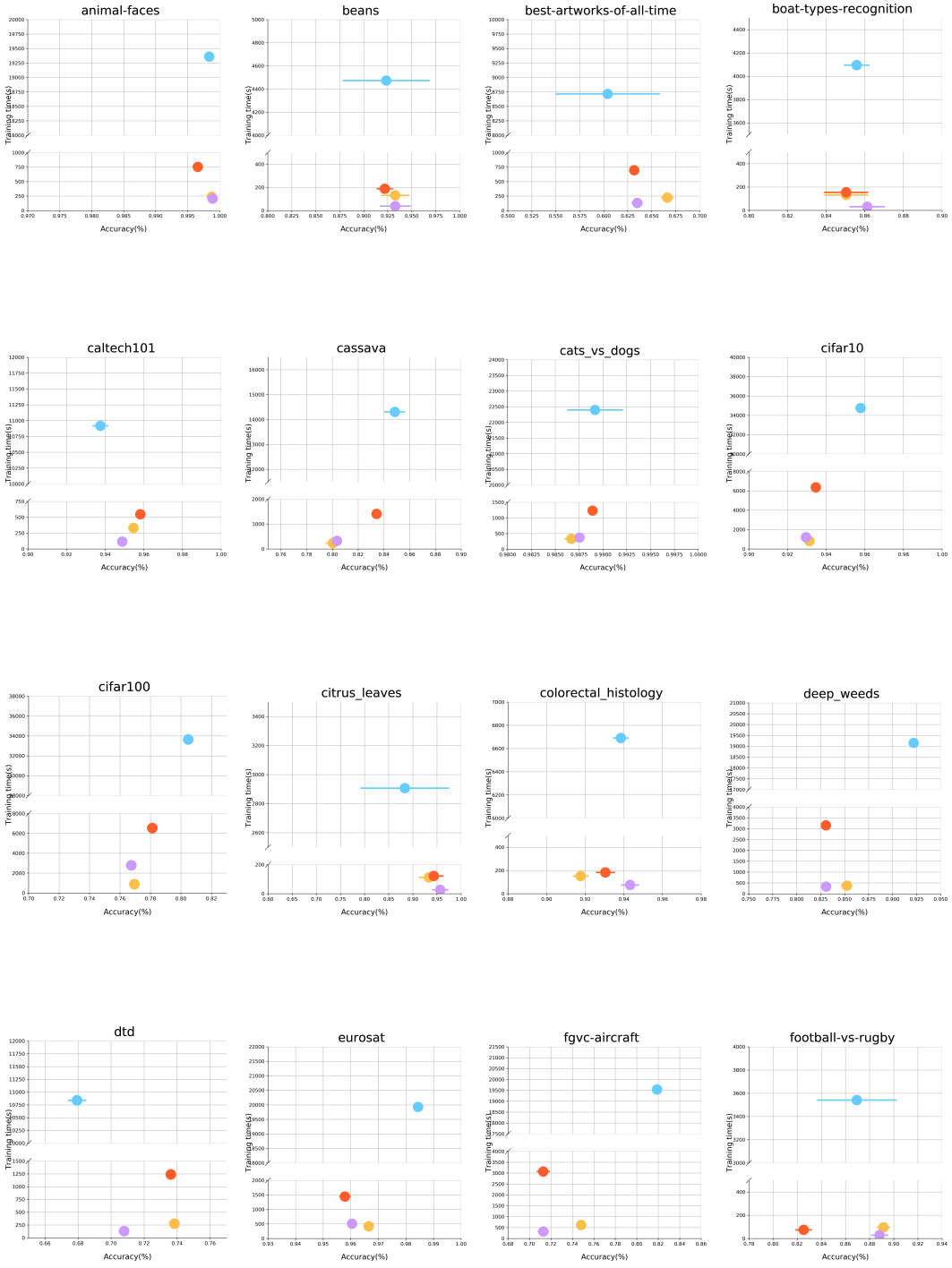


Figure 3.6: Overall accuracy and training time for the datasets considered in the empirical analysis, datasets from 1 to 16

Figure 3.7: Overall accuracy and training time for the datasets considered in the empirical analysis, datasets from 17 to 32

Dataset	ΔAcc	SpeedUp	Dataset	ΔAcc	SpeedUp
AFHQ	+0.10%	94.70×	Football vs Rugby	+1.90%	124.8×
Beans	+0.90%	116.3×	Gemstones	−0.20%	147.6×
Best artworks	+3.10%	65.90×	Horses or Humans	+5.20%	131.3×
Boat types	+0.50%	131.5×	iCub World subset	−1.00%	32.60×
Caltech-101	+1.10%	94.00×	Indian Food	−1.10%	122.6×
Cassava	−4.50%	44.00×	Make up	+1.70%	102.5×
Cats vs Dogs	−0.20%	60.00×	Malaria	−2.00%	46.80×
Chest xray	+0.50%	62.40×	Meat Quality	+0.00%	163.6×
CIFAR10	−2.80%	28.90×	Oxford Flowers102	+4.50%	96.30×
CIFAR100	−3.70%	12.10×	Oxford-IIIT Pets	+4.50%	94.60×
Citrus leaves	+7.30%	109.5×	Plankton	+0.00%	75.50×
Colorect. histology	+0.50%	89.20×	Sars Covid	−0.40%	107.0×
Deep weeds	−9.10%	57.70×	Stanford Cars	−15.7%	76.90×
DTD	+2.90%	82.80×	Stanford Dogs	+5.20%	22.50×
EuroSAT	−2.40%	39.10×	Tensorflow flowers	−1.40%	62.40×
FGVC Aircraft	−10.6%	60.90×	Weather	+0.90%	152.4×

Table 3.2: Quantitative results about the analysis on different datasets. The second column reports the $\Delta Acc = Acc_{\text{top-tuning}} - Acc_{\text{fine-tuning}}$. The third one refers to the corresponding speed-up obtained when using the top-tuning procedure(e.g. on the AFHQ dataset the top-tuning training was 94.70 times faster).

Lastly, although in this work we do not focus on inference time, we report for completeness that the two pipelines needed similar time for a prediction.

3.5.2 Analysis with different head classifiers

To test the generality of our approach we replace the fast kernel classifier with two head classifiers: a shallow net and a ridge regression classifier.
The training procedure is similar to the one presented in section 3.4. For the shallow net, we consider just two configurations with default Stochastic Gradient Descent (SGD) and two different learning rates: $l = \{0.1, 0.01\}$.
The training time reported in the results is the sum of the computational time required for the two training instances. Notice that this architecture is identical to the fine-tuning one. The main difference lies in which part of the model is tuned. The shallow nets update only the parameters of the last three fully connected layers while the fine-tuning one update all the weights of the model.
For the Ridge regressor, we use three different configurations corresponding to three different values of the regularization term $\alpha = \{10^1, 10^{-1}, 10^{-3}\}$. Here again, the training time reported in the results is the sum of the computational time required for the three training instances.
In Figure 3.8 we compare, in terms of accuracy, the fine-tuning model with both the shallow net(left) and the ridge regressor(right) as external classifiers. The obtained results are similar to

the ones shown in Figure 2. Most of the datasets lie around the diagonal for both shallow net and ridge as an external classifier, showing similar accuracy between fine-tuning and top-tuning methods.

In Figure 3.9 we compare, in terms of training time, the fine-tuning model with both the shallow net(left) and the ridge regressor(right) as an external classifier. With the fast kernel classifier as an external classifier, the average speed-up is 84.64 ± 38.97. Instead, by using a shallow net as an external classifier we obtain an average speedup of 40.74 ± 12.43.
The speed-up is instead 16.37 ± 9.57 if we use the ridge regressor as external classifier.
These results show that regardless of the choice of external classifier, top-tuning confirms a training time speed-up of at least one order of magnitude w.r.t. the fine-tuning pipeline. Among them, the fast-kernel model reports a significant boost w.r.t. to its competitors.

3.5.3 Impact of pre-trained model

To weigh the dependency of the results on the pre-trained architecture we extend the results obtained with DenseNet201 on five state-of-the art pre-trained models: (i) EfficientNetB0 [280]; (ii) InceptionResNetV2 [277]; (iii) MobileNetV2[105]; (iv) ResNet152[91]; (v) Xception[41].
We test these models on four different datasets where: fine-tuning has only marginal benefits (Caltech-101, CIFAR100), the top-tuning approach provides better results w.r.t. fine-tuning (DTD), and fine-tuning outperforms top-tuning approach (Stanford Cars).
The training procedure is analogous to the one performed for DenseNet201 with 5 additional pre-trained models and a subset of 4 datasets, performing 600 additional training processes.
Figure 3.10 shows the overall results. Each color refers to a different target dataset, each symbol corresponds to a different pre-trained neural network.

Different pre-trained neural networks show a similar trend to the one obtained by DenseNet201. Figure 3.10(Left) shows that on datasets where the drop between fine-tuning and top-tuning was marginal (e.g. Caltech, DTD) using different pre-trained model results in an analogous behavior. Similarly, on a dataset where such the drop was greater (e.g. Stanford Cars) using different pre-trained results in a similar performance deterioration. Figure 3.10(Right) confirms our findings on the matter of training time speedup, showing that the top-tuning approach is highly faster w.r.t. the fine-tuning one.
Table 3.3 summarizes the results obtained both in terms of accuracy and training time speed-up. It shows quantitatively that different pre-trained networks behave similarly on the same dataset w.r.t. DenseNet201. Moreover, it shows that the top-tuning approach is approximately 70 times faster w.r.t. the fine-tuning one. Such results suggest that our findings are low-dependent from the pre-trained neural network adopted.

Preliminary results with transformers In the last few years a new generation of models, called *transformers*, has been introduced. Such models are usually composed by hundreds of million of parameters, obtaining state-of-the-art performances. Commonly, they are pre-trained on an extended version of ImageNet called ImageNet21k[239]. In the following experiments we replace the first convolutional part of the pipeline with a Vision Transformer(ViT-L/16) as presented in [60]. With 32 target datasets for the top-tuning approach and 2 target datasets for the

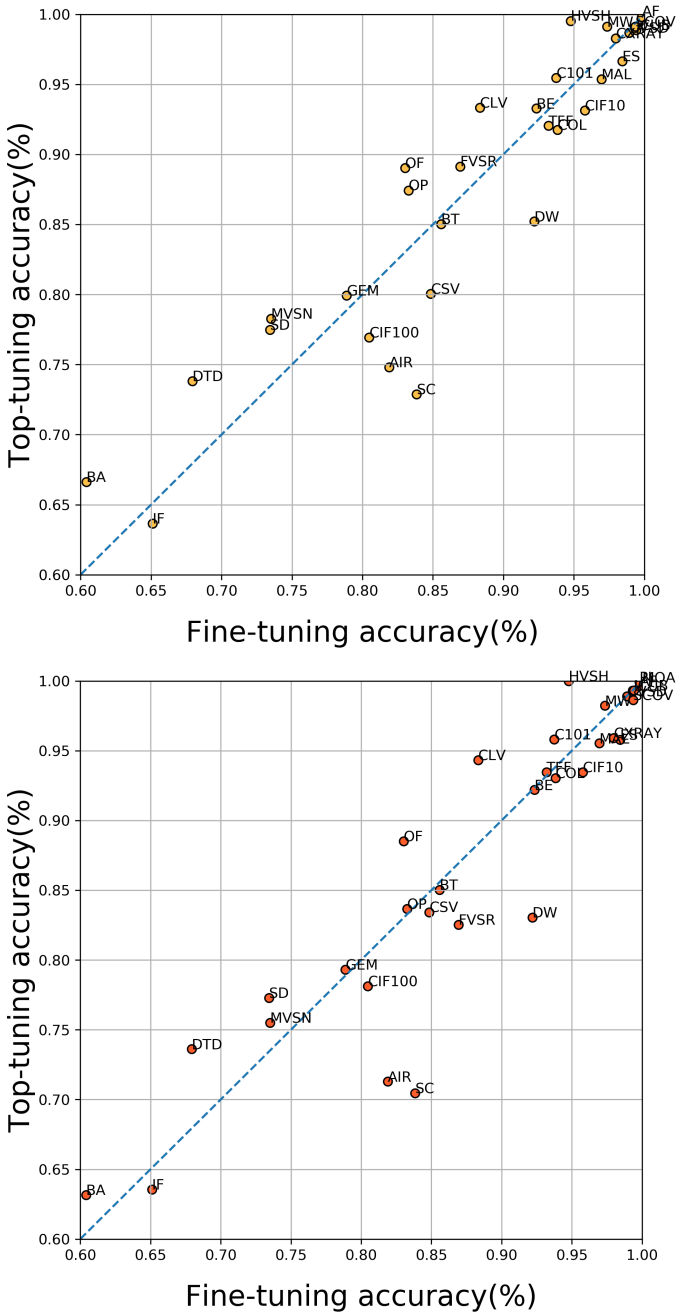


Figure 3.8: Overall accuracy results on different target datasets with shallow net and ridge as external classifiers w.r.t the fine-tuning model. (Top) Accuracies obtained over different target datasets by using a shallow net as external classifier. (Bottom) Accuracies obtained over different target datasets by using ridge regressor as external classifier.

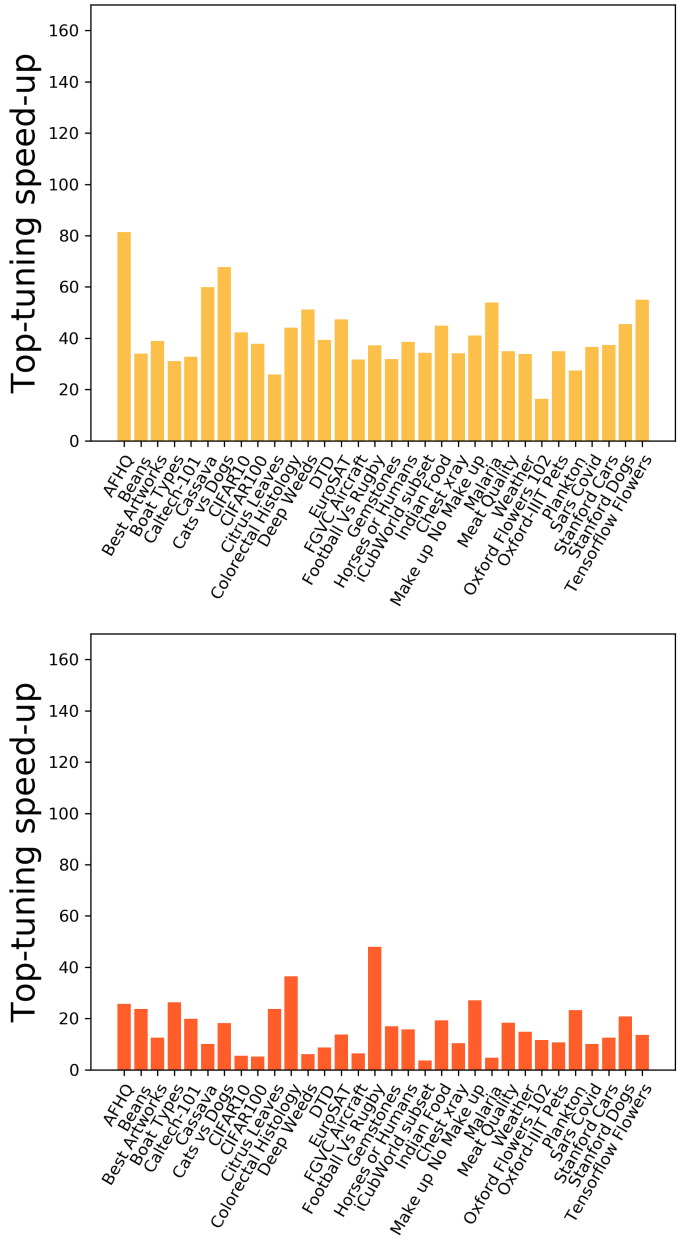


Figure 3.9: Overall speed-up results on different target datasets with shallow net and ridge as external classifiers w.r.t the fine-tuning model. (Left) Speed-up was obtained over different target datasets by using a shallow net as an external classifier. (Right) Speed-up was obtained over different target datasets by using a ridge regressor as an external classifier.

fine-tuning one, we perform 650 additional training processes.
For the top-tuning pipeline, we were able to replicate our analysis on all 32 target datasets. We

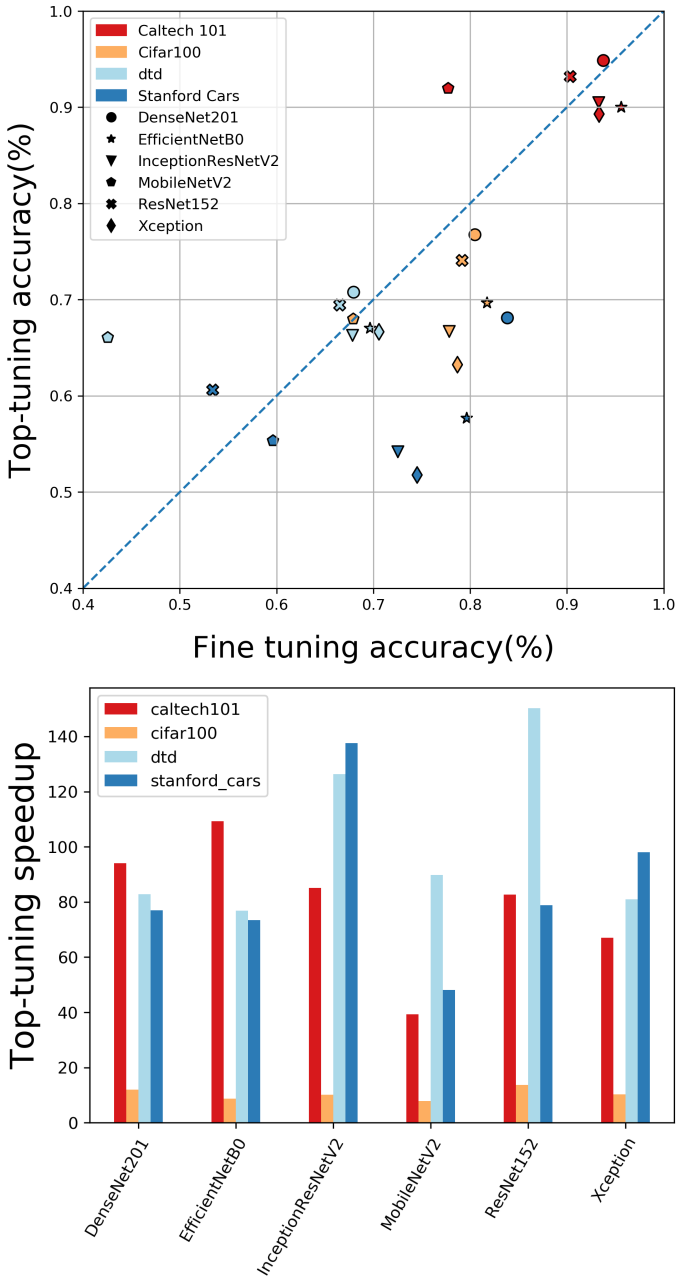


Figure 3.10: Overall results on different pre-trained models. (Left) Accuracies were obtained over different target datasets with different pre-trained models. (Right) Speed-up obtained by the top-tuning model w.r.t the fine-tuning one for each dataset and for each pre-trained model.

report, due to longer inference time, information about the features extraction training time in Table 3.4

	Caltech-101		CIFAR100		DTD		Stanford Cars	
	ΔAcc	SpUp	ΔAcc	SpUp	ΔAcc	SpUp	ΔAcc	SpUp
DenseNet201	+1.10%	94.00	−3.70%	12.1	+2.90%	82.80	−15.7%	76.90
EfficentNetB0	−5.60%	109.3	−12.1%	8.70	−2.60%	76.80	−21.9%	73.40
Inc.ResNetV2	−2.80%	85.10	−11.1%	10.2	−1.50%	126.3	−18.3%	137.6
MobileNetV2	+14.3%	39.40	+0.10%	7.90	+23.5%	89.80	−4.3%	48.10
ResNet152	+2.90%	82.60	−5.10%	13.7	+2.90%	150.2	+7.2%	78.90
Xception	−4.00%	67.10	−15.4%	10.3	−3.90%	81.00	−22.7%	98.10

Table 3.3: Analysis on different pre-trained models: variation in accuracy (ΔAcc) and speedup (SpUp), between fine-tuning and top-tuning.

Dataset	Duration	image/secs	Dataset	Duration	image/secs
Beans[147]	60.56	19.26	Gemstones[37]	207.24	12.40
Best artworks[281]	300.74	26.26	Hors or Hum[196]	134.51	7.64
Boat types[46]	147.80	8.90	iCubWorld subset[211]	1317.19	68.33
Caltech-101[66]	192.79	31.56	Indian Food[227]	213.97	16.83
Cassava[198]	250.73	30.09	Make No Make[275]	203.83	6.65
Cats vs Dogs[65]	523.76	39.97	Malaria[234]	393.19	63.08
CIFAR10[142]	889.63	56.20	Meat quality[290]	199.56	8.55
CIFAR100[142]	734.28	68.09	Oxford Flowers[203]	163.09	12.51
Citrus leaves[236]	51.74	10.32	Oxford-IIIT Pets[209]	152.27	24.17
Colorectal hist [124]	219.60	20.49	Plankton[213]	187.14	24.05
Deep weeds[205]	264.70	59.53	Sars Covid[267]	206.64	10.80
DTD[44]	212.03	17.73	Stanford Cars[141]	148.57	54.81
EuroSAT[92]	382.29	63.56	Stanford Dogs[131]	211.30	56.79
FGVC Aircraft[178]	137.49	48.49	Tensorflow Flowers[282]	222.31	14.86
Footb vs Rugby[77]	151.11	14.58	Weather[271]	147.12	6.88

Table 3.4: Per dataset Vision Transformer inference time

We report the results in Table 3.5, where $\Delta Acc = Acc_{\text{Transform}} - Acc_{\text{Conv}}$ is the accuracy gain of transformer w.r.t. the usual DenseNet201. On average $\Delta Acc = 4.58\% \pm 5.36\%$ showing good improvements by using the pre-trained transformer as a features extractor.

For the fine-tuning approach, due to long training times, we could not replicate the analysis on all the datasets. We carried out the whole analysis on two small datasets (Citrus Leaves and Oxford Flowers) where we reached an absolute accuracy of 97.5% and 99.1%, respectively.
If we compare these values with the ones obtained by the top-tuning pipeline (98.3% and 99.5%, respectively) the results confirm the marginal accuracy benefits of fine-tuning a transformer model. On the training time, we obtain an even more remarkable speedup($226.19\times$ and $185.05\times$, respectively).

Dataset	ΔAcc	Dataset	ΔAcc	Dataset	ΔAcc	Dataset	ΔAcc
AFHQ	−0.03%	CIF10	5.92%	Foot Ru	8.98%	Oxf Flo	12.0%
Beans	3.91%	CIF100	14.56%	Gemst	6.08%	Oxf Pet	5.75%
Best art	10.1%	Citr lea	2.67%	Hor Hu	0.00%	Plankt	0.00%
Boat	7.89%	Col hist	0.72%	iCub	0.12%	Sa Cov	−0.96%
Cal101	1.78%	Deep w	11.1%	Ind Fd	13.0%	St Cars	4.72%
Cassav	8.07%	DTD	9.98%	MkNoM	7.02%	St Dog	13.7%
Cat Do	1.14%	EuSAT	0.87%	Malaria	1.84%	Ten Fl	6.98%
Ch xra	−0.65%	FG Air	−9.40%	Meat qu	−0.63%	Weath	−0.53%

Table 3.5: Analysis on the transformer model for the top-tuning pipeline. $\Delta Acc = Acc_{\text{Transform}} - Acc_{\text{Conv}}$ is the accuracy gain of transformer model w.r.t. convolutional model

3.5.4 The importance of the pre-training dataset.

We now explore the dependency of the results on the pre-training dataset.
Specifically, we fix the pre-trained neural network architecture (DenseNet201), and we investigate the benefit of choosing ImageNet in terms of its size (number of images and classes) and image quality. To this purpose we consider three alternative pre-training datasets:

- **CIFAR100**: we consider CIFAR100 as a simplified version of ImageNet. With this dataset, we test the impact of reduced number of images, classes and image size.
- **ImageNet100**: we extract an ImageNet subset with the same number of images and classes of CIFAR100. To make this dataset as more similar as possible to CIFAR100 in terms of label semantic: 75% of labels selected from ImageNet is the same in CIFAR100, 15% of them are similar and 10% are different. With this dataset, we test the impact of image quality on the obtained results.
- **ImageNet 50k**: we extract an ImageNet subset that contains all the classes in ImageNet with the same number of data points contained in CIFAR100, that is 50k. The obtained dataset has 1000 classes with only 50 points per class. With this dataset, we test how the total number of classes affects the obtained results.

First, we train a DenseNet201 model from scratch on each of the three pre-traning datasets. Then, we apply the two investigated pipelines as described in section 3.4 on five target datasets: one of the most difficult and one of the easiest datasets for both approaches (DTD and CIFAR10, respectively), one where top-tuning approach outperforms fine-tuning (Citrus Leaves) and the opposite case (Deep Weeds). Lastly, we consider the fine-grained Oxford Flowers 102. With five target datasets, and three different pre-training, we perform 450 additional training instances.

Figure 3.11(Left) refers to top-tuning accuracy for each pre-train. ImageNet pre-training brings to the best results on the target datasets. ImageNet 50k is the best alternative, suggesting that the number of classes for the pre-training dataset has a great impact on the top-tuning approach. The

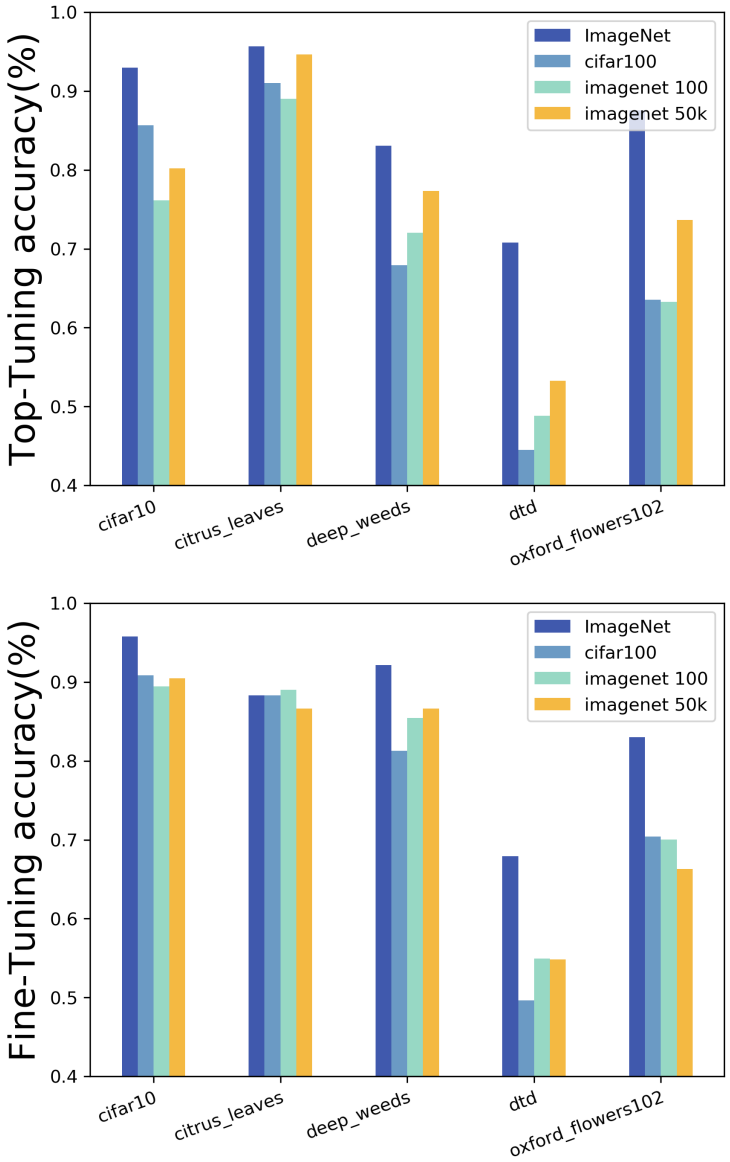


Figure 3.11: Overall results on different pre-trains. For each target dataset, each color represents one of the four different pre-trains datasets. (Left) Accuracies obtained by top-tuning approach (Right) Accuracies obtained by fine-tuning approach.

only exception corresponds to CIFAR10, where CIFAR100 corresponds to the best pre-train. This behavior depends on CIFAR10 being de facto a CIFAR100 subset.

Figure 3.11(Right) refers to fine-tuning accuracy for each pre-train. ImageNet pre-training corresponds to the best accuracy. The difference between the other pre-training configurations is marginal. This is probably due to the fine-tuning training procedure, which mitigates the weight

	Source \ Target↓	CIFAR10	Citrus Leaves	Deep Weeds	DTD	Oxf Flow
Top-Tun.	CIFAR100	**7.30**%	4.7%	15.2%	26.3%	24.0%
	ImageNet100	16.8%	6.7%	11.1%	22.0%	24.3%
	ImageNet 50k	12.7%	**1.0**%	**5.70**%	**17.5**%	**13.9**%
Fine-Tun.	CIFAR100	**4.9**%	0.0%	10.9%	18.3%	**12.6**%
	ImageNet100	6.3%	**−0.6**%	6.7%	**13.0**%	13.0%
	ImageNet 50k	5.3%	1.7%	**5.5**%	13.1%	16.7%

Table 3.6: Accuracy drops for pre-training alternative to ImageNet. Each column represents a different target dataset, each row a different pre-train source. The reported value corresponds to the accuracy drop w.r.t. the original ImageNet pre-training. The lower the value, the better.

of different pre-training. Table 3.6 summarizes the quantitative results in terms of accuracy drop w.r.t. the original ImageNet pre-training.

3.6 Discussion

A popular transfer learning solution to deal with the scarcity of training data is fine-tuning a pre-trained model. However, fine-tuning may still require significant computational resources, in terms of data need, training time, GPU-CPU involvement, and memory usage. This is due to back-propagation and the potentially huge amount of parameters involved.
An alternative solution, often dismissed by the research community as "too naif", consists in adopting a pre-trained model as-it-is as a features extractor, coupling it with an external head classifier. In this chapter we discussed the benefits of this simple alternative, we refer to as *top-tuning approach*, in particular when a fast kernel head classifier is adopted. To support our claim, we reported an extensive experimental analysis involving 32 target datasets, and 99 different settings, through 3460 distinct training processes.
Most of our experiments, confirm that fine-tuning has only marginal benefits, w.r.t. top-tuning approach. In 60% of our experiments, in fact, ΔAcc between fine-tuning and top-tuning is in range $[-2.5\%, +2.5\%]$. Furthermore, using a pre-trained model just as a feature extractor corresponds to a huge reduction in terms of training time, even from hours to a few minutes in different scenarios. Finally, our results showed that the marginal benefit of fine-tuning is low dependent on the neural network architecture used as a pre-trained model. On the other hand, the choice of an appropriate pre-training dataset has a significant impact on the obtained accuracy, especially in the case of top-tuning. The obtained results suggest that the variety of data plays a crucial role.

4 Representation efficiency

In the previous chapter, we compared two different machine learning models, evaluating both their accuracy and training time. We have seen that is possible to make the training process efficient with a limited impact on the accuracy performance. In this chapter we focus on data representation efficiency. To address this problem, as a reference task we select image clustering, a standard unsupervised problem, focusing on plankton images. A well-known problematic in clustering is the increasing hardness of the problem when it deals with high-dimensional data. Finding the optimal solution of a clustering problem is NP-hard and it is very costly or nearly impossible to work with samples having thousands of dimensions. Indeed, for a clustering algorithm is impossible to deal directly with images that, even with a low resolution, can have dozen of thousands of dimensions. To this purpose we implement an unsupervised pipeline that projects the input to a latent space with reduced dimension, making the clustering operation doable. We test our pipeline effectiveness in the plankton monitoring context where operating in an unsupervised manner is crucial.

Monitoring plankton populations in situ are fundamental to preserve the aquatic ecosystem. Plankton microorganisms are in fact susceptible to minor environmental perturbations, that can reflect into consequent morphological and dynamical modifications. Nowadays, the availability of advanced automatic or semi-automatic acquisition systems has been allowing the production of an increasingly large amount of plankton image data. The adoption of machine learning algorithms to classify such data may be affected by the significant cost of manual annotation, due to both the huge quantity of acquired data and the numerosity of plankton species. To address these challenges, we propose an efficient unsupervised learning pipeline to provide accurate classification of plankton microorganisms. We build a set of image descriptors exploiting a two-step procedure. First, a Variational Autoencoder (VAE) is trained on features extracted by a pre-trained neural network. We then use the learned latent space as an image descriptor for clustering.

We compare our method with state-of-the-art unsupervised approaches, where a set of pre-defined hand-crafted features is used for the clustering of plankton images. The proposed pipeline outperforms the benchmark algorithms for all the plankton datasets included in our analysis, providing better image embedding properties. This chapter is organized as follows. In section 4.1 we provide the motivations that inspired our research on this topic. In section 4.3, we outline and review the proposed pipeline in detail and some common architectures that are used in our study. In section 4.4, we describe the datasets included in our analysis, present the experimental setup and discuss our results. In section 4.5 we lay out our conclusions and discuss future developments.

4.1 Motivations

Plankton is a collection of aquatic microorganisms floating passively in the water. It plays a big role in the marine ecosystem. Plankton is indeed at the basis of the aquatic food chain, with phytoplankton being estimated to have produced approximately 50% of the total oxygen in our atmosphere [17]. Recently, it has been proved that local or global perturbations of the aquatic environment, either natural or man-caused, are profoundly impacting both the composition and dynamics of plankton populations [28].
As stated in [212], plankton microorganisms react to even minimal changes in the environment with morphological and dynamical modifications, so they can be regarded as biosensors, reflecting the overall health of the oceans. Thus, detecting and studying plankton population in situ is paramount to protect marine ecosystems [173].
Recently, the development of advanced tools for automatic high-throughput in situ microscopy allowed real-time observation of plankton species [72, 214]. Such systems acquire a huge amount of image data for which manual identification is impractical [269]. Hence, machine learning has nowadays become one of the most studied approaches for the characterization of plankton data [20, 24, 25, 49, 107, 212, 254, 258, 270, 321]. In particular, there has been a surge in interest towards models based on artificial neural networks (ANNs), due to their successes in big data problems and their high expressive power, specifically in the form of convolutional neural networks (CNNs) [50, 53, 174, 230, 235].
A central element in the process of characterizing plankton data is feature selection. The two main approaches are represented by hand-engineered features, such as geometric or Fourier features [319, 321], and deep features, typically based on deep CNNs [50, 286]. Recent works [156, 173, 206] have also explored the possibility to use pre-training and transfer learning (both in- and out-of-domain) to enhance the expressivity of the models and alleviate the computational cost associated with the training of deep CNNs. In [64], the authors demonstrate how context metadata, such as temperature, location, and salinity, improves the performance of classifiers. The literature is however largely dedicated to supervised approaches, for which accurate models can be obtained at the high cost of providing manually annotated data. Besides the development of solutions aimed at automatizing the labeling procedures [110, 257], research on unsupervised models [214] is crucial to avoid bottlenecks in our ability to process information [180].

We propose a new unsupervised approach for the characterization of plankton images. There are more than 4000 existing plankton species, and many of them are very similar, from a morphological point of view, making this problem very challenging[215].
We leverage neural network models that are pre-trained on large general-purpose datasets, such as ImageNet, to extract expressive feature maps in an efficient way, without fine-tuning. We then use an encoder-decoder network architecture to perform dimensionality reduction, producing low-dimensional embedded features that can then be fed to a clustering algorithm.
We tested our pipeline on three plankton datasets with different characteristics, showing that our results surpass state-of-the-art approaches where hand-crafted features (i.e., geometric and texture descriptors) are engineered and used for clustering. Specifically, the main contributions of this work are:

1. A new variational autoencoder-based methodology for efficient unsupervised learning, leveraging the variational autoencoder low dimensional and regular latent space. A key novelty consists in using pre-trained features as informative input to the variational autoencoder, to obtain high-quality descriptive features. Our findings confirm that variational autoencoder can be used not only for generation purposes but also to discover patterns from data in an unsupervised fashion.

2. We show that the high-quality low-dimensional embedding produced by our approach appears to be informative and allows the effective usage of unsupervised machine learning algorithms, such as k-means or Gaussian mixture models, taming the curse of dimensionality.

3. We further show that the produced embedding also leads to high accuracy when used as compressed input to supervised models, which can be trained quickly and on limited resources.
   The proposed pipeline is efficient and ideal for in situ analysis as well as for offline investigations of plankton data.

4.2 Background

Dimensionality reduction: in the machine learning field, the *dimensionality reduction* process consists in reducing the number of features describing some data. It can be a set of powerful techniques in many situations that require low dimensional data such as data visualization, storage, or even when we want to reduce the computation. Such a reduction can be done in different ways. Two common methodologies to perform it are *feature selection*[35] and *features extraction*[248].
The first approach corresponds to producing a new description of our data by selecting just a subset of the original features. Usually, feature selection techniques are categorized into three different groups: *wrapper* methods, *filter* methods, and *embedded* methods.
The second approach consists in producing a set of new features based on the old ones. As we were mentioning a common scenario for dimensionality reduction is data visualization. Some well-known techniques on this topic are: t-distributed stochastic neighbor embedding(t-SNE)[176], Principal Component Analysis(PCA)[218], and Kernel-PCA[256].
For instance, the PCA idea is to build a set of new independent features that are a linear combinations of the old features. The problem we want to optimize is to project the data on the subspace defined by these new features, corresponding to an orthogonal basis of new features, and we want the new samples in such space to be as close as possible to the initial data according to some distance measure. We could say that Principal Component Analysis will search for the best linear subspace of the initial space such that the error of approximating the data by their projections on this subspace is as small as possible.

Autoencoders autoencoders [99, 140] general idea is simple, consisting in setting an encoder and a decoder as neural networks. The procedure consists in learning an optimal encoding-decoding

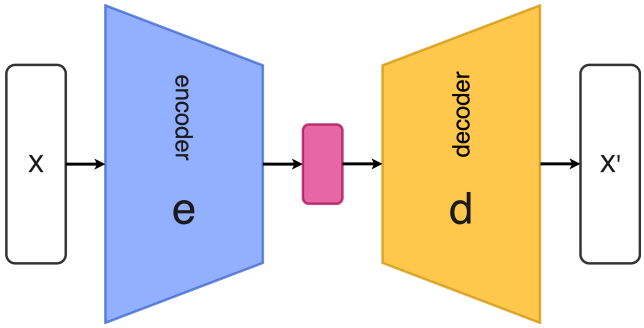


Figure 4.1: The scheme of an autoencoder.

scheme via an iterative minimization process. In other words, autoencoders are unsupervised machine learning models trained to reproduce their inputs while learning a lower dimensional latent representation. Therefore, at each iteration we feed the autoencoder architecture with data samples, comparing the encoded-decoded output with the initial data. Then, we backpropagate the error through the architecture to update the weights of the networks. Intuitively, the autoencoder architecture creates a bottleneck between the encoder and the decoder for data. This way it ensures only the main structured part of the information can go through and be reconstructed by the decoder.
Concretely, autoencoders address this task by learning an *encoder* function e which maps input data $x \in \mathbb{R}^d$ in a latent space $Z \subseteq \mathbb{R}^p$ with $0 < p < d$ and a *decoder* function d which maps a latent representation $z = e(x)$ back to the original input x. These maps are typically parametrized by neural networks. Let us denote the output with $x' = d(e(x))$, autoencoders are then trained to minimize the *reconstruction loss*

$$l(x, x') = \left\|x - x'\right\|^2 = \|x - d(e(x))\|^2 \tag{4.1}$$

They can be interpreted as non-linear dimensionality reduction models.
In the special case where both encoder and decoder have only one layer without non-linearity, we simply have a linear transformation that can be expressed as matrices. This special case is strongly tied to the PCA as we are looking for the best linear subspace to project data on with as little information loss as possible.
We can show the architecture of a generic autoencoder in Figure 4.1

Once the model has been trained, we have both an encoder and a decoder capable of compressing the input x to a latent space with a fixed dimension. Usually, we would like to have a way to produce some new content. We could suppose that, if the latent space is regular enough after the training procedure, i.e. well "organized" by the encoder, we could sample a random point from it and decode such a sample to get new content.
However, the regularity of the latent space described by autoencoders is difficult to ensure and depends on the distribution of the data, the latent space dimension, and the encoder architecture. In general, we do not have any guarantees that the encoder will organize the latent space in a regularized way that can be exploited to generate new points. More often the encoder will just try to separate the samples without taking into account the data structure, producing a severe overfit-

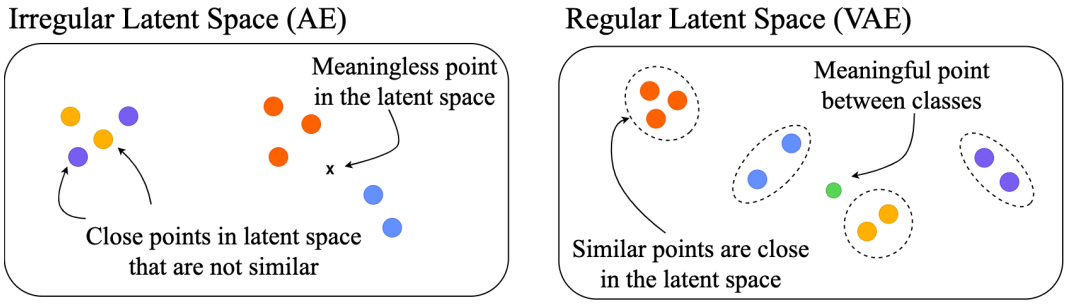


Figure 4.2: Differences between a non-regular and a regular latent space.

ting situation in general. Indeed, autoencoders provide good performances in data compression, but they cannot be used to generate new instances since they do not impose any structure on the latent space, see Figure 4.2.
Moreover, the overfitting phenomenon can be explained mathematically by considering again Equation 4.1 where only the reconstruction loss is considered without any kind of regularization. Such a situation might lead to overfitting.

VARIATIONAL AUTOENCODER. variational autoencoders [134] can be seen as a way to improve the structure of the latent space by encoding the input into a multivariate latent distribution.
Similarly to an autoencoder, its variational counterpart is a model composed of both an encoder and a decoder. The model, just like we showed in Equation 4.1 is trained to minimize the reconstruction error obtained by comparing the encoded-decoded data x' and the initial data x. However, in order to introduce regularisation of the latent space, we can slightly modify the encoding-decoding process: instead of mapping an input as a single point in the latent space, we encode it as a distribution over the latent space.
In principle, we could map our input x to an arbitrary desired distribution, i.e. to the parameters defining such distribution. In real scenarios, more concretely, the encoder maps each input to a multivariate Gaussian distribution $q(z|x)$ with mean and variance parametrized by neural networks. In a few lines, we are going to see that the loss used to train the model contains a quantity called *Kullbach-Leibler diveregence* computing a mathematical divergence measure between two distributions. When the distributions involved are Gaussian, the Kullbach-Leibler diveregence has a closed form, i.e. it can be directly expressed in terms of the means and covariance matrices of the two distributions.
Overall with a standard variational autoencoder, we are mapping our input x to a latent space such that the probability of sampling a point z given x as input is modeled by a gaussian distribution

$$q(z|x) = \mathcal{G}(\mu(x), \sigma(x)). \tag{4.2}$$

The reason to encode an input as a distribution instead of a single point lies in the possibility to express very naturally the latent space regularisation: the distributions returned by the encoder are enforced to be close to a standard normal distribution. We will see in Equation 4.3 that this condition is ensured by a term included in the loss function.
The sample from the latent distribution $z \sim q(z|x)$ is then decoded and the resulting output

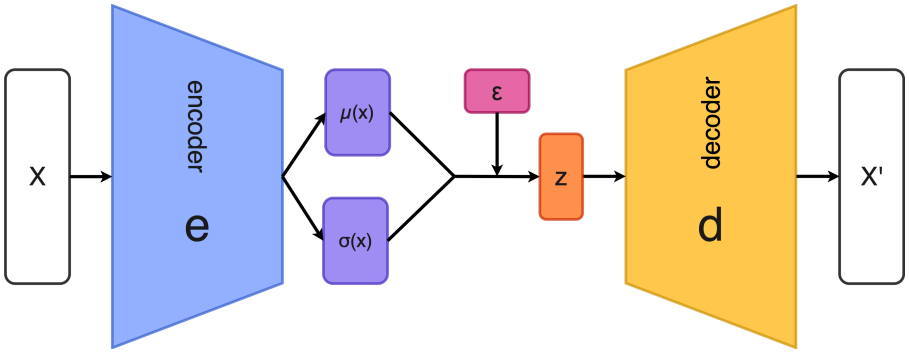


Figure 4.3: The scheme of a variational autoencoder.

$x' = d(z)$ is used to compute the error. The loss function includes reconstruction terms, as in the AE loss, pushing the model to be as performant as possible. Moreover, a regularization term tends to organize the latent space by forcing the model to return a distribution close to a standard normal distribution. Such regularization term is given by the Kullback-Leibler divergence between the encoded distribution and the prior on the latent representation, assumed normal $p(z) = \mathcal{N}(0, I)$. The total loss reads as

$$l(x, x') = \left\|x - x'\right\|^2 + D_{KL}(\mathcal{G}(\mu(x), \sigma(x)), \mathcal{N}(0, I)). \tag{4.3}$$

In order to allow error backpropagation through the encoder, the sampling step is expressed via the *reparametrization trick*:

$$z = \mu(x) + \epsilon\sigma(x) \qquad \epsilon \sim \mathcal{N}(0, I). \tag{4.4}$$

Ultimately, it is the regularization term that forces the model to learn meaningful latent space representations. Without it, the variational autoencoder would try to simply reconstruct the input as closely as possible, for instance, by mapping each input to a delta distribution in the latent space, similarly to an AE, see Figure 4.2. See Figure 4.3 for a schematic representation of a variational autoencoder architecture. A more detailed and formal discussion can be found in [134, 135].

Clustering accuracy In literature, the quality of a clustering algorithm can be evaluated with different metrics[13] such as purity, normalized mutual information, and rand index. In our work, we decided to use purity, defined as

$$\text{purity}(\Omega, C) = \frac{1}{N}\sum_k \max_j |w_k \cap c_j|, \tag{4.5}$$

where Ω is the set of clusters and C is the set of ground-truth classes. Every cluster is then associated with the most represented class.
A purity value of one means that clusters perfectly overlap with the ground truth. An instance of purity computation is shown in Figure 4.4. In the first cluster, the most frequent class is the blue one, with five instances. In the second cluster, the prevalent class is the purple one, with four sam-

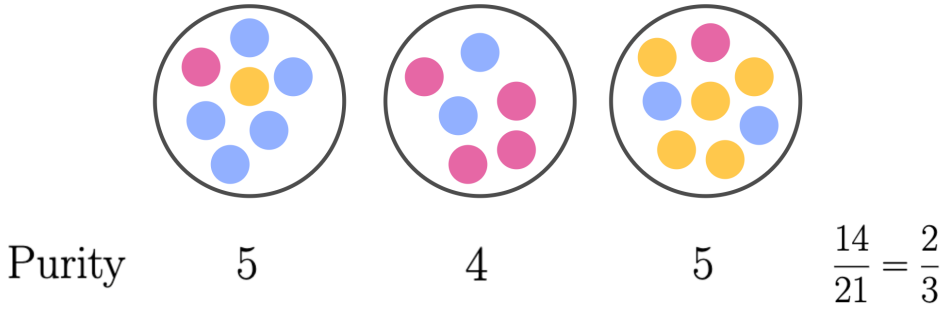


Figure 4.4: An instance of purity computation. First cluster purity corresponds to the number of samples from the most frequent class, i.e. is equal to five. The second cluster purity is equal to four. The last cluster purity is again equal to five. By summing up the cluster purities and normalizing by the total amount of samples we obtain purity $= \frac{2}{3}$

ples. In the last cluster, the most frequent class is the yellow one, with four samples. Summing up the per-cluster purity we obtain a value of 14. Then we can normalize the obtained value, dividing by the total number of samples, equal to 21. Overall we obtain a purity equal to purity $= \frac{14}{21} = \frac{2}{3}$.

Purity decreases when samples from the same class are spread among different clusters, or separate clusters are assigned to the same class. In plankton image analysis, several species have nearly indistinguishable morphological features. Thus, the clustering algorithm can potentially group such species into the same cluster. In our results, we refer to the number of ground truth species overlapping with the same clusters as *number of overlaps*.
A number of overlaps equal to zero mean that the correspondence between ground truth species and the cluster is 1:1, meaning that each cluster represents a unique class (i.e., there are no two or more species that mostly overlap with the same cluster).
We adopted the customized purity implementation described in [214], where the number of overlaps is introduced and used to evaluate clustering performances together with the purity.

Pre-training as we saw in the previous chapter, in a typical training scenario, the weights of a model are adjusted, starting from a random initialization, to optimize a measure that is determined by the task at hand, such as accuracy for classification, over a validation set. However, the quality of the results can be impacted by the scarcity of data or by the computational cost of training, especially in the context of deep learning.
One common solution to this kind of problem is represented by the use of *pre-trained* models. The goal is to exploit large state-of-the-art models, such as DenseNet[108], ResNet[277], Xception[41] or transformers[60] in which the weights have been already optimized for a similar task. Usually, the pre-training is performed on a large and general-purpose dataset, such as the ImageNet dataset [51]. A pre-trained model can then be used as is, possibly embedded with frozen weights in a larger model, or by fine-tuning it, hence further optimizing its weights on a specific task. In our study, we use a pre-trained model, DenseNet201 [108] as a feature extractor, without fine-tuning. This is a cheap and efficient way to produce better representations of our input images, as it only involves an inference step.

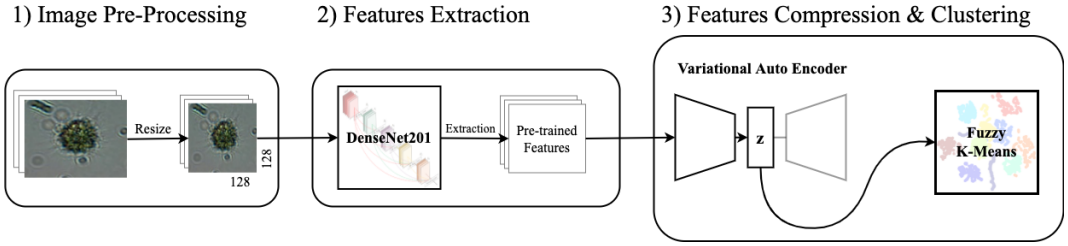


Figure 4.5: The proposed pipeline is made by three steps: Image Pre-processing, Features Extraction, Features compression, and Clustering

4.3 Methodology

Given the above discussion, in this section we first describe the proposed pipeline we utilized in our work. Then we present technical details about the conducted experiments. Lastly, we report details about the datasets used in the empirical analysis.

Pipeline: we first describe our pipeline, which is schematically represented in Figure 4.5. It includes the following steps:

1. Pre-processing: images are resized to the same size of 128×128 pixels. Then are normalized to be compatible with the pre-trained neural network used in the subsequent step.

2. Features extraction: pre-processed images are given as input to a deep neural network pre-trained on the ImageNet dataset[51]. The output of this step is a high-dimensional feature vector for each image.

3. Dimensionality reduction and clustering: feature vectors are reshaped and used as input to train a convolutional Variational Auto Encoder (VAE). The learned latent space is exploited to map input features into a low-dimensional embedding that is finally fed to a clustering algorithm.

Experiment Details: in our experiments, the entire pipeline is implemented in PyTorch [217]. Feature extraction is performed using DenseNet201, pre-trained on ImageNet without further fine-tuning. The network has approximately 20M parameters and achieves a top-5 accuracy of 0.9446 on the ImageNet validation set [108]. Input images are resized to 128×128 pixels and normalized accordingly to DenseNet201 specification. With our input format, DenseNet201 produces an output shape given by $(\text{channels}, \text{height}, \text{width}) = (1920, 4, 4)$. The output is then reshaped to obtain higher performances with the convolutional (V)AE. We performed several tests, noticing better performances with a lower number of input channels and a higher width and height compared to the original output shape. Thus, we considered two candidate shapes: the first one with an approximately homogeneous distribution among channels, width, and height $r_1 = (30, 32, 32)$ and another one with a lower number of channels and a corresponding higher width $r_2 = (3, 32, 320)$. We report in section 4.4 the results obtained with both shapes. The

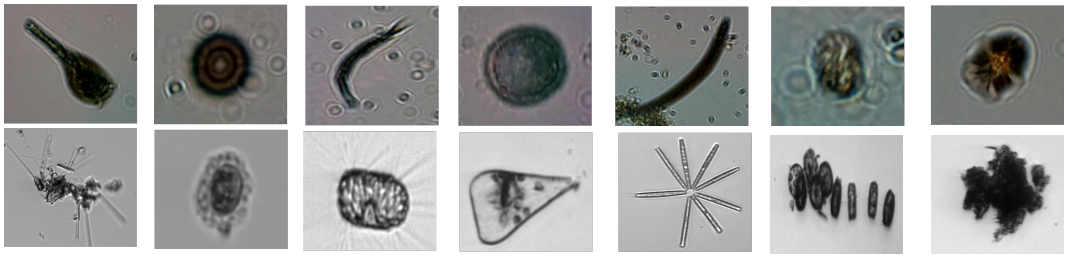


Figure 4.6: Sample images from seven different classes included in the datasets considered for our analysis. (Top): samples from the lensless dataset. (Bottom): Samples from the WHOI dataset.

AE and variational autoencoder used to produce our results are composed of an encoder made of three convolutional layers. The decoder is implemented with three convolutional transpose layers. The variational autoencoder has two additional dense layers after the encoder, to parametrize the mean and the log variance of the multivariate distributions in the latent space. AE and variational autoencoder are trained for a total of 100 epochs using SGD and Adam optimizer respectively. Learning rate is initialized to 0.001 for both optimizers with an exponential learning rate decay for SGD. Batch size is set to 64. Hyperparameters were selected with a cross-validation procedure, adopting a k-fold approach (k=5). We used the scikit-fuzzy [303] implementation of Fuzzy K-Means [63] for clustering the latent space data. Results in Table 4.5 are obtained using the sklearn implementation of Kernel ridge [252] and implementing a neural network classifier in TensorFlow [1]. The neural network is composed of 2 hidden dense layers of 256 and 128 neurons. It is trained for 100 epochs, with a batch size of 32, and using stochastic gradient descent (SGD) optimizer with a learning rate of 0.01. For the kernel ridge, we used a Gaussian kernel. Hyper-parameters are tuned using grid search maximizing the classification accuracy.

Datasets details: We considered three datasets for the evaluation of the proposed pipeline. In this section, we will provide few details on each of them. See Figure 4.6 for an example of the included plankton species.

- Lensless: the Lensless microscope dataset was introduced in [214]. It consists of images acquired using a lensless microscope, extracted from videos. More precisely is a collection of videos containing ten freshwater species of plankton captured with a lensless microscope[326]. Each video is ten seconds long and contains one or more species. As the method is unsupervised, no labels are provided to the classifier during training It includes 10 classes, with 640 color images each. Figure 4.6(top) shows sample images for seven classes included in the dataset.

- WHOI 40: this dataset was released in [214] and it is a subset of the Woods Hole Oceanographic Institution (WHOI) Plankton Dataset[1] (years 2011-2014). See Figure 4.6(bottom) for an example of seven classes included in the dataset.

[1] https://hdl.handle.net/10.1575/1912/7341

The WHOI provides a public dataset comprising millions of still monochromatic images of microscopic marine plankton, collected in situ by an automated submersible imaging-in-flow cytometry exploiting an Imaging FlowCytobot (IFCB). To use this dataset as a benchmark to test our unsupervised classifier, we extract a collection of 40 species of plankton (100 images per species, randomly selected), using both the segmented binary image and the portion of the gray-scale image containing the plankton cell body.

- WHOI 22: this dataset was introduced in [269], and contains images extracted from the WHOI dataset. The authors, after a manual inspection of WHOI dataset, defined 22 explicit categories that represent subjective consideration of taxonomic knowledge, ecological perspective, and practical issues regarding groupings that can be feasibly distinguished from morphology visible in the images. Many categories correspond to phytoplankton taxa at the genus level or groups of a few morphologically similar genera. Overall it includes 22 species, 300 grayscale images each.

4.4 Experiments

We applied our pipeline to the three plankton datasets, described in the previous section. First, we verified the impact on the models' performances of the type of input used for training. To do this, we compared both AE and variational autoencoder trained either on the original images or on the pre-trained features reshaped as described in previous section. We performed multiple experiments considering five different latent space sizes, evaluating the results in terms of clustering purity and a number of overlaps (see Section 4.3) on the available test set, for each of the considered datasets. We specify that, since WHOI 40 is not originally distributed with a specific test set, we therefore performed an 80:20 train/test split as a preliminary step for this dataset. A purity of one with zero overlaps corresponds to clusters perfectly agreeing with the ground truth.

To prove the robustness of our method, we adopted a k-fold approach (k=5) repeating each experiment five times, with different train and validation splits. We then report mean and standard deviation for the purity and the number of overlaps on the test set for each dataset, in Tables 4.1, 4.2, 4.3.

Table 4.1 shows our results on the Lensless dataset. As we can observe, using pre-trained features significantly increases the purity with an average improvement of 30% over the original images, considering both types of reshaping.
It is possible to appreciate such improvement in Figure 4.7, where an instance of learned latent space, with and without, pre-trained features is shown. Thus, motivating the adoption of pre-trained features in our pipeline. Moreover, using an input feature reshape with three channels $r_2 = (3, 32, 320)$ results in a slight improvement over the case with more channels $r_1 = (30, 32, 32)$. It is worth noticing that the variational autoencoder generally outperforms the AE for all the con-

sidered inputs and latent space sizes, with $Z = 500$ giving an average test purity of 0.98 without any overlap.

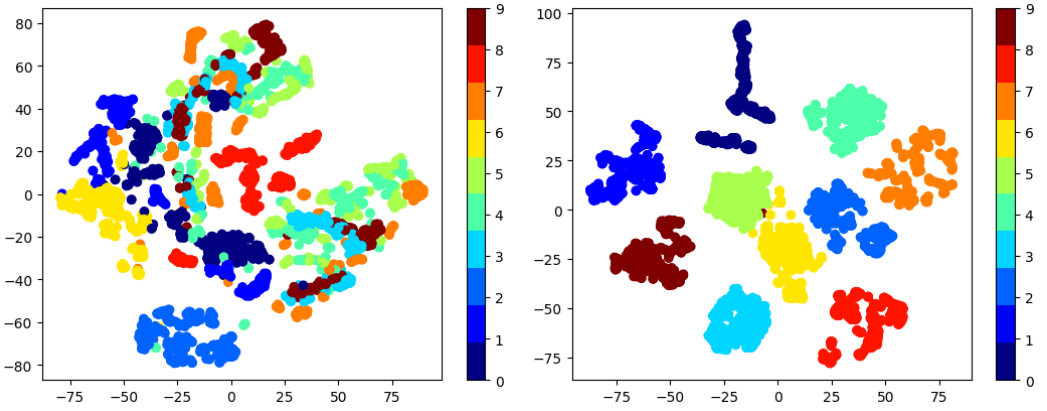


Figure 4.7: t-SNE of the latent space learned by a variational autoencoder on the lensless dataset. (Left) The learned latent space when we directly used as input the images

Table 4.2 and Table 4.3 show our results on the WHOI 40 dataset and the WHOI 22, respectively. These two datasets pose different challenges. The WHOI 40 includes a relatively high number of classes, with a more coarse granularity when compared to WHOI 22. Moreover, WHOI 22 includes images that contain multiple plankton cells, as well as a class for heterogeneous detritus (see, for instance, the last two images on the right in Figure 4.6(bottom)). In Table 4.1, 4.2 and 4.3, the notation $FE_{r_{1/2}}$ refers to pre-trained features reshaped as $r_{1/2}$.
As we can observe, using an input feature reshape with three channels $r_2 = (3, 32, 320)$ results now in a significant improvement for variational autoencoder over the case with more channels $r_1 = (30, 32, 32)$.
Generally, the best performances correspond to a latent space size $Z = 500$, with a significant improvement with respect to small sizes. The highest average test purity for the WHOI 40 corresponds to 0.77 with four overlaps (i.e., four couples of ground truth species overlapping with the same clusters). The highest average test purity for the WHOI 22 is equal to 0.68 with two overlaps.

We benchmarked our results using a state-of-the-art unsupervised learning pipeline based on a set of 131 hand-crafted features and fuzzy k-means. The method was introduced in [214], the same paper where two of the datasets used in this analysis (i.e., Lensless and WHOI 40) were released. To our knowledge, a state-of-the-art clustering benchmark was not available for the WHOI 22. To benchmark our results, we used the approach described in [214] to perform clustering on the WHOI 22, based on the pipeline described in their paper.
As shown in Table 4.4, our best embedding (feature reshape r_2, variational autoencoder with latent space size $Z = 500$), outperforms the state-of-the-art approach for all the datasets included in our work. Our results are marginally better in terms of purity for the Lensless dataset. They are instead significantly better when considering the two more challenging datasets WHOI 40 and WHOI 22. We obtain not only a higher purity but also a reduction in the number of overlaps. It

is worth noticing that the purity and number of overlaps standard deviation among the different experimental runs is low for all the datasets, proving the robustness of the proposed methodology.

Algorithm/Z	10	30	50	100	500
image-AE	0.47 ± 0.05 (1.4 ± 0.5)	0.53 ± 0.03 (1.4 ± 0.49)	0.56 ± 0.02 (1.6 ± 0.49)	0.55 ± 0.01 (1.8 ± 0.4)	0.60 ± 0.01 (1.4 ± 0.49)
image-VAE	0.53 ± 0.017 (1.4 ± 0.5)	0.55 ± 0.04 (1.6 ± 0.49)	0.58 ± 0.01 (2.0 ± 0.63)	0.59 ± 0.01 (1.6 ± 0.48)	0.62 ± 0.01 (2.0 ± 0.0)
FE_{r_1}-AE	0.38 ± 0.02 (1.6 ± 0.5)	0.62 ± 0.02 (1.2 ± 0.40)	0.75 ± 0.04 (0.4 ± 0.49)	0.84 ± 0.03 (0.2 ± 0.4)	0.95 ± 0.01 (0.0 ± 0.0)
FE_{r_1}-VAE	0.94 ± 0.01 (0.0 ± 0.0)	0.97 ± 0.01 (0.0 ± 0.0)	0.97 ± 0.01 (0.0 ± 0.0)	0.97 ± 0.01 (0.0 ± 0.0)	0.97 ± 0.002 (0.0 ± 0.0)
FE_{r_2}-AE	0.40 ± 0.05 (1.6 ± 0.5)	0.70 ± 0.02 (0.8 ± 0.4)	0.75 ± 0.03 (0.4 ± 0.49)	0.86 ± 0.05 (0.2 ± 0.4)	0.97 ± 0.01 (0.0 ± 0.0)
FE_{r_2}-VAE	0.98 ± 0.01 (0.0 ± 0.0)	0.98 ± 0.03 (0.0 ± 0.0)	0.98 ± 0.01 (0.0 ± 0.0)	0.98 ± 0.02 (0.0 ± 0.0)	0.98 ± 0.02 (0.0 ± 0.0)

Table 4.1: Clustering purity on Lensless for latent space size Z. The first two rows consider the image as input to the variational autoencoder. The next two consider features with the approximate homogeneous distribution among channels, width, and height $r_1 = (30, 32, 32)$. The last two rows consider features with a lower number of channels and a corresponding higher width $r_2 = (3, 32, 320)$

Algorithm/Z	10	30	50	100	500
FE_{r_1}-AE	0.25 ± 0.01 (11.8 ± 1.16)	0.41 ± 0.03 (8.0 ± 1.26)	0.50 ± 0.02 (6.8 ± 1.46)	0.59 ± 0.01 (6.6 ± 1.01)	0.69 ± 0.01 (4.8 ± 0.74)
FE_{r_1}-VAE	0.21 ± 0.01 (13.8 ± 0.99)	0.22 ± 0.02 (14.4 ± 1.62)	0.30 ± 0.04 (13.2 ± 1.93)	0.40 ± 0.08 (10.6 ± 1.2)	0.67 ± 0.04 (5.4 ± 1.35)
FE_{r_2}-AE	0.25 ± 0.01 (13.2 ± 0.74)	0.42 ± 0.03 (9.0 ± 0.63)	0.51 ± 0.01 (6.2 ± 0.40)	0.62 ± 0.01 (5.8 ± 1.16)	0.72 ± 0.006 (4.6 ± 0.80)
FE_{r_2}-VAE	0.66 ± 0.01 (5.8 ± 0.75)	0.71 ± 0.02 (5.8 ± 1.16)	0.73 ± 0.02 (5.2 ± 0.98)	0.77 ± 0.01 (3.8 ± 1.16)	0.77 ± 0.01 (4.0 ± 0.63)

Table 4.2: Clustering purity on WHOI 40 for latent space size Z. In this table, only the two reshapes $r_1 = (30, 32, 32)$ and $r_2 = (3, 32, 320)$ are considered.

Algorithm/Z	10	30	50	100	500
FE_{r_1}-AE	0.22 ± 0.02 (5.0 ± 1.1)	0.40 ± 0.02 (2.6 ± 0.8)	0.48 ± 0.01 (2.4 ± 0.48)	0.55 ± 0.01 (1.0 ± 0.0)	0.65 ± 0.02 (1.4 ± 0.49)
FE_{r_1}-VAE	0.22 ± 0.02 (7.6 ± 1.2)	0.30 ± 0.01 (5.8 ± 1.93)	0.33 ± 0.07 (5.8 ± 1.32)	0.44 ± 0.07 (3.8 ± 1.72)	0.68 ± 0.01 (2.0 ± 0.4)
FE_{r_2}-AE	0.24 ± 0.02 (4.2 ± 1.16)	0.38 ± 0.01 (3.6 ± 0.49)	0.51 ± 0.02 (2.2 ± 0.74)	0.60 ± 0.02 (1.6 ± 0.49)	0.66 ± 0.01 (1.2 ± 0.4)
FE_{r_2}-VAE	0.63 ± 0.004 (2.0 ± 0.63)	0.66 ± 0.01 (1.6 ± 0.49)	0.68 ± 0.005 (1.6 ± 0.49)	0.68 ± 0.006 (1.4 ± 0.5)	0.68 ± 0.01 (1.8 ± 0.4)

Table 4.3: Clustering purity on WHOI 22 for latent space size Z. In this table only the two reshapes $r_1 = (30, 32, 32)$ and $r_2 = (3, 32, 320)$ are considered.

Algorithm/Dataset	Lensless	WHOI 40	WHOI 22
Pipeline from [214]	0.93 (0) [214]	0.71 (5) [214]	0.56 (3)
Ours	**0.98 (0)**	**0.77 (4)**	**0.68 (2)**

Table 4.4: Purity comparison between our best average results and the available state-of-the-art.

Finally, we benchmarked the proposed approach and the quality of our lower dimensional embeddings, with respect to supervised algorithms. To this end, we considered two different classifiers i.e., a Fully Connected neural network (FC) and ridge regression, training on top of our best embedding for all three datasets. We performed five different experiments, comparing the resulting best test classification accuracy against available state-of-the-art supervised approaches, based on different types of classifiers. For the WHOI 22, we compared our results with the ones reported in [321] and [269], where a set of hand-crafted features, appositely designed and selected for plankton images is engineered, and fed to an SVM classifier. For the WHOI 40 dataset and the Lensless dataset, we compared our results with the ones reported in [214], where a set of 131 hand-crafted features is extracted and fed to a two layers neural network (for Lensless) and a Random Forest (RF) classifier (for WHOI-40).
As we can observe in Table 4.5, a ridge regression classifier on top of our best embedding outperforms the state-of-the-art supervised classification results, for all the three datasets included in our analysis (1.000 versus 0.980, 0.957 versus 0.790 and 0.883 versus 0.880, for the Lensless, the WHOI 40 and the WHOI 22, respectively).
These results proved that our embedding provides a better representation of the input data compared to hand-crafted features, specifically designed for plankton cells.
For completeness, in Table 4.6, we indicated the total time required to execute our pipeline for the best model.

Algorithm/Dataset	Lensless	WHOI 40	WHOI 22
Our embedding + FC	1.000	0.948	0.868
Our embedding + ridge	**1.000**	**0.957**	**0.883**
Features from [214] + FC	0.980[214]	—	—
Features from [214] + RF	—	0.790[214]	—
Features from [321] + SVM	—	—	0.880[321]
Features from [269] + SVM	—	—	0.880[269]

Table 4.5: Supervised learning benchmarks. The state-of-the-art results are directly taken from the original publications.

Lensless	WHOI 40	WHOI 22
(238.56 ± 2.00) s	(195.75 ± 1.04) s	(194.93 ± 2.51) s

Table 4.6: Time requested for the execution of our pipeline (best model).

4.5 Discussion

In this thesis, we introduced an efficient unsupervised learning pipeline for the characterization of plankton images. Input images are pre-processed and fed to a neural network (DenseNet201 in our experiments) pre-trained on ImageNet, without fine-tuning. The resulting set of features are used as inputs to train an encoder-decoder neural network (an autoencoder or a variational autoencoder) and the resulting latent space representations of the inputs are used as a lower dimensional set of embedded features, that are then passed to a clustering algorithm (a fuzzy k-means in our experiments).
We exploited three datasets extracted from two different acquisition systems and posing different challenges. The Lensless dataset, with a coarse granularity but is characterized by low-resolution and noisy images; the WHOI 40, is less coarse with respect to the Lensless and with a relatively higher number of classes (40) and the WHOI 22, with fine-grained features. We showed that a variational autoencoder with latent space size $Z = 500$ and pre-trained input features, reshaped as (channels, height, width) $= (3, 32, 320)$, generally gives the best results in terms of purity and overlaps number, for all the datasets included in our work. We further showed that our approach outperforms state-of-the-art unsupervised learning approaches [214] where hand-crafted features are engineered and used for clustering.
We further proved the quality of the embedded features produced by our pipeline using a supervised classification framework (in terms of test accuracy). Precisely, we showed that our embedding features coupled to a ridge regression classifier outperform state-of-the-art classifiers where hand-crafted features are used as input for SVM [269, 321], fully connected neural networks and random forests [214].
The variational autoencoder architecture considered for the tests of the proposed pipeline is shallow, and the ImageNet pre-trained DenseNet201 is only used for feature extraction. Hence, the pipeline can be run on embedded devices (e.g., a Rasperry-Pi), allowing for in situ recognition, which may be fundamental for plankton population studies.
It is worth underlining that our pipeline is general with respect to the source of input data. A com-

plete analysis of the performances of another kind of data is out of the scope of this thesis, however, the proposed pipeline can be potentially extended to other domains. A significant advantage of our approach is indeed represented by the usage of pre-trained features and an encoder-decoder network to obtain a quality embedding for plankton data.
Differently from the hand-crafted features exploited in the works we used as benchmarks, these features do not require any engineering nor tuning at any step to adapt to a specific dataset. For instance, the computation of shape-descriptors is instead a multi step-process, starting with a segmentation algorithm (as done in [214, 269]) to identify the plankton cell. The segmentation can be performed using image processing tools or deep learning. The quality of the features is then highly dependent on the quality of the segmentation, which requires tuning according to specific properties of the dataset (e.g., acquisition system, brightness, noise). Avoiding this step, our pipeline represents an efficient approach with a significant advantage in terms of time and resources.
Finally, as a further development, the implementation of an end-to-end solution would be crucial for easy deployment in real-life scenarios. Additionally, it would be interesting and useful to test the approach for anomaly detection. These aspects are currently under study.

5 Datasets Similarity

In the previous chapters, we have seen the outstanding deep learning performances in the computer vision field where they became a de facto standard. Nevertheless, despite this great success, it is still troublesome to compare different computer vision problems or determine whether a problem is harder than another.
Another challenge is related to the number of different tasks we can have in computer vision. Indeed, we can have a plethora of different tasks including classification, identification, detection, recognition, estimation of pose, egomotion or optical flow.
To define more specific boundaries, we decided to focus on image classification and even in this context is arduous to compare different classification tasks. This complexity can be appreciated by trying to answer to the following question: given two image datasets how can we measure how similar they are? A related question to this is: can we compute a *distance* measure between the two datasets?
Intuitively, it is easy to understand that a dogs image dataset is going to be very different from a dataset about planes. It is more difficult to quantify this difference and build a relation between more than two datasets. Indeed it is troublesome to understand whether a new dataset about flowers is going to be closer to the dogs or the planes one. This situation is shown in Figure 5.1.
A good starting point could be about what "similar" means with the available tools we have. Clearly, computing a similarity measures between image datasets requires to choose a representation of the data we want to compare. Different choices can be made and we will see that In the very first steps of our algorithms, we commonly used convolutional neural networks as explained in section 2.5. Another common tool used in the image analysis field is the transformer, as shown in section 2.6. In this sense, the output coming from such models is usually related to common shapes and patterns detected in the images. We are going to see that the features, extracted in such a way are the starting point to further analysis and similarity measures computations.
To partly address these challenges we propose two methodologies base on two different approaches. The first one is going to exploit histograms to represent the datasets features distributions and compute distance between them. The second one is going to rely on an approximation of gaussian kernel, named Random Fourier Features, to compute similarity between data points.
This chapter is organized as follows: in section 5.2 we introduce a set of theoretical concepts about distances. In section 5.3 presents an account of the relevant background while section 5.4 reports details on our methodology. Additionally, we provide the hyperparameters setup and furnish information about the datasets utilized in the experiments. In section 5.5, we present the findings of our empirical investigation. Finally, section 5.6 is left to concluding remarks.



Figure 5.1: Three samples from different datasets. Intuitively it is easy to understand that they are all different one from another. More difficult is to quantify such discrepancy.

5.1 Motivations

The issue of computing dataset distance involves two distinct topics. First, from a theoretical point of view, it can be interesting to quantify similarity in a principled way. Being able to describe different kinds of problems can be compelling and give us a formalization about how they relate one to another. The second important aspect has a more applied flavour and is about transferability processes. Indeed, the questions that we are posing can be relevant both for transfer learning and meta-learning processes, with an impact on the efficiency. For instance, in transfer learning, the aim is to reduce the number of resources used when we are moving to a new context. As we have seen before, one of the common requirements of such models is the availability of a huge amount of data. In some contexts, this could be an easy problem to solve. For some well-known scenarios, it is possible to recover online good datasets, both general-purpose and domain-specific. We have also seen in chapter 3 that a model pre-trained on a good general-purpose dataset such as ImageNet or ImageNet21k can provide valuable results even in more specific contexts. At the same time in chapter 3 we have also seen that these general-purpose datasets provide poor performances when fine-tuned to some precise contexts. For instance when the ratio between the number of samples and the number of classes is low and the inter-class variability is low. This happened with datasets like Stanford Cars and Fine-Grained Visual Categorization Aircraft. The similarity between different classes is high enough to make them hard to be separated. This unpleasant scenario could happen again. In particular, some settings like the medical and biological ones tend to deal with datasets with few samples. In such a context, it could be good to have a good pre-trained model on a similar dataset and not on a general-purpose one. In this sense, by providing a good distance metric between datasets it could be possible to perform a better transfer procedure. Indeed, across similar datasets, it should be easier to transfer knowledge. The intuition behind this is that "transfer" usually means changing the parameters of the model to fit the new one. In a situation where the new setting is more similar to the previous one, this could mean faster fine-tuning as the new weights of the model should be more similar to the old ones. For these reasons, it could be seen as a natural way to perform a transfer operation. Moreover, by defining such a metric we could estimate how tough a transfer process could be.

5.2 Background

In this section, we are going to provide a description of datasets and their links to probability distributions. The purpose of this section is to provide the theoretical framework that we will mostly rely on in this chapter.

Datasets: a bag of datasets is a collection of T, typically labelled, datasets, defined as:

$$S_t = (x_{t,i}, y_{t,i})_{i=1}^{n_t} \qquad t = 1, \dots, T. \tag{5.1}$$

where n_t is the cardinality of each dataset. The inputs x_i^t belongs to some input space $X_t = \mathbb{R}^{d_t}$, where d_t is the input dimension. The labels y_i^t belong to some subset of $Y_t \subset [1, \dots, C]$ among a total of C possible labels. Specifically, we have in mind image datasets, where d_t could be the number of pixels, and the labels identify a class of objects.

Taking a different perspective on our data we can see each dataset as an identical and independent set of samples from a probability distribution P_t on $X_t \times Y_t$. Note that, each dataset defines an empirical distribution:

$$\widehat{P}_t = \frac{1}{n_t} \sum_{i=1}^{n_t} \delta_{(x_{t,i}, y_{t,i})} \tag{5.2}$$

where $\delta_{(x_{t,i}, y_{t,i})}$ is a Dirac delta function centered in $(x_{t,i}, y_{t,i})$. In order to study such distributions, we assume representations are available to embed the inputs in a common space Z. It is worth noticing that this is not true in general. Different datasets can be represented by diverse features with a specific size. In the case of images, for instance, we can have a different number of pixels. Despite this, some techniques are available to force such input to a space with the same dimensionality. More in general, a sequence of transformations can be applied to our data, both for preprocessing (e.g. resize, cropping) and feature extraction. As we have seen in chapter 3, a common technique consists in resizing all images to the same size and using a convolutional neural network to obtain more informative features. Whatever the pipeline, we are going to use Φ to express the mapping operation from the original dataset to the common space Z:

$$\Phi_t : X_t \to Z. \tag{5.3}$$

Here $Z = \mathbb{R}^p$ is the feature space where all inputs are embedded. Note that each map Φ_t defines a probability distribution on Z, via

$$Q_t(A) = P_t(\Phi_t^{-1}(A)), \tag{5.4}$$

for $A \subset F$ measurable. In other words, each Φ_t defines a random variable with law Q_t.

As we were mentioning before, in our image example each map Φ_t arises from a resizing step for each image followed by the application of a common pre-trained deep networks, used as a feedforward feature extractor.

POINTS DISTANCE : in the next pages we are going to discuss the notion of distance between datasets. It is convenient to start by recalling here the definition of *distance* between two points. Usually, the concept of distance is associated with a *metric space*, which is a set equipped with the notion of distance between its elements, usually called *points*. In this sense a distance metric is a function $d : (\cdot, \cdot) \to \mathbb{R}$ satisfying the following properties:

1. Identity: the distance between a point a and itself is zero
$$d(a, a) = 0 \tag{5.5}$$
2. Positivity: the distance between two distinct points a and b is always positive
$$d(a, b) > 0 \tag{5.6}$$
3. Symmetry: the distance between two points a and b is always the same as the distance between b and a
$$d(a, b) = d(b, a) \tag{5.7}$$
4. Triangle inequality: the distance between two points a and c it is always smaller or equal with respect to one obtained by operating a "detour" via a point b
$$d(a, c) \leq d(a, b) + d(b, c) \tag{5.8}$$

The definition of distance applies to many common metrics. The most famous one is probably the *euclidean distance*. Given two points $a, b \in \mathbb{R}^n$ is defined as:

$$d_{eu}(a, b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{5.9}$$

In general, there are many distance measures between points that we can use. We are going to list some of them:

- Manhattan distance: is the sum of the absolute differences of their Cartesian coordinates. The most direct path between two points is the shortest one only along one direction
$$d_{M1}(a, b) = ||a - b||_1 = \sum_{i=1}^{n} |p_i - q_i| \tag{5.10}$$
- Chebyshev distance: the distance between two vectors is the greatest of their differences along any coordinate
$$d_{Ch}(a, b) = \max_i(|a_i - b_i|) \tag{5.11}$$
An intuition about Chebyshev distance is that, on a two-dimensional grid, represents a measure as if the shortest path between two points can take steps in any of the eight grid directions at equal cost.

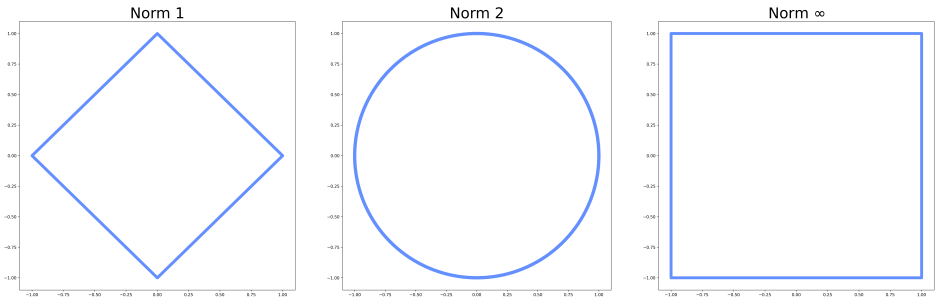


Figure 5.2: The mathematical norms set equal to the unit sphere. (Left) Norm-1 corresponds to Manhattan distance (Center) Norm-2 corresponds to euclidean distance (Right) norm-∞ corresponding to Chebyshev distance.

- Minkowski distance[192]: this distance can be considered a generalization of the euclidean distance and the Manhattan distance and is defined as

$$d_{Mink}(a,b) = (\sum_{i=1}^{n} |a_i - b_i|^p)^{\frac{1}{p}} \tag{5.12}$$

  We notice that we can also have $p \in]0,1]$. This category of distances is usually indicated as *fractional quasinorm*. They are a type of *semimetrics* because they are not proper norms violating the triangle inequality[138].

Minkowski distance measure is a generalization of Manhattan, euclidean, and Chebyshev distance. In this sense, there is a correspondence between the aforementioned distances and some basic geometrical concepts, at least in the $\mathbb{R}^2$ space. For instance, remembering that the summation is only a finite approximation of an integral, we can notice that the euclidean distance corresponds to the mathematical norm-2. In a similar way, Manhattan distance and Chebyshev distance correspond to the mathematical norm-1 and norm-∞, respectively. Moreover, we can easily represent such norms by setting them equal to the unit sphere. This is shown in Figure 5.2.

A different similarity measure between points is *cosine distance*. It is defined as the cosine of the angle between the points viewed as vectors in an inner product space. It is the dot product of the vectors divided by the product of their lengths

$$d_{cos}(a,b) = \frac{a \cdot b}{||a||\,||b||} = = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{5.13}$$

This similarity always belongs to the interval $[-1,1]$ and solely depends on the angle between the two vectors. Intuitively, when two vectors are proportional, orthogonal, or opposite the cosine similarity provides as output 1, 0 or -1, respectively.
In general, it is not easy to determine which distance measure can be more suitable for a specific

task. Certainly, there is not a "best" metric in general but in the deep learning context, we commonly have to deal with features belonging to high-dimensional spaces. A work by Domingos[56] underlines that when working with such spaces our intuitions do not apply to the tri-dimensional space we usually deal with. For instance, in high dimensions, most of the mass of a multivariate gaussian is not close to the mean but in an increasingly distant shell around it. This peculiar, high-dimensional situation can lead to unusual behaviors by pattern algorithms based on proximity like K-Nearest neighbors. It can be meaningful to reason about which metrics can perform better under such conditions. In a work by Hinnenburg et al[97] and in the aforementioned, by Domingos[56] it is shown that Minkowski distances with $p = [1, 2]$ can provide a reliable distance measure for classification algorithms such K-Nearest Neighbors. Another work by Aggarawal et al[5] showed that in Minkowski distances the higher p, the poorer contrast between closest and further neighbors. In this sense, metrics such as Manhattan and fractional quasinorms probably behave better than higher norm metrics like euclidean distance.

Statistical distances : in the previous section we have seen different techniques to compute distances between points in a metric space. Usually, in statistics, we are even more interested in the distance between different data distributions. This perspective can be especially useful in estimating similarity measures, as it enables the comparison of different populations and provides valuable insights into their characteristics. We are going to refer to these distances as *statistical distances*.
Given two distributions we could expect that in general, a measure between the two distributions will respect all the properties shown in Equation 5.5, Equation 5.6, Equation 5.7 and Equation 5.8. This is not true as usually we deal with different situations where a statistical distance violates the non-negativity property shown in Equation 5.6. We are going to refer to such statistical distance as *pseudometrics*. Instead, when the symmetry property in Equation 5.7 is violated we are dealing with *quasimetrics*. Another scenario is the one where the triangle inequality in Equation 5.8 is violated corresponding to the aforementioned scenario of semimetrics. One last common situation is the one where a statistical distance only satisfies the identity and the non-negativity property. Such statistical distances are frequent in statistics and we are going to refer to them as *statistical divergences*. In general, all distance metrics between probability distributions are also divergences, but the opposite is not true. A divergence may or may not be a distance metric.

In general, we can have different ways to categorize statistical distances between distributions. In the following pages, we are going to consider a partitioning between *integral probability metrics* and *f-divergences*. Given two distributions P and Q defined on the same space $\mathcal{X}$ and a class of functions $\mathcal{F}$, an integral probability metric(IPM) is defined as

$$D_{IPM}(P, Q) = \sup_{f \in \mathcal{F}} | \mathop{\mathbb{E}}_{x \sim P} f(x) - \mathop{\mathbb{E}}_{x' \sim Q} f(x')| \tag{5.14}$$

that intuitively is considering the worst case that maximizes the distance by using the function $f \in \mathcal{F}$. The integral probability metrics class includes some popular measures such as maximum mean discrepancy, Wasserstein distance, and energy distance. We are going to provide more details about them in the following pages.

The second class of statistical distance we are going to consider is the f-divergences. Given two distributions P and Q the f-divergence of P from Q is defined as

$$D_f(P,Q) = \int Q(x) f\left(\frac{P(x)}{Q(x)}\right) dx \tag{5.15}$$

Also, the f-divergences class includes a plethora of important statistical distances such as Kullback-Leibler, Jenson-Shannon, and Pearson χ^2. We can start by analyzing the class of integral probability metrics

- Integral Probability Metrics (IPM):
  - Wasserstein distance: introduced by Kantorovich[121] and later reformulated by Vaserstein[296], is a distance function between probability distributions on a given metric space.
    In its most general version, the Wasserstein distance cannot be seen as a special case of integral probability metrics. Indeed, it is defined as

$$W_e(P,Q) = \left(\inf_{\mu \in \mathcal{L}(P,Q)} \int \rho(x,y) d\mu(x,y)\right)^{1/e} \tag{5.16}$$

    where $\mathcal{L}$ is the set of all measures on the metric space. However, we saw in Equation 5.14 that, depending on the class of function $\mathcal{F}$ we select, different distance measures are provided.
    By choosing $\mathcal{F} = \{f : ||f||_L \leq 1\}$ we obtain the Kantorovich metric. From the Kantorovich-Rubinstein theorem, we have that the Kantorovich metric is the dual representation of the Wasserstein distance, defined as:

$$W_1(P,Q) = \inf_{\mu \in \mathcal{L}(P,Q)} \int \rho(x,y) d\mu(x,y) \tag{5.17}$$

    Intuitively, when each distribution is viewed as a unit amount of earth piled in metric space, Wasserstein distance provides the minimum cost of turning one pile into the other, corresponding to the amount of earth that needs to be moved, multiplied by the mean distance between the source and the destination pile. Because of this analogy, the metric is known in computer science as the *earth mover's distance*.
  - Maximum Mean discrepancy: formalized by Smola et al.[265] consists in computing distances between distributions as the distances between mean embeddings of features produced via a feature map $\phi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a reproducing kernel Hilbert space[9].
    The formulation of maximum mean discrepancy is:

$$MMD(P,Q) = ||\mathbb{E}_{X \sim P}[\phi(x)] - \mathbb{E}_{X' \sim Q}[\phi(X')]||_{\mathcal{H}} \tag{5.18}$$

    Its relation with the more general definition of Integral probability metrics is related to the family of the function $\mathcal{F}$ we choose to consider. Indeed, by selecting $\mathcal{F} = \{f :$

Distance	$\mathcal{F}$
Wasserstein 1	$\{f : \|\|f\|\|_L \leq 1\}$
MMD	$\{f : \|\|f\|\|_{\mathcal{H}} \leq 1\}$

Table 5.1: Integral probability metrics and their corresponding class of functions $\mathcal{F}$

$||f||_{\mathcal{H}} \leq 1\}$ we obtain the maximum mean discrepancy formulation.
A simple example can be shown by considering $\mathcal{X} = \mathcal{H} = \mathbb{R}^d$ and $\phi(x) = x$. In this case:

$$\begin{aligned} MMD(P,Q) = ||\mathbb{E}_{X\sim P}[\phi(x)] - \mathbb{E}_{X'\sim Q}[\phi(X')]||_{\mathcal{H}} = \\ = ||\mathbb{E}_{X\sim P}(X) - \mathbb{E}_{X'\sim Q}[X']||_{\mathbb{R}^d} = \\ = ||\mu_P - \mu_Q||_{\mathbb{R}^d} \end{aligned} \tag{5.19}$$

showing that in this simple scenario, the distance between the distribution simply corresponds to the distance between the mean of the distributions. Clearly, with such a simple approximation, two distributions having the same mean will result in having a distance equal to zero, even if they differ in terms of variance and the subsequent distribution momentums. A possible alternative consists in applying the kernel trick, showing that kernels such as the gaussian one, provide a maximum mean discrepancy equal to zero if and only if the two distributions P and Q are identical.

We can report in Table 5.1 a brief recap about the IPM we analyzed and their corresponding family of the function $\mathcal{F}$ defining them as in Equation 5.14.

- F-Divergences:
  - Bhattacharyya distance: it is not a metric, even if it is generally referred to as a "distance". It is instead a semimetric, since it does not obey the triangle inequality. This distance is strictly related to the Bhattacharyya coefficient, measuring the overlap between two distributions in the same space. It is defined as

$$D_{Bhat}(P,Q) = -ln(B_C(P,Q)) \tag{5.20}$$

where $B_C(P,Q)$ is the Bhattacharyya coefficient defined as

$$B_C(P,Q) = \int_{x\in\mathcal{X}} \sqrt{P(x)Q(x)}dx \tag{5.21}$$

Intuitively when the two distributions coincide the distance between them is zero. Indeed we have that

$$B_C(P,Q) = \int_{x\in\mathcal{X}} \sqrt{P(x)P(x)}dx = \int_{x\in\mathcal{X}} \sqrt{P(x)^2}dx = \int_{x\in\mathcal{X}} P(x)dx = 1$$

resulting in $D_{Bhat}(P,P) = 0$. Instead, when the two distributions do not overlap in $\mathcal{X}$ we have that

$$B_C(P,Q) = \int_{x\in\mathcal{X}} \sqrt{P(x)Q(x)}dx = 0$$

resulting in $D_{Bhat}(P,Q) = +\infty$.
It is easy to see that Bhattacharyya distance is an f-divergence measure where the f function involved is $f\left(\frac{P(x)}{Q(x)}\right) = \sqrt{\frac{P(x)}{Q(x)}}$ as

$$D_{Bhat}(P,Q) = -ln\left(\int_{x\in\mathcal{X}} \sqrt{P(x)Q(x)}dx\right) = -ln\left(\int_{x\in\mathcal{X}} Q(x)\frac{\sqrt{P(x)Q(x)}}{Q(x)}dx\right) =$$

$$= -ln\left(\int_{x\in\mathcal{X}} Q(x)\sqrt{\frac{P(x)}{Q(x)}}dx\right) = -ln\left(\int_{x\in\mathcal{X}} Q(x)f\left(\frac{P(x)}{Q(x)}\right)\right) = -ln(D_F(P,Q))$$

- Hellinger distance: defined in terms of the Hellinger integral, introduced by Hellinger[93]. It is a proper statistical distance measure, respecting the aforementioned four properties. It is defined as

$$D_H(P,Q)^2 = \frac{1}{2}\int_{x\in\mathcal{X}}\left(\sqrt{P(x)} - \sqrt{Q(x)}\right)^2 \tag{5.22}$$

From the Cauchy-Schwarz inequality, we can derive that $0 < D_H(P,Q) < 1$. Moreover, it is easy to see that Hellinger distance is an f-divergence measure where the f function involved is $f\left(\frac{P(x)}{Q(x)}\right) = \left(\sqrt{\frac{P(x)}{Q(x)}} - 1\right)^2$ as

$$D_H(P,Q)^2 = \frac{1}{2}\int_{x\in\mathcal{X}}\left(\sqrt{P(x)} - \sqrt{Q(x)}\right)^2 =$$

$$= \frac{1}{2}\int_{x\in\mathcal{X}}\left(P(x) + Q(x) - 2\sqrt{P(x)Q(x)}\right) =$$

$$= \frac{1}{2}\int_{x\in\mathcal{X}} Q(x)\left(\frac{P(x)}{Q(x)} + 1 - 2\sqrt{\frac{P(x)}{Q(x)}}\right) = \frac{1}{2}\int_{x\in\mathcal{X}} Q(x)\left(\sqrt{\frac{P(x)}{Q(x)}} - 1\right)^2 =$$

$$= \frac{1}{2}D_F(P,Q)$$

Moreover, in its discrete formulation, it is easy to see that the Hellinger distance is strongly tied to the Bhattacharyya coefficient, as:

$$D_H(P,Q)^2 = 1 - B_C(P,Q) \tag{5.23}$$

To prove it we can notice that:

$$D_H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_i^n \left(\sqrt{p_i} - \sqrt{q_i}\right)^2}$$

By elevating to the square both sides and by remembering that for a general discrete distribution D it holds that $\sum_{i=1}^{n} d_i = 1$:

$$D_H(P,Q)^2 = \frac{1}{2}\sum_i^n (\sqrt{p_i} - \sqrt{q_i})^2 = \frac{1}{2}\sum_i^n (p_i + q_i - 2\sqrt{p_i q_i}) =$$

$$= \frac{1}{2}\sum_{i=1}^{n} p_i + \frac{1}{2}\sum_{i=1}^{n} q_i - \sum_{i=1}^{n}\sqrt{p_i q_i} = 1 - \sum_{i=1}^{n}\sqrt{p_i q_i} = 1 - B_C(P,Q)$$

- Kullback–Leibler divergence: introduced by Kullback and Leibler[145] is a statistical divergence measure. Indeed it does not satisfy the triangle inequality nor the symmetry property. It is defined as

$$D_{KL}(P||Q) = \int P(x)\ \log\left(\frac{P(x)}{Q(x)}\right) dx \tag{5.24}$$

It is easy to show, via Gibbs inequality, that $D_{KL}(P||Q) = 0 \Leftrightarrow P = Q$.
It can also be shown that $D_{KL}(P||Q) \geq 0$
It is also easy to show that Kullbach-Leibler divergence is an f-divergence measure where the f function involved is $f\left(\frac{P(x)}{Q(x)}\right) = \frac{P(x)}{Q(x)} log\left(\frac{P(x)}{Q(x)}\right)$ as

$$D_{KL}(P||Q) = \int_{x\in\mathcal{X}} P(x)\, log\left(\frac{P(x)}{Q(x)}\right) = \int_{x\in\mathcal{X}} Q(x)\, \frac{P(x)}{Q(x)}\, log\left(\frac{P(x)}{Q(x)}\right) =$$
$$= D_F(P,Q)$$

The Kullbach-Leibler divergence also has some interesting properties. The quantity $D_{KL}(P||Q)$ is convex in the pair of probability measures P, Q, i.e. by having two pairs of probability distributions (P_1, Q_1), (P_2, Q_2) and $\lambda \in [0,1]$, then:
$D_{KL}(\lambda P_1 + (1-\lambda)P_2||\lambda Q_1 + (1-\lambda)Q_2) \leq \lambda D_{KL}(P_1||Q_1) + (1-\lambda)D_{KL}(P_2||Q_2)$
Moreover, the divergence measure is additive for independent distributions, i.e. given two pairs of independent distributions P_1, P_2, Q_1, Q_2 such that: $P(x,y) = P_1(x)P_2(y)$ and $Q(x,y) = Q_1(x)Q_2(y)$ then:
$D_{KL}(P||Q) = D_{KL}(P_1||Q_1) + D_{KL}(P_2||Q_2)$

So far we have been thinking of D_{KL} as a way to compute the distance between two distinct distributions. A slightly different point of view consists in trying to replace a function we have with an estimate of it. The idea is to replace the original function with another one that is as most as similar as possible to the original one but simpler. A new function can be proposed and then we can compute the distance from the original one by using the D_{KL}. We can even extend this reasoning by proposing a

parametric function, replacing the original, and minimizing the D_{KL} over its parameters. In this sense, there is a strong tie between this approach and neural networks. Indeed, such models are function approximators that, minimizing a good objective function, can com learn a wide range of complex functions. We have just seen that we can use and minimize the D_{KL} to obtain a good estimate of the original function. This is exactly what is done when using Variational Autoencoders, which learn to map an input (e.g. an image) into a gaussian distribution in a latent space and compute exactly the D_{KL} between such gaussian and the standard gaussian:
$D_{KL}(\mathcal{N}(\mu, \sigma)||\mathcal{N}(0, I))$
In the Variational Autoencoder context, the intuition is to use the D_{KL} value as a regularization term in the objective function model.

- Jensen–Shannon divergence: introduced by Jensen and Shannon[165] it is based on the Kullbach-Leibler divergence, being a symmetrized and smoothed version of it. Therefore it is a proper statistical distance defined as

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}\left(P||\frac{P+Q}{2}\right) + \frac{1}{2}D_{KL}\left(Q||\frac{P+Q}{2}\right) \tag{5.25}$$

the Jensen–Shannon divergence is everywhere defined and bounded and it is equal to zero only when P and Q coincides. It is possible to show that Jensen–Shannon is an f-divergence measure where the f function involved is
$f\left(\frac{P(x)}{Q(x)}\right) = \frac{1}{2}\left(\frac{P(x)}{Q(x)}log\left(\frac{2P(x)}{P(x)+Q(x)}\right) + log\left(\frac{2Q(x)}{P(x)+Q(x)}\right)\right)$ as

$$D_{JS}(P,Q) = \frac{1}{2}D_{KL}\left(P||\frac{P+Q}{2}\right) + \frac{1}{2}D_{KL}\left(Q||\frac{P+Q}{2}\right) =$$
$$= \frac{1}{2}\int P(x)log\left(\frac{2P(x)}{P(x)+Q(x)}\right)dx + \frac{1}{2}\int Q(x)log\left(\frac{2Q(x)}{P(x)+Q(x)}\right)dx =$$
$$= \frac{1}{2}\int P(x)log\left(\frac{2P}{P(x)+Q(x)}\right) + Q(x)log\left(\frac{2Q(x)}{P(x)+Q(x)}\right)dx =$$
$$= \int Q(x)\frac{1}{2}\left(\frac{P(x)}{Q(x)}log\left(\frac{2P(x)}{P(x)+Q(x)}\right) + log\left(\frac{2Q(x)}{P(x)+Q(x)}\right)\right)dx =$$
$$= D_F(P,Q)$$

– Pearson's χ-square divergence: derived directly as an interpretation of the Pearson's statistical χ^2 test that was introduced by Pearson [219] it is defined as

$$D_\chi(P,Q) = \int \frac{(P(x)-Q(x))^2}{Q(x)}dx \tag{5.26}$$

the interpretation given by Pearson is related to statistical tests. Indeed the χ^2 test can be used to perform three different types of comparison. One is a goodness of fit to establish whether an observed frequency distribution differs from a theoretical

Distance/Divergence	$f(t)$
Bhattacharyya distance	$\sqrt{t}$
Hellinger distance	$(\sqrt{t}-1)^2$
Kullbach-Leibler divergence	$t\,log(t)$
Jensen-Shannon divergence	$\frac{1}{2}\left(t\,\frac{2t}{t+1}+log\frac{2}{t+1}\right)$
Pearson's χ-square divergence	$(t-1)^2$

Table 5.2: f-divergences and their corresponding $f(t)$, with $t=\left(\frac{P(x)}{Q(x)}\right)$

distribution. Another one is the independence test to assess whether observations consisting of measures on two variables, expressed in a contingency table, are independent of each other. Pearson's χ-square quantity defined above can be also used as a divergence measure, similar to other divergences we have seen previously. It is also easy to see that is an f-divergence measure where the f function involved is

$$f\left(\frac{P(x)}{Q(x)}\right)=\left(\frac{P(x)}{Q(x)}-1\right)^2 \text{ as}$$

$$D_\chi(P,Q)=\int\frac{(P(x)-Q(x))^2}{Q(x)}dx=\int Q(x)\frac{(P(x)-Q(x))^2}{Q(x)^2}dx=$$

$$=\int Q(x)\frac{P(x)^2+Q(x)^2-2P(x)Q(x)}{Q(x)^2}dx=\int Q(x)\left(\frac{P(x)^2}{Q(x)^2}+1-2\frac{P(x)}{Q(x)}\right)dx=$$

$$=\int Q(x)\left(\frac{P(x)}{Q(x)}-1\right)^2dx=D_F(P,Q)$$

We can report in Table 5.2 a brief recap about the f-divergences we analyzed and their corresponding $f\left(\frac{P(x)}{Q(x)}\right)$ as defined in Equation 5.15

We also have statistical distances that do not belong to the above definitions for some reason or that belong to both of them. The last scenario is the one of *total variation*. The total variation distance is based on the concept of total variation introduced by Jordan[32]. Its peculiarity can be described both by the definitions of integral probability metrics and f-divergences. It is a proper statistical distance measure, satisfying the four properties included in the definition of statistical distance. It is defined as

$$D_{TV}=\max_{A\subset\mathcal{X}}|P(A)-Q(A)| \tag{5.27}$$

representing the largest possible difference that two probability distributions can assign to the same event. We can also provide an alternative formulation for the total variation distance:

$$D_{TV} = \frac{1}{2}\sum_{x\in\mathcal{X}} |P(x) - Q(x)| \tag{5.28}$$

We are not going to prove that the two formulations are equivalent but at least we can show that there exists a set A such that these two formulations are equivalent. We can consider the first formulation together with a set $A = \{x \in \mathcal{X} : P(x) \geq Q(x)\}$. Then, by the definition of probability mass and by remembering that $P(x) - Q(x) \geq 0$ when $x \in A$

$$P(A) - Q(A) = \sum_{x\in A} p(x) - q(x) \Leftrightarrow$$
$$\Leftrightarrow |P(A) - Q(A)| = \sum_{x\in A} |p(x) - q(x)|$$

Now we can consider the complement set A^c, and by remembering that $P(A^c) = 1 - P(A)$ and again by the definition of probability mass

$$P(A^c) - Q(A^c) = 1 - P(A) - (1 - Q(A)) = Q(A) - P(A) = \sum_{x\in A^c} p(x) - q(x)$$

Now, by considering that $P(x) - Q(x) < 0$ when $x \in A^c$

$$Q(A) - P(A) = \sum_{x\in A^c} p(x) - q(x) \Leftrightarrow$$
$$\Leftrightarrow -|P(A) - Q(A)| = -\sum_{x\in A} |p(x) - q(x)| \Leftrightarrow$$
$$\Leftrightarrow |P(A) - Q(A)| = \sum_{x\in A} |p(x) - q(x)|$$

Now, to cover the whole space $\mathcal{X}$ it is sufficient to sum up the two equations we got from the sets A and A^c showing that

$$2|P(A) - Q(A)| = \sum_{x\in\mathcal{X}} |P(x) - Q(X)| \Leftrightarrow$$
$$\Leftrightarrow |P(A) - Q(A)| = \frac{1}{2}\sum_{x\in\mathcal{X}} |P(x) - Q(X)|$$

The total variation distance can be related to Kullback-Leibler divergence by Pinsker's inequality:

$$D_{TV}(P, Q) \leq \sqrt{\frac{1}{2} D_{KL}(P||Q)} \tag{5.29}$$

This inequality has been proved in different manners[71, 172, 289] showing an explicit tie between these two statistical measures. Total variation distance and Kullbach-Leibler divergence are also tied together by an inequality proposed by Bretagnolle and Huber[29]

$$D_{TV}(P,Q) \leq \sqrt{1 - e^{-D_{KL}(P||Q)}} \tag{5.30}$$

Moreover, total variation is tied to Hellinger distance by the following inequality[139]:

$$D_H(P,Q)^2 \leq D_{TV}(P,Q) \leq \sqrt{2} D_H(P,Q) \tag{5.31}$$

Random Fourier Features: in the previous section we have seen a different kind of distance measure between distributions and the main differences between two important measure families: Integral Probability Metrics and F-Divergences. In this section we are going to consider one of the distances analyzed before, the Maximum Mean Discrepancy, and how it is related to the Random Fourier Features, used to approximate the inner product of the kernel matrix as defined in Equation 2.29. Previous works showed that it is possible to speed up the training of kernel machines by first mapping the input data to a low-dimensional feature space using randomization, and then utilizing pre-existing rapid linear methods.
If we consider a learning problem with a dataset such as the one we introduced at the very beginning of **??** we usually want to find a hyperplane, separating the data into some categories according to a certain performance measure such as a loss function.
Such a simple approach usually does not work with complicated data as, usually, they are not linearly separable. When we deal with kernel methods we want to map our inputs to a new space $\mathcal{V}$. The dimension of $\mathcal{V}$ is usually high, and kernel methods do not operate explicitly in this space. Indeed, they use the kernel trick. If we define a positive definite kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, by Mercer's theorem[189] there exists a feature map $\phi : \mathcal{X} \mapsto \mathcal{V}$ such that

$$k(x,y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{V}} \tag{5.32}$$

By using the kernel trick and the representer theorem[132] we can set up non-linear models of our data that are linear in $k(\cdot,\cdot)$, as

$$m(x) = \sum_{n=1}^{N} \alpha_n k(x, x_n) = \langle w, \phi(x) \rangle_{\mathcal{V}} \tag{5.33}$$

Even if the kernel trick allows us to treat non-linear data with linear models in high dimension, the main problem with this family of methods is in their computational costs as they need to compute and invert a Gram matrix, resulting in a cubic cost.
In 2007 Rahimi and Recht [233] proposed a methodology to approximate the inner product in Equation 5.32. This way we have that $m(x) = \sum_{n=1}^{N} \alpha_n k(x, x_n) = \sum_{n=1}^{N} \alpha_n \langle \phi(x_n), \phi(x) \rangle_{\mathcal{V}} \approx \sum_{n=1}^{N} \alpha_n z(x_n)^T z(x) = \beta^T z(x)$

resulting in:

$$m(x) = \beta^T z(x) \tag{5.34}$$

meaning that if we have a good approximation z of the feature map ϕ then we can project our data using directly z.

Remembering that the Gram matrix at location (i, j) contains the product between the i-th and j-th vectors, i.e. a similarity measure between such vectors, this procedure can be useful if we want to compute an approximate version of such similarity.
Consider a matrix Ω

$$\Omega = \begin{bmatrix} \Omega_{00} & \cdots & \Omega_{0m} \\ \vdots & \omega_i & \vdots \\ \Omega_{f0} & \cdots & \Omega_{fm} \end{bmatrix}$$

where every row $\omega_i \sim \mathcal{N}(0, \frac{1}{\sigma^2} I)$. If we consider the set of features F extracted via a features map Φ such as a pre-trained convolutional neural network, and compute $X = [cos(F\Omega), sin(F\Omega)]$, it is possible to show that $\langle X, X' \rangle$ corresponds to an approximation of the gaussian kernel. Indeed, by Euler formula:

$$X = [cos(F\Omega), sin(F\Omega)] = e^{i\omega^T X}$$

If we compute the inner product between two set of points:

$$\langle X, X' \rangle = \frac{1}{N} \sum e^{i\omega^T X} e^{i\omega^T X'} = \frac{1}{N} \sum e^{i\omega^T (X - X')}$$

By increasing the size M of the original Ω matrix we can obtain a better approximation of the original gaussian kernel as:

$$\frac{1}{N} \sum e^{i\omega^T (X - X')} \approx \int_{\mathbb{R}^D} e^{i\omega^T (X - X')} d\omega = e^{-\frac{||X - X'||^2}{2}}$$

that is the value computed by a gaussian kernel.

5.3 Related works

In the previous section, we considered different methodologies to compute the distance between distributions. We have seen that two main families exist, Integral Probability Metrics and F-Divergences. The research field of computing distances between datasets has been explored to some extent in an unstructured way by a number of studies, focusing on different but related concepts. Indeed, some works concentrate on computing a distance/divergence measure[3, 10] between datasets. Others focus on computing a transferability measure [74, 202]. In this section, we will discuss in particular two instances of previous works that attempted to compute this quantity. Aside from their good formalization we focus on such works because the apply to of the distributions metric we introduced in the previous section. The first work will focus on a set of techniques, named optimal transport techniques, based on an extension of Integral Probability Metrics, the Wasserstein distance. The second will focus instead on computing a divergence between distributions via the Kullbach-Leibler F-Divergence. We are going to show that the theoretical concept of distance between datasets can be studied from different perspectives and it can be tied to the concept of transferability or to the meta-learning context.
In a work proposed by Alvarez-Melis and Fusi[10] the similarity notion is presented as central in

different domains like meta-learning and domain adaptation. They also point out some potential issues about recent methodologies as being often heuristic, making strong assumptions on the labels set across datasets, and being also architecture dependent. In their work, an algorithm is proposed to compute datasets distance quantity. Such methodology is model agnostic, does not involve training, and works on datasets with completely disjointed labels. The central element of this research is *optimal transport*[221, 298]. In this context, given a complete and separable metric space $\mathcal{X}$ and two probability distributions $\alpha \in \mathcal{P}(\mathcal{X})$ and $\beta \in \mathcal{P}(\mathcal{X})$ the Kantorovich formulation[122] is produced as:

$$OT(\alpha, \beta) \stackrel{\text{def}}{=} \min_{\pi \in \Pi(\alpha,\beta)} \int_{\mathcal{X}\times\mathcal{X}} c(x, z) d\pi(x, z) \tag{5.35}$$

where $c(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ is the ground cost that can coincide with the metric $d_{\mathcal{X}}$ belonging to the space $\mathcal{X}$. Instead, $\Pi(\alpha, \beta)$ is the set of coupling that consists of the joint probability distributions over the product space $\mathcal{X} \times \mathcal{X}$ with marginals α and β. In such a situation we can define the *p-Wasserstein distance* as:

$$W_p(\alpha, \beta) \stackrel{\text{def}}{=} OT(\alpha, \beta)^{1/p} \tag{5.36}$$

Usually the exact distributions α and β are not known in practice so the finite approximations $\hat{\alpha} = \frac{1}{n}\sum_{i=1}^{n} \delta_{x^i}$ and $\hat{\beta} = \frac{1}{m}\sum_{i=1}^{m} \delta_{z^i}$ are computed.
In section 5.2 we considered the dataset as sets of features X and labels Y. In this context the distance between two datasets z, z' can be defined as:

$$d_{\mathcal{Z}}(z, z') = (d_{\mathcal{X}}(x, x') + d_{\mathcal{Y}}(y, y'))^{1/p} \tag{5.37}$$

Even if it is not an easy task, theoretically the distance term about the features $d_{\mathcal{X}}(x, x')$ can be computed in different ways, e.g. euclidean distance in features space. Even from a theoretical point of view, it is not clear how to compute distances between labels, in particular when they are coming from unrelated datasets. When some prior knowledge of the label space is available, e.g. when working with hierarchical datasets, this information can be exploited to define a notion of distance. However, the most common scenario is one where such knowledge is missing. In this scenario, we only have some information about label occurrences in relation to their features. Therefore we can map our labels as:

$$y \mapsto \alpha_y(X) = P(X|Y = y) \tag{5.38}$$

With such representation, computing a distance means choosing a statistical divergence between probability distribution. Optimal transport via Wasserstein distance is a good choice as provides a valid metric, is computable from a finite number of samples, and supports sparse distributions. With this idea in mind, we can redefine the distance between two datasets as

$$d_{\mathcal{Z}}((x, y), (x', y')) \stackrel{\text{def}}{=} (d_{\mathcal{X}}(x, x')^p + W_p^p(\alpha_y), \alpha_{y'})^{1/p} \tag{5.39}$$

This approach is eventually tested in three different scenarios. The first one is dataset selection for transfer learning. The authors show that on some simple datasets like MNIST[155], FASHION-MNIST[307], KMNIST[45], USPS[112] and the `letters` split of EMNIST[47] it is possible to compute a distance metric. The results are used to derive a transferability measure $\mathcal{T}$ within source and target datasets. This work shows a correlation between such transferability measures and the obtained results on the target datasets.

A second interesting work is the on presented by Achille et al.[3]. In their work, the aim is to provide a vectorial representation of image classification datasets. The key idea is the following: given a dataset with ground-truth labels such images are processed through a small network called *probe network*. Then the network's parameters are considered and an embedding based on the Fisher information matrix[67] is produced.
In section 5.2 we considered the dataset as a set of inputs X and labels Y. If we consider an image x as input and its corresponding label y we can see a deep network as a family of functions $p_w(y|x)$ trained to approximate the posterior $p(y|x)$ by minimizing a cross-entropy loss. The weights included in the net can be more or less "important" according to the influence they have on determining the networks' outputs. Such importance can be quantified by considering a slight perturbation of the weights

$$w' = w + \varepsilon w \tag{5.40}$$

and by measuring the Kullback-Leibler divergence, as shown in section 5.2, between the original distribution $p_w(y|x)$ and the new one $p_{w'}(y|x)$ given by the perturbation in Equation 5.40. The second-order approximation of such divergence is:

$$\mathbb{E}[KL(p_{w'}(y|x)||p_w(y|x))] = \varepsilon w \cdot F\varepsilon w + o(\varepsilon w^2) \tag{5.41}$$

where KL is the Kullback-Leibler divergence and F is the Fisher information matrix:

$$F = \mathbb{E}[\nabla_w log(p_w(y|x))\nabla_w log(p_w(y|x))^T] \tag{5.42}$$

showing the covariance of the log-likelihood with respect to the model parameters. The Fisher information matrix contains information about a particular parameter involved in the training process. Intuitively, when a parameter is not important in determining the outcome, the corresponding entry in the matrix will be small. One possible problem with this approach is the low stability related to the highly irregular and noisy loss landscape inherent in many deep learning architectures. To avoid such situations a more robust estimator that leverages connections to variational inference is computed.
The main advantages of this embedding are related to the good properties the Fisher information matrix is bringing: the task embedding does not depend on the label but only on the distribution predicted by the model. Moreover, it has some correlation with the task difficulty and, when a hierarchical dataset is provided, it correlates with the taxonomical distance on the dataset.

5.4 Methodology

In the previous sections we have seen that different metrics can be computed to estimate the distance between distributions. Given the above discussion, we first describe the two approaches we considered to compute datasets' distance. We will see that the overall procedure involves three subsequent steps:

- Representing the input images via resize and features extraction, $x_{t,i}, \mapsto z_{t,i}$
- Apply one between the two specific approaches, i.e. histograms or Random Fourier Features to represent each dataset
- Compute a distance measure between every dataset

Then, we present technical details about the conducted experiments. We consider the hyperparameters tuning.

Pipelines : in our work we considered two different methodologies to compute dataset distance between different image datasets:

1. *Histograms*: after preprocessing an image dataset and map our input images to the same features subspace z via a pre-trained convolutional neural network. This way, a one-dimensional histogram can be computed for each of the T embedded datasets and for each coordinate s:
$$z^s_{t,1}, \dots, z^s_{t,n} \mapsto \widehat{H}^s_t \in \mathbb{R}^b \tag{5.43}$$
where b is the fixed number of bins. Once a histogram is produced for every feature column s, we can compare such histograms to the ones of another dataset, by computing a distance measure between every histogram couple.
2. *Random Fourier Features*: in the second set of experiments, we first perform the same steps of the histogram analysis by preprocessing the images. Then we consider a pre-trained transformer, to map our input images to the same features subspace z. Lastly, remembering that each kernel matrix location represents a similarity measure between the two corresponding entries of the matrix, we can approximate the inner product of such matrix via the procedure explained in section 5.2

Experiments details: we have seen in the previous section that different approaches can be adopted to compute distance measures between distributions. Each procedure can include a set of hyper-parameters that must be set a priori. Our analysis does not prioritize exhaustive exploration of hyper-parameters. Rather, we examine a selection of feasible configurations. A common hyperparameter to both pipelines is the image size. In order to represent all the inputs in the same way, i.e. to have the same dimensionality, we need to resize our images. Such a procedure, which is simple by itself, is indeed an important step as we have to set a fixed size. To perform the distance computation in a reasonable time, we decided to resize all the images to 128×128 pixels.

Some datasets have a dimension that is already comparable to the size we choose. However, some datasets like CIFAR10/100 are way smaller (32×32) while other datasets such as DTD have bigger images on average(453×500). The aforementioned operation can be performed also via cropping. The downside of this technique is the wasted image information given by the black-padded image portion after the crop. Nonetheless, the cropping operation preserves the image content proportion. This property is not true, in general, when performing a resize operation.
Another important hyperparameter involved in the process is the pre-trained model. In the most simple and intuitive procedure, extracting features via a pre-trained model is not necessary as we could compare directly images from different datasets. Actually, comparing images is complicated because of their size and also because most of them can contain wide uniform regions carrying a small information amount. On the other hand, selecting a pre-trained model can help in providing informative features as output.
Deciding which pre-trained model to use will determine output features depending both on the architecture and on the pre-train itself. In the histogram anlysis we applied a DenseNet201 pre-trained on ImageNet. Instead, in the second approach, we used a Vision Transformer (ViT-L/16) as presented in [60]. We changed the pre-trained model involved hoping to produce a more informative output, due both to the architecture and to the wider pre-trained, as Vision Transformers are usually pre-trained on ImageNet-21k, a larger version of the traditional ImageNet. This idea is also confirmed by previous results such as the one shown in Table 3.5. On the other hand, extracting features via a Vision Transformer is more expensive in terms of inference time, as shown in Table 3.4.

We now focus on hyperparameters specific to the adopted methodology.

- Histograms: an important aspect of the histogram analysis is the number of hyperparameters involved in the process. In the next lines we will provide a full list of the hyperparameters involved in the process:
  - Bins number b: in order to obtain a proper histogram, we need to partition the feature space into a fixed amount of bins. Having the same number of bins makes different datasets comparable. In our experiments, we used 20 bins.
  - Percentile p: even if some pre-trained models force their output in a certain interval value, this is not true in general. This can be a problem, as considering the maximum of all the feature values in certain features will result in a very sparse histogram. To address this problem we decided to force all the values above the 90-th percentile into the last bin. This is visible in Figure 5.4, where the last bin is usually higher than previous bins, as it contains all occurrences not belonging to the 90-th percentile.
- Random Fourier Features:
  - Standard deviation σ: the first thing we consider in the random Fourier features procedure is the randomly generated matrix Ω. Such a matrix is generated according to some standard deviation σ, influencing how each value deviates from the average of the distribution. Therefore, different values of σ can result in very different outputs. In our experiments we focused on $\sigma \in [10^{-5}, 10^{-1}]$

Super Category	Classes	Images	Duration(s)	images/sec
Actinopterygii	53	2619	79.26	33.04
Amphibia	115	17703	305.83	57.88
Animalia	77	6590	137.12	48.06
Arachnida	56	5959	119.08	50.04
Aves	964	235521	3268.36	72.06
Chromista	9	542	61.75	8.78
Fungi	121	7606	149.74	50.80
Insecta	1021	118555	1769.64	66.99
Mammalia	186	32823	533.03	61.58
Mollusca	93	9377	170.91	54.87
Plantae[1]	2101	196613	2780.59	70.71
Protozoa	4	381	59.36	6.42
Reptilia	289	40881	657.78	62.15

Table 5.3: Information on the iNaturalist2017 dataset. We report the number of classes and images. We also report time information about the features extraction via a Vision Transformer(ViT-L/16).

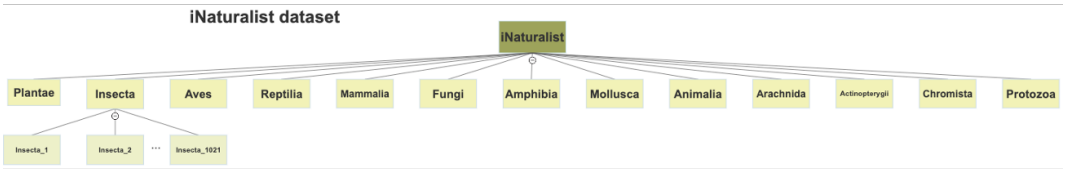


Figure 5.3: The internal hierarchy of the iNaturalist 2017 dataset. Every class belongs to a super category.

- Number of feature vectors M: another important parameter related to the randomly generated matrix Ω is the amount of column vector involved, determined a priori. We showed that by increasing the value of M we get a better approximation of the gaussian kernel, associated with an increased computational cost. In our experiments we focused on $M \in [5, 50]$

iNaturalist dataset: to partly address the problems we had on the datasets shown in previous pages we decided to consider a new peculiar dataset called *iNaturalist2017* [292]. It has been introduced as part of the *iNat Challenge 2017 large scale species classification* competition. The natural world contains at least several million species of plants and animals. Without specific knowledge, many species are extremely difficult to classify due to their visual similarity. The iNaturalist dataset contains approximately 5000 species grouped into thirteen super categories, with a combined training and validation set of approximately half-million images collected and verified. We report some useful details about the dataset in Table 5.3.

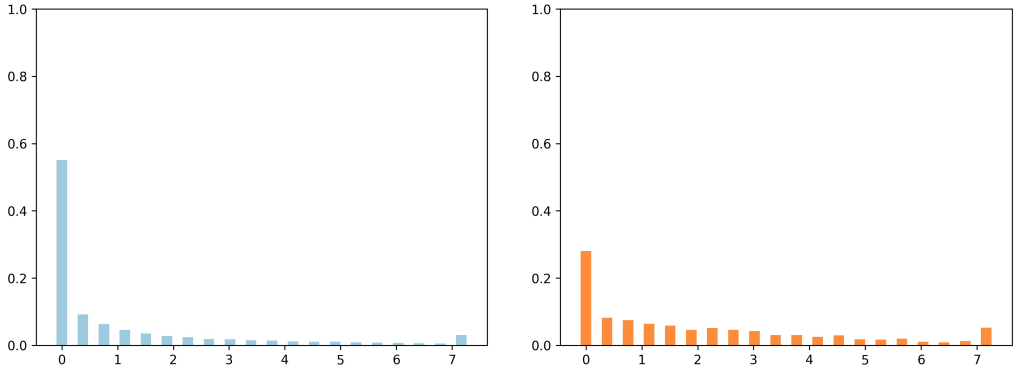


Figure 5.4: Two histograms produced by partitioning the feature number 18432 in 20 bins after extracting them via a DenseNet201 pre-trained on ImageNet

We decided to focus on the iNaturalist dataset as it provides an internal hierarchy that we can exploit to compare our metrics to. By following a procedure similar to the one explained in the previous pages, we extracted the image features with a Vision Transformer. We also report as a reference, the time needed to extract image features for every class via the Vision Transformer architecture. The next step was to compute a centroid for every class via the random Fourier features procedure described in the previous pages. The hierarchical structure of iNaturalist is shown in Figure 5.3.

5.5 Experiments

In the previous sections, we saw that different procedures can be applied to compute a distance measure between distributions. In the following pages, we are going to show a set of empirical analyses we performed on a bag of datasets from small to medium size.

5.5.1 Histogram analysis

A common practice to study different distributions consists in estimating their distribution via partitioning of the input space. Such an approach, as part of the non-parametric techniques, does not rely on the labels nor on one a-priori choice of the functions involved to map our data.
In this first set of experiments, we preprocessed a bag of image datasets by standardizing and normalizing their values and resizing them to the same size. This way we could consider a pre-trained convolutional neural network, to map our input images to the same features subspace z. In this sense: $x_{t,i}, \mapsto z_{t,i} = \Phi_t(x_{t,i})$.
This way, a one-dimensional histogram can be computed for each of the T embedded datasets, and for each coordinate s we can compute a histogram like the one defined by Equation 5.43. Note that, each histogram $\widehat{H}_t^s$ can be seen as an empirical approximation of the corresponding marginal Q_t^s.
We can show in Figure 5.4 two instances of histogram produced as output from this procedure, from the same feature, with different datasets.

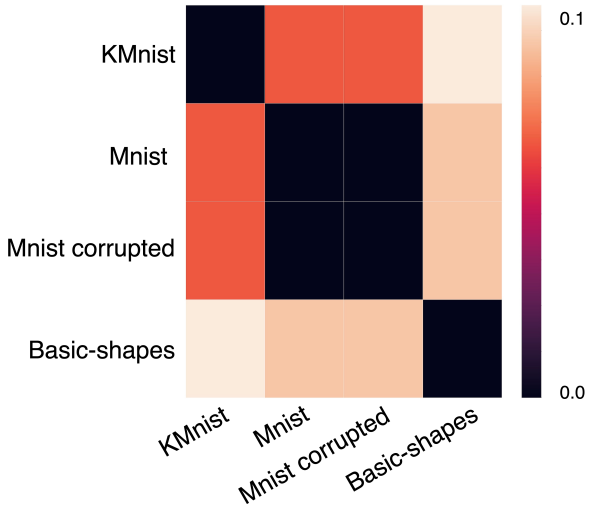


Figure 5.5: Normalized heatmap produced by comparing four different datasets: three about handwritten digits, one about geometric shapes. The distance between the handwritten digits datasets is smaller w.r.t. the one with the basic shapes

Following the above reasoning we obtain, for every dataset, a number of histograms equal to the number of features. Then, an approximate distance metric between two datasets can be defined as the summation of the distance between the same histograms coming from the two datasets:

$$d(Q_i, Q_j) \approx \sum_{s=1}^{p} d(\widehat{H}_i^s, \widehat{H}_j^s) \tag{5.44}$$

We can show the distance between four different datasets in Figure 5.5. Three datasets are about handwritten digits, one is instead about geometric shapes. We can notice that the distance between the three handwritten digit datasets is smaller w.r.t. the one with the basic shapes.

After we obtained good results on such a simple scenario, we tested on similar contexts with a reduced amount of datasets, obtaining similar results.
We decided to test it also on a more complicated setting. We included 35 different datasets with a huge overlap with the ones studied in chapter 3. All the details about such a group of datasets can be found in Table 3.1. The results for such analysis are shown in Figure 5.6. We can notice that some clusters are visible. For instance, cifar10 and cifar100 are close to one another. Also, the distance between flower datasets is small. This is also true for animal datasets such as Oxford-III pets and Stanford dogs and cats vs dogs. At the same time, the distance between texture datasets like euroSAT and DTD is small.

Overall with this first approach, we obtained mixed results as it produces coherent results when dealing with simple contexts with few datasets such as the one shown in Figure 5.5. Moreover, even on more complicated situations such as the one shown in Figure 5.6 similar datasets are closer to one another w.r.t. visually different datasets. On the other hand not having a ground truth to compare our results to is problematic and not all the results included in the analysis are coherent.

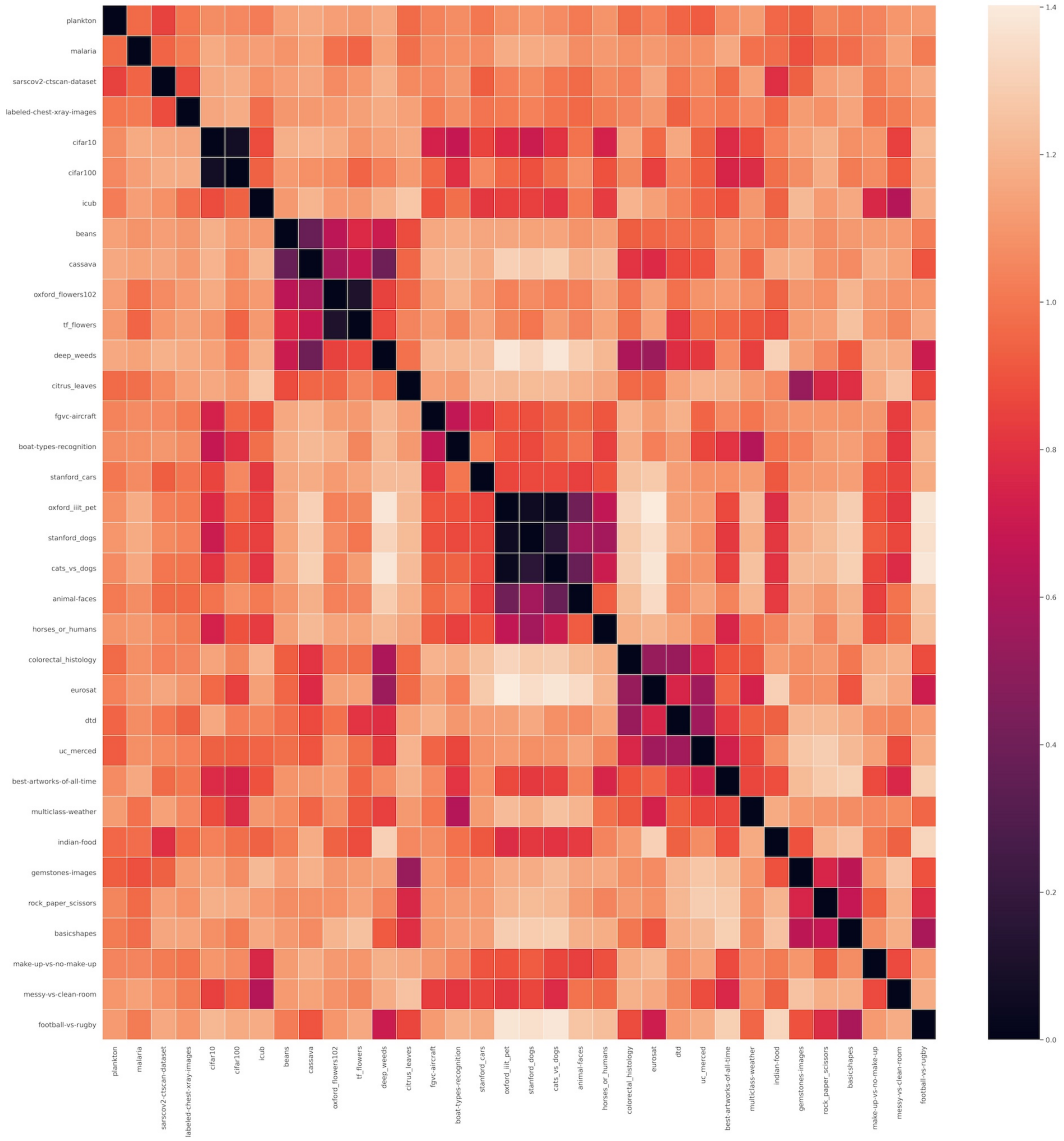


Figure 5.6: Normalized heatmap produced by comparing thirty-five different datasets.

Moreover, the histogram approach requires a wide number of hyperparameters we need to fix a-priori, making this approach susceptible to our choices.

5.5.2 Random Fourier Features analysis

By considering the theoretical framework introduced in the previous sections we performed an empirical analysis of dataset distance via the random Fourier features technique. In practice, the first steps of the procedure, like we already saw for the histograms, consisted in using a pre-trained model to extract informative features from images. Then we multiplied, for every dataset, such

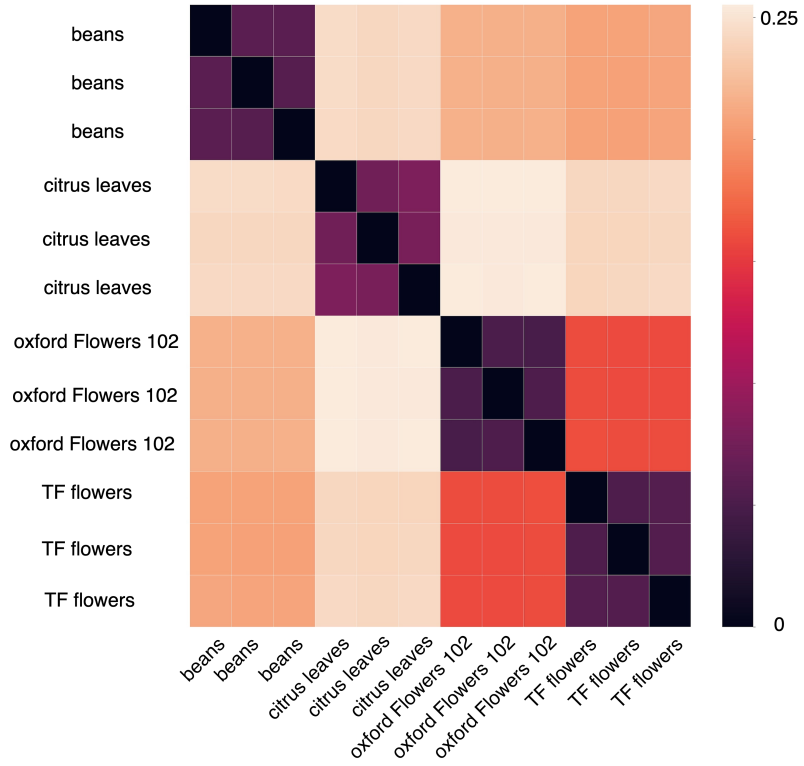


Figure 5.7: Preliminary test with random fourier features. Four datasets are considered and split into three subset stratified on the label. Subgroup belonging to the same dataset are closer to each other w.r.t. other datasets.

features for the random matrix Ω. After duplicating the result and applying sine transformation to the first half and cosine transformation to the second half of the output, we computed a column-wise average.

In order to provide a preliminary test with the random Fourier features approach, we decided to consider four different datasets. We split each dataset into three subgroups, stratified on the class label. Then, we computed the distance between each subgroup to very whether a dataset is, in fact, closer to itself w.r.t. other datasets. The results of these preliminary experiments are shown in Figure 5.7.

We can notice that subgroups belonging to the same dataset are closer to each other w.r.t. other datasets. We can also notice that, aside from subgroups belonging to the same dataset, the two closest datasets to one another are Oxford Flowers 102 and TensorFlow Flowers, which are visually and conceptually similar.
The second test we performed was about including more datasets and checking their distance according to the random Fourier features methodology. Even with six datasets, only the results were worse than expected. However we decided to print a principal component analysis to two dimensions of the data, resulting in the plot we show in Figure 5.8, where every point represents

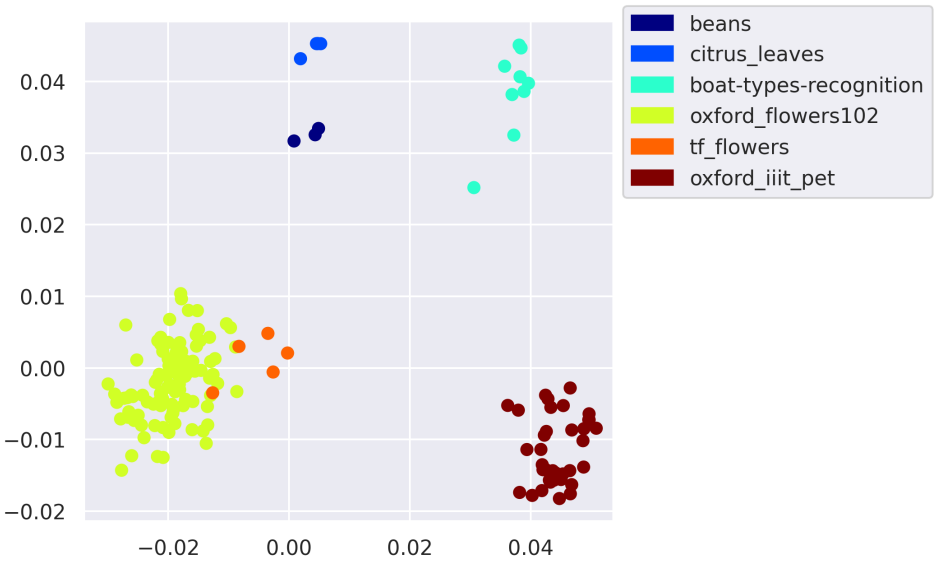


Figure 5.8: Principal component analysis of six datasets. Every point represents a class centroid. Points with the same color correspond to centroids belonging to the same dataset.

the dimensionality reduced version of a class centroid. Points with the same color correspond to centroids belonging to the same dataset.

We can observe that, even if the heatmap distance could provide non-optimal results, we can still have a good clusterization on different datasets.

Overall, we can state that we obtained mixed results on the considered contexts. One possible explanation can be found in the total absence of ground truth to compare to. In this sense, once a distance metric is computed we do not have a clear way to validate it.

To alleviate such a problem, we decided to consider the iNaturalist dataset as it provides an internal hierarchy that we can exploit to compare our metrics to.

Our analysis on the iNaturalist dataset started by addressing a simple question: does a distance measure defined by the random Fourier features correlate to a taxonomical distance? To address such a question we can consider Figure 5.3 where we can notice that two classes belonging to the same super-category will have a taxonomical distance equal to 2. At the same time, classes belonging to the different super category will provide a taxonomical distance equal to 4.

We started by considering two super-categories: protozoa and chromista. For every class, between every other class, we computed both the taxonomical distance and the distance according to random Fourier features. We report the results in Figure 5.9(left)

We can notice that classes belonging to the same super-category, i.e. the ones with a taxonomical distance equal to 2, result in random Fourier Feature distance that is lower w.r.t. classes belonging to the different super-categories. These results confirmed our intuition about the similarity between different classes.

Then we planned a second experiment with the same setting but the number of super-categories included was equal to three. The super-categories included in these experiments were: protozoa, chromista, and Aactinopterygii corresponding again to the smallest number of classes between

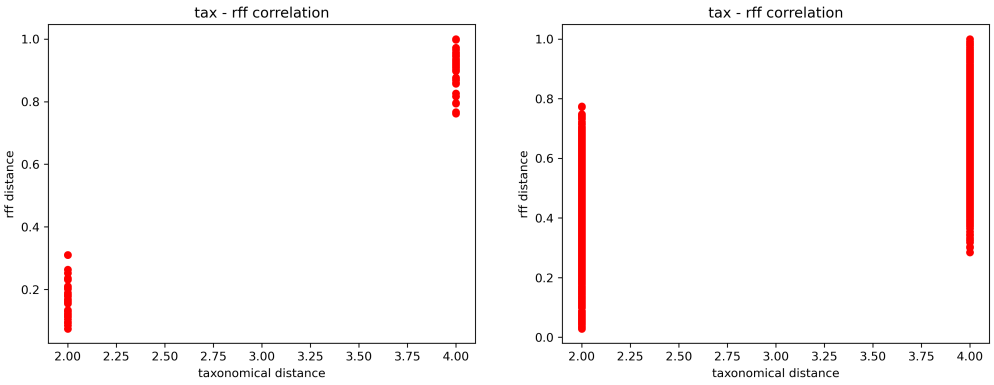


Figure 5.9: Correlation between taxonomical and random Fourier features distance. On the x-axis, we have the taxonomical distance between every class and every other class included in the two super-categories considered. On the y-axis, we have the corresponding distance according to the random Fourier features procedure. (left) Two super categories involved (right) Three super categories involved

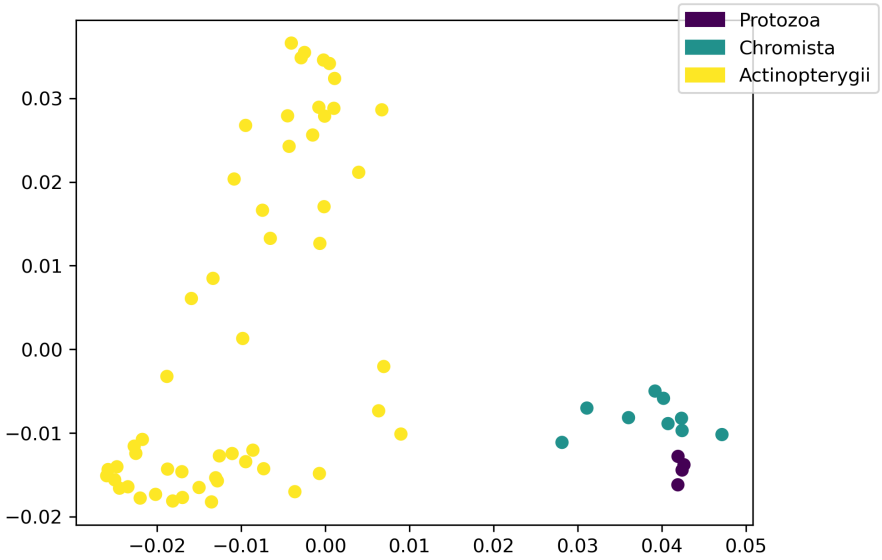


Figure 5.10: Principal component analysis of three super categories. Every point represents a class centroid. Points with the same color correspond to centroids belonging to the same super-category

the iNaturalist dataset. Results are shown in Figure 5.9(right). In this setting we can see that the representation of different super-categories is less tidy, as apparently, it can happen that the random Fourier Feature distance between two classes belonging to the same super-category can be greater than the one between two classes belonging to different super categories.
Guided by previous experiments with histograms, we also decided to check the clusterization of super-categories in a latent space with a very low dimension by projecting our centroids via a PCA technique. We can show such results in Figure 5.10

Also in this situation we can notice that apparently, even with a low number of dimensions, super-categories are well separated, providing promising results. It is worth noticing that a situation like the one shown in Figure 5.10 can partly explain the unsatisfactory results shown in Figure 5.9(right). Indeed we can notice that even if classes are well separated in the latent space, for instance, the distance between the two farthest classes of the Actinopterygii is comparable to the distance between some couples of Actinopterygii and chromista classes, resulting in the correlation plot shown before.

5.6 Discussion

In this chapter, we focused on different methodologies to compute a distance measure between different datasets. Similarly to previous chapters, we focused on image classification and even in this context we saw that it can be arduous to compare different classification problems. Intuitively, we tried to address the following question: given two datasets how can we measure how similar they are? In other words, can we compute a *distance* measure between the two datasets?
We decided to address these questions because it can be interesting to quantify similarity in a more principled way and to get an intuition about how different datasets relate one to another. This information could be very useful in all contexts where transferring information from previous contexts is a common practice. like domain adaptation or transfer learning.
We saw that it can be difficult to quantify this difference and build a relation between more than a few datasets. We saw that we can use pre-trained convolutional neural networks or Vision Transformers as feature extractors. Then we applied two different techniques. In subsection 5.5.1 we exploited histograms as feature density estimators to compute a distance measure between different datasets. In subsection 5.5.2 we considered the same group of datasets and we computed an approximation of the gaussian kernel to compute the similarity between them. With both techniques, we obtained mixed results. When dealing with a small number of datasets it is possible to compute a coherent metric between different datasets. This is confirmed both with histograms and the random Fourier features approach. At the same time, when working with a bigger amount of dataset is it difficult to have a clear scenario. However, further investigation about clusterization in low-dimensional latent space via principal component analysis technique, resulted in a good separation between different datasets with both approaches, showing encouraging results overall.

Part III

Remarks and conclusions

6 Conclusions

Machine learning techniques spread in practical scenarios, is mostly due to the impossibility of traditional techniques to deal with simple problems that require the retrieval of specific task-related information. However, machine learning has demonstrated remarkable capability to generalize to previously unseen data.
In chapter 2 we have seen that once we defined the basic ingredients needed in the machine learning context, i.e. task, performance measure, and experience, we can describe the problem with a model depending on a large number of parameters, each one providing just a part of the proposed solution. With the experience, coming from data and algorithm iterations we can optimize the parameter's value to improve on a certain task according to the selected performance measure.
We have seen that different machine learning settings, such as supervised and unsupervised learning or predictive and descriptive models, can be defined. The optimization process involved is usually based on an iterative technique, computing the gradient of the parameters and moving along the opposite direction.
A family of models fully relying on the optimization mechanism we just described, is the artificial neural networks one. The use of neural networks became, in the last decade, a common practice. In the beginning, neural networks were made of a very reduced amount of layers, with a limited capacity to solve complicated problems. In the last decade the complexity of neural networks, evaluated on the number of parameters involved and the computational time needed to train them, increased exponentially.
This set of methodologies we usually refer to as deep learning techniques became the de-facto standard in a large variety of fields. Their astonishing ability to solve different kinds of problems has been proven, from very simple and specific tasks to more general problems. Since 2012 with the ImageNet large-scale visual recognition challenge(ILSVRC), the dominant approach has been based on convolutional neural networks. Such models focus on a small portion of the image at-a-time via the convolution operation.
Convolutional neural networks have been applied to a large set of research fields including, but not limited to, image recognition, speech recognition, object detection of images, video recognition, and natural language processing.
In the last two years a new approach, referred to as transformers, has been proposed showing state-of-the-art performances in similar contexts to the ones covered by convolutional neural networks.

The huge improvement in performances obtained by recent models came at a cost from different points of view. The convolutional model introduced in 2012, i.e. AlexNet, which won the ILSVRC competition was made by 62.3 million learnable parameters with a training time of between five and six days on two GPUs. Since 2012 a plethora of different architectures has been proposed with an increasing number of parameters. Some model families such as ResNet, Efficient-

Net, and InceptionResNet stuck to a similar amount of parameters. Some others, like MobileNet, decided to reduce the number of parameters involved in order to be run on embedded/mobile devices. Nonetheless, architectures such as VGG moved in the opposite direction with more than 100 million learnable parameters.
Since 2017 with the introduction of the attention mechanism and the transformer architecture, the number of parameters involved in the training process increased exponentially. Some of the first introduced models had comparable parameters number w.r.t. convolutional models. Indeed, models such as ELMo and BERT are made by 94 million and 340 million learnable parameters, respectively. Nonetheless, many recent models pushed forward the number of learnable parameters involved. For instance, The GPT-2 model and the Megatron-LM, introduced in 2019, involved 1.5 billion and 8.3 billion learnable parameters, respectively. The GPT-3 model and the Megatron-Turing NLG models, introduced in 2020, involve more than a hundred billion parameters. The massive increase in the number of parameters involved in the process is, at least partly, coherent with the computational resources involved in the process: the amount of computation needed to train a modern neural architecture moved approximately from $4 \cdot 10^2$ PFLOPS of AlexNet to $3 \cdot 10^8$ PFLOPS of GPT-3.
The amount of energy needed to train the more recent architecture increased drastically in the last few years showing a problematic situation in terms of resources needed to obtain the next state-of-the-art performance.

In this thesis, we have seen different methodologies to alleviate the computational costs of some typical machine learning problems.
In chapter 3 we focused on image classification, considering a simple transfer learning approach that exploits pre-trained convolutional features as input for a fast kernel method. By performing more than 3000 training processes, involving 32 target datasets and 99 different settings, we showed that this fast-kernel approach provides comparable accuracy w.r.t. fine-tuning, with a training time that is between one and two orders of magnitude smaller.
Our results suggested that the fast-kernel approach provides a useful alternative to fine-tuning in small/medium datasets, especially when training efficiency is crucial. This is typical of robotics devices and autonomous systems, where multiple training may need to be done on the fly. Moreover, our results showed that the marginal benefit of fine-tuning is low dependent on the neural network architecture used as a pre-trained model. On the other hand, the choice of an appropriate pre-training dataset has a significant impact on the obtained accuracy, particularly for the fast-kernel methodology.
In chapter 4 we introduced and discussed the impossibility, of a clustering algorithm, to deal directly with images that, even with a low resolution, can have tens of thousands of dimensions. At the same time, the feature output size of modern architecture is usually between a thousand to tens of thousands of elements.
To this purpose, we implemented an unsupervised pipeline that projects the input to a latent space with reduced dimension, making the clustering operation doable.
We tested our pipeline effectiveness in the plankton monitoring context where operating in an unsupervised manner is crucial. Indeed, detecting and studying plankton populations in situ is paramount to protecting marine ecosystems as they can be regarded as biosensors, reflecting the overall health of the oceans.

In our work, we leveraged pre-trained neural network models to extract expressive feature maps in an efficient way, without fine-tuning. We then use an encoder-decoder network architecture to perform dimensionality reduction, producing low-dimensional embedded features that can then be fed to a clustering algorithm. We assessed our methodology on three plankton datasets with different characteristics and increasing complexity.
In chapter 5 we introduced and discussed different methodologies to compare two or more image datasets. Indeed, each dataset can be seen as a set of points sampled by an unknown distribution that we can estimate and analyze. Often, a key hallmark of deep learning models is the lack of good, labeled data. To overcome this process, different research fields focus on transferring knowledge from related but different data distributions. In this sense, estimating such distributions can be important to provide a better comprehension of our data and whether a set of data is suitable for transferring information to a new setting.
We introduced different ways to estimate such distributions such as histograms and kernel density estimation, coupled with various methodologies to compute distances between them such as integral probability metrics and f-divergences.
Our results showed that, even on simple tasks involving images, the concept of dataset distance is elusive and very complicated to quantify. In particular, three different approaches were tested providing similar results. Indeed, it is possible to obtain information on different image datasets, via good partitioning, as long as we analyze a small datasets subset.

Overall, in this thesis, we considered a set of techniques that can alleviate machine learning computational costs. In our work, we focused on three different efficiency aspects: training time, compressed data representation, and datasets distance. Each one of them focuses on a distinct efficiency aspect of machine learning efficiency.
Aiming for computational efficiency in machine learning tasks is necessary to reduce the greater budget required nowadays by such models. Reducing the cost of modern machine learning algorithms is going to be one of the greatest challenges we will face in the future, in order to keep them computationally accessible to the scientific community.
The field of efficient machine learning is dedicated to reducing the computational costs of machine learning models. It is constantly evolving through ongoing research and despite recent advances in this field, there are still several open problems and future research directions that need to be addressed. One key area is training time efficiency, where different techniques can be employed to reduce the amount of time required to train complex models. Another important area is data representation efficiency, which seeks to optimize the way in which data is represented, both in terms of compression and quality, to improve model accuracy and reduce computational overhead. Finally, data scarcity is another critical issue that needs to be addressed. Despite the significant advancements made through specialized techniques in recent years, there is still ample opportunity for further improvements in this field, particularly in the context of small and imbalanced datasets. Addressing these open problems and developing new techniques to improve machine learning efficiency will be crucial for the continued growth and success of the field.

Bibliography

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.
2. A. Abdul, J. Chen, H.-Y. Liao, and S.-H. Chang. "An emotion-aware personalized music recommendation system using a convolutional neural networks approach". *Applied Sciences* 8:7, 2018, p. 1103.
3. A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. C. Fowlkes, S. Soatto, and P. Perona. "Task2vec: Task embedding for meta-learning". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6430–6439.
4. G. Afendras and M. Markatou. "Optimality of training/test size and resampling effectiveness in cross-validation". *Journal of Statistical Planning and Inference* 199, 2019, pp. 286–301.
5. C. C. Aggarwal, A. Hinneburg, and D. A. Keim. "On the surprising behavior of distance metrics in high dimensional space". In: *International conference on database theory*. Springer. 2001, pp. 420–434.
6. S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.
7. U. C. Allard, F. Nougarou, C. L. Fall, P. Giguère, C. Gosselin, F. Laviolette, and B. Gosselin. "A convolutional neural network for robotic arm guidance using sEMG based frequency-features". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2464–2470.
8. M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari. "A state-of-the-art survey on deep learning theory and architectures". *Electronics* 8:3, 2019, p. 292.
9. M. A. Alvarez, L. Rosasco, N. D. Lawrence, et al. "Kernels for vector-valued functions: A review". *Foundations and Trends® in Machine Learning* 4:3, 2012, pp. 195–266.
10. D. Alvarez-Melis and N. Fusi. "Geometric dataset distances via optimal transport". *Advances in Neural Information Processing Systems* 33, 2020, pp. 21428–21439.

11. S.-i. Amari. "Backpropagation and stochastic gradient descent method". *Neurocomputing* 5:4-5, 1993, pp. 185–196.
12. S.-i. Amari, J. Ba, R. B. Grosse, X. Li, A. Nitanda, T. Suzuki, D. Wu, and J. Xu. "When does preconditioning help or hurt generalization?" In: *International Conference on Learning Representations*. 2020.
13. E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. "A comparison of extrinsic clustering evaluation metrics based on formal constraints". *Information retrieval* 12, 2009, pp. 461–486.
14. S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan. "Medical image analysis using convolutional neural networks: a review". *Journal of medical systems* 42:11, 2018, pp. 1–13.
15. V. Badrinarayanan, A. Kendall, and R. Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". *IEEE transactions on pattern analysis and machine intelligence* 39:12, 2017, pp. 2481–2495.
16. P. Baldi and R. Vershynin. "The capacity of feedforward neural networks". *Neural networks* 116, 2019, pp. 288–311.
17. M. J. Behrenfeld, J. T. Randerson, C. R. McClain, G. C. Feldman, S. O. Los, C. J. Tucker, P. G. Falkowski, C. B. Field, R. Frouin, W. E. Esaias, et al. "Biospheric primary production during an ENSO transition". *Science* 291:5513, 2001, pp. 2594–2597.
18. M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine-learning practice and the classical bias–variance trade-off". *Proceedings of the National Academy of Sciences* 116:32, 2019, pp. 15849–15854.
19. M. Belkin, S. Ma, and S. Mandal. "To understand deep learning we need to understand kernel learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 541–549.
20. M. C. Benfield, P. Grosjean, P. F. Culverhouse, X. Irigoien, M. E. Sieracki, A. Lopez-Urrutia, H. G. Dam, Q. Hu, C. S. Davis, A. Hansen, et al. "RAPID: research on automated plankton identification". *Oceanography* 20:2, 2007, pp. 172–187.
21. Y. Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.
22. M. Benzi. "Preconditioning techniques for large linear systems: a survey". *Journal of computational Physics* 182:2, 2002, pp. 418–477.
23. G. Bertasius, H. Wang, and L. Torresani. "Is space-time attention all you need for video understanding?" In: *ICML*. Vol. 2. 3. 2021, p. 4.
24. S. K. Biswas, T. Zimmerman, L. Maini, A. Adebiyi, L. Bozano, C. Brown, V. P. Pastore, and S. Bianco. "High throughput analysis of plankton morphology and dynamic". In: *Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues XVII*. Vol. 10881. International Society for Optics and Photonics. 2019, p. 1088109.

25. M. B. Blaschko, G. Holness, M. A. Mattar, D. Lisin, P. E. Utgoff, A. R. Hanson, H. Schultz, E. M. Riseman, M. E. Sieracki, W. M. Balch, and B. Tupper. "Automatic In Situ Identification of Plankton". In: *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05) - Volume 1*. Vol. 1. 2005, pp. 79–86. DOI: 10.1109/ACVMOT.2005.29.

26. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. "Learnability and the Vapnik-Chervonenkis dimension". *Journal of the ACM (JACM)* 36:4, 1989, pp. 929–965.

27. L. Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

28. D. Boyce, M. Lewis, and B. Worm. "Global phytoplankton decline over the past century". *Nature* 466, 2010, pp. 591–6. DOI: 10.1038/nature09268.

29. J. Bretagnolle and C. Huber. "Estimation des densités: risque minimax". *Séminaire de probabilités de Strasbourg* 12, 1978, pp. 342–363.

30. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". *Advances in neural information processing systems* 33, 2020, pp. 1877–1901.

31. M. Browne and S. S. Ghidary. "Convolutional neural networks for image processing: an application in robot vision". In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2003, pp. 641–652.

32. J. Camille. "Sur la serie de Fourier". *Camptesrendushebdomadaires des séances de l'Academie des sciences* 92, 1881, pp. 228–230.

33. A. Canziani, A. Paszke, and E. Culurciello. "An analysis of deep neural network models for practical applications". *arXiv preprint arXiv:1605.07678*, 2016.

34. F. Ceola, E. Maiettini, G. Pasquale, G. Meanti, L. Rosasco, and L. Natale. "Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot". *IEEE Transactions on Robotics* 38:5, 2022, pp. 3154–3172.

35. G. Chandrashekar and F. Sahin. "A survey on feature selection methods". *Computers & Electrical Engineering* 40:1, 2014, pp. 16–28.

36. O Chapelle, B Schölkopf, and A Zien. *Semi-Supervised Learning Cambridge*. 2006.

37. D. Chemkaeva. *Football vs Rugby*. 2020. URL: https://www.kaggle.com/lsind18/gemstones-images/download.

38. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". *IEEE transactions on pattern analysis and machine intelligence* 40:4, 2017, pp. 834–848.

39. Y. Cheng, D. Wang, P. Zhou, and T. Zhang. "A survey of model compression and acceleration for deep neural networks", 2017.

40. Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha. "StarGAN v2: Diverse Image Synthesis for Multiple Domains". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

41. F. Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.
42. V. Christlein, L. Spranger, M. Seuret, A. Nicolaou, P. Král, and A. Maier. "Deep generalized max pooling". In: *2019 International conference on document analysis and recognition (ICDAR)*. IEEE. 2019, pp. 1090–1096.
43. M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. "Describing Textures in the Wild". In: *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2014.
44. M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. "Describing textures in the wild". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 3606–3613.
45. T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. "Deep learning for classical japanese literature". *arXiv preprint arXiv:1812.01718*, 2018.
46. P.-A. Clorichel. *Boat types recognition*. 2018. URL: https://www.kaggle.com/clorichel/boat-types-recognition/download.
47. G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. "EMNIST: Extending MNIST to handwritten letters". In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.
48. J. Collins, J. Sohl-Dickstein, and D. Sussillo. "Capacity and Trainability in Recurrent Neural Networks". In: *International Conference on Learning Representations*.
49. P. Culverhouse, R Ellis, R. Simpson, R Williams, R. Pierce, and J. Turner. "Automatic categorisation of five species of Cymatocylis (Protozoa, Tintinnida) by artificial neural network". *Marine Ecology Progress Series*, 1994, pp. 273–280.
50. J. Dai, R. Wang, H. Zheng, G. Ji, and X. Qiao. "ZooplanktoNet: Deep convolutional network for zooplankton classification". In: *OCEANS 2016 - Shanghai*. 2016, pp. 1–6. DOI: 10.1109/OCEANSAP.2016.7485680.
51. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
52. L. Deng, G. Hinton, and B. 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.
53. S. Dieleman, J. De Fauw, and K. Kavukcuoglu. "Exploiting cyclic symmetry in convolutional neural networks". In: *International conference on machine learning*. PMLR. 2016, pp. 1889–1898.
54. K. K. Dobbin and R. M. Simon. "Optimally splitting cases for training and testing high dimensional classifiers". *BMC medical genomics* 4:1, 2011, pp. 1–8.
55. C. Doersch and A. Zisserman. "Multi-task self-supervised visual learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2051–2060.

56. P. Domingos. "A few useful things to know about machine learning". *Communications of the ACM* 55:10, 2012, pp. 78–87.

57. P. Domingos. "A unified bias-variance decomposition". In: *Proceedings of 17th international conference on machine learning*. Morgan Kaufmann Stanford. 2000, pp. 231–238.

58. A. Domínguez. "A History of the Convolution Operation [Retrospectroscope]". *IEEE Pulse* 6:1, 2015, pp. 38–49. DOI: 10.1109/MPUL.2014.2366903.

59. L. Dong, S. Xu, and B. Xu. "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 5884–5888.

60. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

61. J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* 12:7, 2011.

62. V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". *arXiv preprint arXiv:1603.07285*, 2016.

63. J. C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". *Journal of Cybernetics* 3:3, 1973, pp. 32–57. DOI: 10.1080/01969727308546046. eprint: https://doi.org/10.1080/01969727308546046. URL: https://doi.org/10.1080/01969727308546046.

64. J. S. Ellen, C. A. Graff, and M. D. Ohman. "Improving plankton image classification using context metadata". *Limnology and Oceanography: Methods* 17:8, 2019, pp. 439–461.

65. J. Elson, J. J. Douceur, J. Howell, and J. Saul. "Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization". In: *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., 2007.

66. L. Fei-Fei, R. Fergus, and P. Perona. "Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories". *Computer Vision and Image Understanding* 106:1, 2007, pp. 59–70.

67. R. A. Fisher. "On the mathematical foundations of theoretical statistics". *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 222:594-604, 1922, pp. 309–368.

68. P. Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012.

69. M. K. Fleming and G. W. Cottrell. "Categorization of faces using unsupervised feature extraction". In: *IJCNN*. 1990.

70. R. Fletcher and C. M. Reeves. "Function minimization by conjugate gradients". *The computer journal* 7:2, 1964, pp. 149–154.

71. G. B. Folland. *A course in abstract harmonic analysis*. Vol. 29. CRC press, 2016.

72. T. O. Fossum, G. M. Fragoso, E. J. Davies, J. E. Ullgren, R. Mendes, G. Johnsen, I. Ellingsen, J. Eidsvik, M. Ludvigsen, and K. Rajan. "Toward adaptive robotic sampling of phytoplankton in the coastal ocean". *Science Robotics* 4:27, 2019, eaav3041. DOI: 10.1126/scirobotics.aav3041. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aav3041. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aav3041.

73. S. Frizzi, R. Kaabi, M. Bouchouicha, J.-M. Ginoux, E. Moreau, and F. Fnaiech. "Convolutional neural network for video fire and smoke detection". In: *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2016, pp. 877–882.

74. Y. S. Fu, W. Ge, and J. Plested. "FERM: A FEature-space Representation Measure for Improved Model Evaluation". *work* 10, 2022, p. 11.

75. K. Fukushima and S. Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

76. D. Garcia-Gasulla, F. Parés, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura. "On the behavior of convolutional nets for feature extraction". *Journal of Artificial Intelligence Research* 61, 2018, pp. 563–592.

77. T. Gautam. *Football vs Rugby*. 2021. URL: https://www.kaggle.com/ligtfeather/football-vs-rugby-image-classification/download.

78. S. Geman, E. Bienenstock, and R. Doursat. "Neural networks and the bias/variance dilemma". *Neural computation* 4:1, 1992, pp. 1–58.

79. H. Gholamalinezhad and H. Khosravi. "Pooling methods in deep neural networks, a review". *arXiv preprint arXiv:2009.07485*, 2020.

80. S. Gidaris, A. Bursuc, N. Komodakis, P. Pérez, and M. Cord. "Boosting few-shot visual learning with self-supervision". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 8059–8068.

81. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

82. J. H. Grace and A. Young. *The algebra of invariants*. University Press, 1903.

83. A. Graves. "Long short-term memory". *Supervised sequence labelling with recurrent neural networks*, 2012, pp. 37–45.

84. J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. "Recent advances in convolutional neural networks". *Pattern recognition* 77, 2018, pp. 354–377.

85. J. Guérin, O. Gibaru, S. Thiery, and E. Nyiri. "CNN Features are also Great at Unsupervised Classification". In: *8th International Conference on Computer Science, Engineering and Applications*. Academy & Industry Research Collaboration Center (AIRCC). 2018, pp. 83–95.

86. A Hadjidimos. "Successive overrelaxation (SOR) and related methods". *Journal of Computational and Applied Mathematics* 123:1-2, 2000, pp. 177–199.

87. D. Han, J. Kim, and J. Kim. "Deep pyramidal residual networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5927–5935.

88. F. He, T. Liu, and D. Tao. "Control batch size and learning rate to generalize well: Theoretical and empirical evidence". *Advances in Neural Information Processing Systems* 32, 2019, pp. 1143–1152.

89. K. He, G. Gkioxari, P. Dollár, and R. Girshick. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

90. K. He and J. Sun. "Convolutional Neural Networks at Constrained Time Cost". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

91. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

92. P. Helber, B. Bischke, A. Dengel, and D. Borth. "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification". *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.

93. E. Hellinger. "Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen." *Journal für die reine und angewandte Mathematik* 1909:136, 1909, pp. 210–271.

94. M. R. Hestenes, E. Stiefel, et al. "Methods of conjugate gradients for solving linear systems". *Journal of research of the National Bureau of Standards* 49:6, 1952, pp. 409–436.

95. S. Hijazi, R. Kumar, C. Rowen, et al. "Using convolutional neural networks for image recognition". *Cadence Design Systems Inc.: San Jose, CA, USA* 9, 2015.

96. D. E. Hilt and D. W. Seegrist. *Ridge, a computer program for calculating ridge regression estimates*. Department of Agriculture, Forest Service, Northeastern Forest Experiment …, 1977.

97. A. Hinneburg, C. C. Aggarwal, and D. A. Keim. "What is the nearest neighbor in high dimensional spaces?" In: *26th Internat. Conference on Very Large Databases*. 2000, pp. 506–515.

98. G. Hinton, N. Srivastava, and K. Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". *Cited on* 14:8, 2012, p. 2.

99. G. E. Hinton and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". *science* 313:5786, 2006, pp. 504–507.

100. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". *arXiv preprint arXiv:1207.0580*, 2012.

101. S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation* 9:8, 1997, pp. 1735–1780.

102. A. E. Hoerl and R. W. Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". *Technometrics* 12:1, 1970, pp. 55–67.

103. J. J. Hopfield. "Neurons with graded response have collective computational properties like those of two-state neurons." *Proceedings of the national academy of sciences* 81:10, 1984, pp. 3088–3092.

104. R. A. Horn. "The hadamard product". In: *Proc. Symp. Appl. Math*. Vol. 40. 1990, pp. 87–169.

105. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". *CoRR* abs/1704.04861, 2017. arXiv: 1704.04861.

106. B. Hu, Z. Lu, H. Li, and Q. Chen. "Convolutional neural network architectures for matching natural language sentences". *Advances in neural information processing systems* 27, 2014.

107. Q. Hu and C. Davis. "Automatic plankton image recognition with co-occurrence matrices and support vector machine". *Marine Ecology Progress Series* 295, 2005, pp. 21–31.

108. G. Huang, Z. Liu, and K. Q. Weinberger. "Densely Connected Convolutional Networks". *CoRR* abs/1608.06993, 2016. arXiv: 1608.06993.

109. J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7310–7311.

110. A. J. Hughes, J. D. Mornin, S. K. Biswas, D. P. Bauer, S. Bianco, and Z. J. Gartner. "Quantius: Generic, high-fidelity human annotation of scientific images at 105-clicks-per-hour". *bioRxiv*, 2017. DOI: 10.1101/164087. eprint: https://www.biorxiv.org/content/early/2017/07/15/164087.full.pdf. URL: https://www.biorxiv.org/content/early/2017/07/15/164087.

111. M. Huh, P. Agrawal, and A. A. Efros. "What makes ImageNet good for transfer learning?" *CoRR* abs/1608.08614, 2016. arXiv: 1608.08614.

112. J. J. Hull. "A database for handwritten text recognition research". *IEEE Transactions on pattern analysis and machine intelligence* 16:5, 1994, pp. 550–554.

113. M. Hussain, J. J. Bird, and D. R. Faria. "A study on cnn transfer learning for image classification". In: *UK Workshop on computational Intelligence*. Springer. 2018, pp. 191–202.

114. M. L. Hutchinson, E. Antono, B. M. Gibbons, S. Paradiso, J. Ling, and B. Meredig. "Overcoming data scarcity with transfer learning". *CoRR* abs/1711.05099, 2017. arXiv: 1711.05099. URL: http://arxiv.org/abs/1711.05099.

115. I. K. M. Jais, A. R. Ismail, and S. Q. Nisa. "Adam optimization algorithm for wide and deep neural network". *Knowledge Engineering and Data Science* 2:1, 2019, pp. 41–46.

116. M. Jamil and X.-S. Yang. "A literature survey of benchmark functions for global optimisation problems". *International Journal of Mathematical Modelling and Numerical Optimisation* 4:2, 2013, pp. 150–194.

117. L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu. "A survey of deep learning-based object detection". *IEEE access* 7, 2019, pp. 128837–128868.

118. J. Jin, A. Dundar, and E. Culurciello. "Flattened Convolutional Neural Networks for Feedforward Acceleration". In: *International Conference on Learning Representations Workshop*. 2015.

119. L. Jing and Y. Tian. "Self-supervised visual feature learning with deep neural networks: A survey". *IEEE transactions on pattern analysis and machine intelligence* 43:11, 2020, pp. 4037–4058.

120. M. Jogin, Mohana, M. S. Madhulika, G. D. Divya, R. K. Meghana, and S Apoorva. "Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning". In: *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2018, pp. 2319–2323. DOI: 10.1109/RTEICT42901.2018.9012507.

121. L. V. Kantorovich. "Mathematical methods of organizing and planning production". *Management science* 6:4, 1960, pp. 366–422.

122. L. V. Kantorovich. "On the translocation of masses". *Journal of mathematical sciences* 133:4, 2006, pp. 1381–1382.

123. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. "Large-scale video classification with convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.

124. J. N. Kather, F. G. Zöllner, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and C.-A. Weis. *Collection of textures in colorectal cancer histology*. 2016. DOI: 10.5281/zenodo.53169.

125. J. D. M.-W. C. Kenton and L. K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of NAACL-HLT*. 2019, pp. 4171–4186.

126. D. S. Kermany, M. Goldbaum, W. Cai, C. C. Valentim, H. Liang, S. L. Baxter, A. McKeown, G. Yang, X. Wu, F. Yan, et al. "Identifying medical diagnoses and treatable diseases by image-based deep learning". *Cell* 172:5, 2018, pp. 1122–1131.

127. D. S. Kershaw. "The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations". *Journal of computational physics* 26:1, 1978, pp. 43–65.

128. N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *International Conference on Learning Representations*.

129. S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. "Transformers in vision: A survey". *ACM computing surveys (CSUR)* 54:10s, 2022, pp. 1–41.

130. S. Khirirat, H. R. Feyzmahdavian, and M. Johansson. "Mini-batch gradient descent: Faster convergence under data sparsity". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017, pp. 2880–2887.

131. A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei. "Novel Dataset for Fine-Grained Image Categorization : Stanford Dogs". In: 2012.

132. G. Kimeldorf and G. Wahba. "Some results on Tchebycheffian spline functions". *Journal of mathematical analysis and applications* 33:1, 1971, pp. 82–95.

133. D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

134. D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2014.

135. D. P. Kingma, M. Welling, et al. "An introduction to variational autoencoders". *Foundations and Trends® in Machine Learning* 12:4, 2019, pp. 307–392.

136. A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. "Big transfer (bit): General visual representation learning". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*. Springer. 2020, pp. 491–507.

137. S. Kornblith, J. Shlens, and Q. V. Le. "Do better ImageNet models transfer better?" In: 2019.

138. G. Köthe. "Topological vector spaces". In: *Topological Vector Spaces I*. Springer, 1983, pp. 123–201.

139. C. Kraft. "Some conditions for consistency and uniform consistency of statistical procedures". *University of California Publication in Statistics* 2, 1955, pp. 125–141.

140. M. A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks". *AIChE journal* 37:2, 1991, pp. 233–243.

141. J. Krause, M. Stark, J. Deng, and L. Fei-Fei. "3D Object Representations for Fine-Grained Categorization". In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

142. A. Krizhevsky. *Learning multiple layers of features from tiny images*. Technical report. 2009. Chap. Chapter 3.

143. A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

144. A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

145. S. Kullback and R. A. Leibler. "On information and sufficiency". *The annals of mathematical statistics* 22:1, 1951, pp. 79–86.

146. S. Kumra and C. Kanan. "Robotic grasp detection using deep convolutional neural networks". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 769–776.

147. M. A. Lab. *Bean disease dataset*. 2020. URL: https://github.com/AI-Lab-Makerere/ibean/.

148. K. J. Lang, A. H. Waibel, and G. E. Hinton. "A Time-delay Neural Network Architecture for Isolated Word Recognition". *Neural Netw.* 3:1, 1990, pp. 23–43. URL: http://dx.doi.org/10.1016/0893-6080(90)90044-L.

149. K. Lange. *Optimization*. Vol. 95. Springer Science & Business Media, 2013.

150. J. Larsen and C. Goutte. "On optimal data split for generalization estimation and model selection". In: *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (Cat. No. 98TH8468)*. IEEE. 1999, pp. 225–234.

151. G. Larsson, M. Maire, and G. Shakhnarovich. "FractalNet: Ultra-Deep Neural Networks without Residuals". In: *International Conference on Learning Representations*.

152. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". *Neural Comput.* 1:4, 1989, pp. 541–551. URL: http://dx.doi.org/10.1162/neco.1989.1.4.541.

153. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". *Neural Computation* 1:4, 1989, pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

154. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* 86:11, 1998, pp. 2278–2324.

155. Y. LeCun, C. Cortes, and C. Burges. *MNIST handwritten digit database*. 2010.

156. H. Lee, M. Park, and J. Kim. "Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning". In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3713–3717.

157. C. Lemaréchal. "Cauchy and the gradient method". *Doc Math Extra* 251:254, 2012, p. 10.

158. D. Li, X. Chen, M. Becchi, and Z. Zong. "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs". In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*. 2016, pp. 477–484. DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.76.

159. H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. "Pruning Filters for Efficient ConvNets". In: *International Conference on Learning Representations*.

160. M. Li, T. Zhang, Y. Chen, and A. J. Smola. "Efficient mini-batch training for stochastic optimization". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 661–670.

161. Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. "Medical image classification with convolutional neural network". In: *2014 13th international conference on control automation robotics & vision (ICARCV)*. IEEE. 2014, pp. 844–848.

162. Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. "A survey of convolutional neural networks: analysis, applications, and prospects". *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

163. Z. Li, Z. Qin, K. Huang, X. Yang, and S. Ye. "Intrusion detection using convolutional neural networks for representation learning". In: *International conference on neural information processing*. Springer. 2017, pp. 858–866.

164. J. Liesen and Z. Strakos. *Krylov subspace methods: principles and analysis*. Oxford University Press, 2013.

165. J. Lin. "Divergence measures based on the Shannon entropy". *IEEE Transactions on Information theory* 37:1, 1991, pp. 145–151.

166. G. W. Lindsay. "Convolutional neural networks as a model of the visual system: Past, present, and future". *Journal of cognitive neuroscience* 33:10, 2021, pp. 2017–2031.

167. Z. C. Lipton, J. Berkowitz, and C. Elkan. "A critical review of recurrent neural networks for sequence learning". *arXiv preprint arXiv:1506.00019*, 2015.

168. L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. "Deep learning for generic object detection: A survey". *International journal of computer vision* 128:2, 2020, pp. 261–318.

169. X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. "Self-supervised learning: Generative or contrastive". *IEEE Transactions on Knowledge and Data Engineering*, 2021.

170. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10012–10022.

171. S. Long, X. He, and C. Yao. "Scene text detection and recognition: The deep learning era". *International Journal of Computer Vision* 129:1, 2021, pp. 161–184.

172. L. H. Loomis. *Introduction to abstract harmonic analysis*. Courier Corporation, 2013.

173. A. Lumini and L. Nanni. "Deep learning and transfer learning features for plankton classification". *Ecological informatics* 51, 2019, pp. 33–43.

174. J. Y. Luo, J.-O. Irisson, B. Graham, C. Guigand, A. Sarafraz, C. Mader, and R. K. Cowen. "Automated plankton image analysis using convolutional neural networks". *Limnology and Oceanography: Methods* 16:12, 2018, pp. 814–827. DOI: https://doi.org/10.1002/lom3.10285. eprint: https://aslopubs.onlinelibrary.wiley.com/doi/pdf/10.1002/lom3.10285. URL: https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.1002/lom3.10285.

175. A. Lydia and S. Francis. "Adagrad—an optimizer for stochastic gradient descent". *Int. J. Inf. Comput. Sci* 6:5, 2019, pp. 566–568.

176. L. Van der Maaten and G. Hinton. "Visualizing data using t-SNE." *Journal of machine learning research* 9:11, 2008.

177. E. Maiettini, G. Pasquale, L. Rosasco, and L. Natale. "Speeding-up object detection training for robotics with falkon". In: *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2018, pp. 5770–5776.

178. S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. *Fine-Grained Visual Classification of Aircraft*. Technical report. 2013. arXiv: 1306.5151 [cs-cv].

179. E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir. "Proving the lottery ticket hypothesis: Pruning is all you need". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6682–6691.

180. K. Malde, N. O. Handegard, L. Eikvil, and A.-B. Salberg. "Machine intelligence and the data-driven future of marine science". *ICES Journal of Marine Science* 77:4, 2019, pp. 1274–1285. ISSN: 1054-3139. DOI: 10.1093/icesjms/fsz057. eprint: https://academic.oup.com/icesjms/article-pdf/77/4/1274/33513034/fsz057.pdf. URL: https://doi.org/10.1093/icesjms/fsz057.

181. F. Mamalet and C. Garcia. "Simplifying ConvNets for Fast Learning". In: *Artificial Neural Networks and Machine Learning – ICANN 2012*. Ed. by A. E. P. Villa, W. Duch, P. Érdi, F. Masulli, and G. Palm. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 58–65. ISBN: 978-3-642-33266-1.

182. M. Manguoglu, M. Koyutürk, A. H. Sameh, and A. Grama. "Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers". *SIAM journal on Scientific Computing* 32:3, 2010, pp. 1201–1216.

183. I. Masi, Y. Wu, T. Hassner, and P. Natarajan. "Deep face recognition: A survey". In: *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE. 2018, pp. 471–478.

184. D. Masters and C. Luschi. "Revisiting small batch training for deep neural networks". *arXiv preprint arXiv:1804.07612*, 2018.

185. A. Mathis, P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis, and M. Bethge. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". *Nature neuroscience* 21:9, 2018, pp. 1281–1289.

186. W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." *Bulletin of mathematical biology* 52 1-2, 1943, pp. 99–115;

187. G. Meanti, L. Carratino, E. De Vito, and L. Rosasco. "Efficient Hyperparameter Tuning for Large Scale Kernel Ridge Regression". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 6554–6572.

188. G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. "Kernel methods through the roof: handling billions of points efficiently". *Advances in Neural Information Processing Systems* 33, 2020, pp. 14410–14422.

189. J. Mercer. "Xvi. functions of positive and negative type, and their connection the theory of integral equations". *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character* 209:441-458, 1909, pp. 415–446.

190. F. Milletari, N. Navab, and S.-A. Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation". In: *2016 fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 565–571.

191. E. Million. "The hadamard product". *Course Notes* 3:6, 2007.

192. H. Minkowski. *Geometrie der zahlen*. BG Teubner, 1910.

193. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

194. T. M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

195. M. Moro, V. P. Pastore, C. Tacchino, P. Durand, I. Blanchi, P. Moretti, F. Odone, and M. Casadio. "A markerless pipeline to analyze spontaneous movements of preterm infants". *Computer Methods and Programs in Biomedicine* 226, 2022, p. 107119. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2022.107119. URL: https://www.sciencedirect.com/science/article/pii/S0169260722005004.

196. L. Moroney. *Horses or Humans Dataset*. 2019. URL: http://laurencemoroney.com/horses-or-humans-dataset.

197. C. Musco and C. Musco. "Recursive sampling for the nystrom method". *Advances in neural information processing systems* 30, 2017.

198. E. Mwebaze, T. Gebru, A. Frome, S. Nsumba, and J. Tusubira. *iCassava 2019Fine-Grained Visual Categorization Challenge*. 2019. arXiv: 1908.02900 [cs.CV].

199. J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. "Max-pooling convolutional neural networks for vision-based hand gesture recognition". In: *2011 IEEE international conference on signal and image processing applications (ICSIPA)*. IEEE. 2011, pp. 342–347.

200. A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. "Speech recognition using deep neural networks: A systematic review". *IEEE access* 7, 2019, pp. 19143–19165.

201. N. Nayman, A. Golbert, A. Noy, T. Ping, and L. Zelnik-Manor. *Diverse Imagenet Models Transfer Better*. 2022. DOI: 10.48550/ARXIV.2204.09134. URL: https://arxiv.org/abs/2204.09134.

202. C. Nguyen, T. Hassner, M. Seeger, and C. Archambeau. "Leep: A new measure to evaluate transferability of learned representations". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7294–7305.

203. M.-E. Nilsback and A. Zisserman. "Automated Flower Classification over a Large Number of Classes". In: *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*. 2008.

204. E. J. Nyström. "Über Die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben". *Acta Mathematica* 54:none, 1930, pp. 185 –204. DOI: 10.1007/BF02547521. URL: https://doi.org/10.1007/BF02547521.

205. A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, B. Calvert, M. Rahimi Azghadi, and R. D. White. "DeepWeeds: A Multiclass Weed Species Image Dataset for Deep Learning". *Scientific Reports* 9:2058, 1 2019. DOI: 10.1038/s41598-018-38343-3. URL: https://doi.org/10.1038/s41598-018-38343-3.

206. E. C. Orenstein and O. Beijbom. "Transfer learning and deep feature extraction for planktonic image data sets". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 1082–1088.

207. N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. "Deep learning vs. traditional computer vision". In: *Science and information conference*. Springer. 2019, pp. 128–144.

208. S. J. Pan and Q. Yang. "A survey on transfer learning". *IEEE Transactions on knowledge and data engineering* 22:10, 2009, pp. 1345–1359.

209. O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. "Cats and Dogs". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012.

210. N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. "Image transformer". In: *International conference on machine learning*. PMLR. 2018, pp. 4055–4064.

211. G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale. "Are we done with object recognition? The iCub robot's perspective". *Robotics and Autonomous Systems* 112, 2019, pp. 260–281.

212. V. P. Pastore, T. Zimmerman, S. K. Biswas, and S. Bianco. "Establishing the baseline for using plankton as biosensor". In: *Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues XVII*. Vol. 10881. International Society for Optics and Photonics. 2019, 108810H.

213. V. P. Pastore, T. G. Zimmerman, S. Biswas, and S. Bianco. "Annotation-free Learning of Plankton for Classification and Anomaly Detection". *bioRxiv*, 2019. DOI: 10.1101/856815. eprint: https://www.biorxiv.org/content/early/2019/11/27/856815.full.pdf. URL: https://www.biorxiv.org/content/early/2019/11/27/856815.

214. V. P. Pastore, T. G. Zimmerman, S. K. Biswas, and S. Bianco. "Annotation-free learning of plankton for classification and anomaly detection". *Scientific Reports* 10:1, 2020, p. 12142. ISSN: 2045-2322. DOI: 10.1038/s41598-020-68662-3. URL: https://doi.org/10.1038/s41598-020-68662-3.

215. V. P. Pastore, N. Megiddo, and S. Bianco. "An Anomaly Detection Approach for Plankton Species Discovery". In: *Image Analysis and Processing – ICIAP 2022*. Ed. by S. Sclaroff, C. Distante, M. Leo, G. M. Farinella, and F. Tombari. Springer International Publishing, Cham, 2022, pp. 599–609. ISBN: 978-3-031-06430-2.

216. V. P. Pastore, M. Moro, and F. Odone. "A semi-automatic toolbox for markerless effective semantic feature extraction". *Scientific Reports* 12:1, 2022, pp. 1–12.

217. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://

papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

218. K. Pearson. "LIII. On lines and planes of closest fit to systems of points in space". *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2:11, 1901, pp. 559–572.

219. K. Pearson. "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling". *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50:302, 1900, pp. 157–175.

220. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. "Deep Contextualized Word Representations". In: *NAACL-HLT*. Association for Computational Linguistics, 2018, pp. 2227–2237.

221. G. Peyré, M. Cuturi, et al. "Computational optimal transport: With applications to data science". *Foundations and Trends® in Machine Learning* 11:5-6, 2019, pp. 355–607.

222. B. T. Pham, I. Prakash, A. Jaafari, and D. T. Bui. "Spatial prediction of rainfall-induced landslides using aggregating one-dependence estimators classifier". *Journal of the Indian Society of Remote Sensing* 46:9, 2018, pp. 1457–1470.

223. R. R. Picard and K. N. Berk. "Data splitting". *The American Statistician* 44:2, 1990, pp. 140–147.

224. L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen. "Sign language recognition using convolutional neural networks". In: *European conference on computer vision*. Springer. 2014, pp. 572–578.

225. E. Polak and G. Ribiere. "Note sur la convergence de méthodes de directions conjuguées". *Revue française d'informatique et de recherche opérationnelle. Série rouge* 3:16, 1969, pp. 35–43.

226. S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar. "A survey on deep learning: Algorithms, techniques, and applications". *ACM Computing Surveys (CSUR)* 51:5, 2018, pp. 1–36.

227. N. Prabhavalkar. *Indian food*. 2020. URL: https://www.kaggle.com/nehaprabhavalkar/indian-food-101/download.

228. L. Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, 2002, pp. 55–69.

229. N. Qian. "On the momentum term in gradient descent learning algorithms". *Neural networks* 12:1, 1999, pp. 145–151.

230. H. Qin, X. Li, Z. Yang, and M. Shang. "When underwater imagery analysis meets deep learning: A solution at the age of big visual data". In: *OCEANS 2015 - MTS/IEEE Washington*. 2015, pp. 1–5. DOI: 10.23919/OCEANS.2015.7404463.

231. Z. Qin, F. Yu, C. Liu, and X. Chen. "How convolutional neural networks see the world—A survey of convolutional neural network visualization methods". *Mathematical Foundations of Computing* 1:2, 2018.

232. A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training", 2018.

233. A. Rahimi and B. Recht. "Random features for large-scale kernel machines". *Advances in neural information processing systems* 20, 2007.

234. S. Rajaraman, S. K. Antani, M. Poostchi, K. Silamut, M. A. Hossain, R. J. Maude, S. Jaeger, and G. R. Thoma. "Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images". *PeerJ* 6, 2018, e4568.

235. D. Rathi, S. Jain, and S Indu. "Underwater fish species classification using convolutional neural network and deep learning". In: *2017 Ninth international conference on advances in pattern recognition (ICAPR)*. IEEE. 2017, pp. 1–6.

236. H. T. Rauf, B. A. Saleem, M. I. U. Lali, M. A. Khan, M. Sharif, and S. A. C. Bukhari. "A citrus fruits and leaves dataset for detection and classification of citrus diseases through machine learning". *Data in brief* 26, 2019, p. 104340.

237. W. Rawat and Z. Wang. "Deep convolutional neural networks for image classification: A comprehensive review". *Neural computation* 29:9, 2017, pp. 2352–2449.

238. R. Reed. "Pruning algorithms-a survey". *IEEE Transactions on Neural Networks* 4:5, 1993, pp. 740–747.

239. T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor. "ImageNet-21K Pretraining for the Masses". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

240. A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. L. Zitnick, J. Ma, et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". *Proceedings of the National Academy of Sciences* 118:15, 2021, e2016239118.

241. H. Robbins and S. Monro. "A Stochastic Approximation Method". *The Annals of Mathematical Statistics* 22:3, 1951, pp. 400–407. URL: https://doi.org/10.1214/aoms/1177729586.

242. R. Rojas. "The backpropagation algorithm". In: *Neural networks*. Springer, 1996, pp. 149–182.

243. A. Rudi, L. Carratino, and L. Rosasco. "Falkon: An optimal large scale kernel method". *Advances in neural information processing systems* 30, 2017.

244. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Learning representations by back-propagating errors". *nature* 323:6088, 1986, pp. 533–536.

245. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Neurocomputing: Foundations of Research". In: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

246. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". *International journal of computer vision* 115, 2015, pp. 211–252.

247. S. J. Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

248. A. O. Salau and S. Jain. "Feature extraction: a survey of the types, techniques, applications". In: *2019 international conference on signal processing and communication (ICSC)*. IEEE. 2019, pp. 158–164.

249. H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee. "Recent advances in recurrent neural networks". *arXiv preprint arXiv:1801.01078*, 2017.

250. F. Santosa and W. W. Symes. "Linear inversion of band-limited reflection seismograms". *SIAM Journal on Scientific and Statistical Computing* 7:4, 1986, pp. 1307–1330.

251. J. Sarzynska-Wawer, A. Wawer, A. Pawlak, J. Szymanowska, I. Stefaniak, M. Jarkiewicz, and L. Okruszek. "Detecting formal thought disorder by deep contextualized word representations". *Psychiatry Research* 304, 2021, p. 114135.

252. C. Saunders, A. Gammerman, and V. Vovk. "Ridge Regression Learning Algorithm in Dual Variables". In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, 515–521. ISBN: 1558605568.

253. D. Scherer, A. Müller, and S. Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition". In: *International conference on artificial neural networks*. Springer. 2010, pp. 92–101.

254. M. S. Schmid, C. Aubry, J. Grigor, and L. Fortier. "The LOKI underwater imaging system and an automatic identification model for the detection of zooplankton taxa in the Arctic Ocean". *Methods in Oceanography* 15-16, 2016. Computer Vision in Oceanography, pp. 129–160. ISSN: 2211-1220. DOI: https://doi.org/10.1016/j.mio.2016.03.003. URL: https://www.sciencedirect.com/science/article/pii/S2211122015300050.

255. B. Schölkopf, R. Herbrich, and A. J. Smola. "A generalized representer theorem". In: *International conference on computational learning theory*. Springer. 2001, pp. 416–426.

256. B. Schölkopf, A. Smola, and K.-R. Müller. "Nonlinear component analysis as a kernel eigenvalue problem". *Neural computation* 10:5, 1998, pp. 1299–1319.

257. S.-M. Schröder, R. Kiko, and R. Koch. "Morphocluster: efficient annotation of Plankton images by clustering". *Sensors* 20:11, 2020, p. 3060.

258. K. Schulze, U. M. Tillich, T. Dandekar, and M. Frohme. "PlanktoVision - an automated analysis system for the identification of phytoplankton". *BMC Bioinformatics* 14:1, 2013, p. 115. ISSN: 1471-2105. DOI: 10.1186/1471-2105-14-115. URL: https://doi.org/10.1186/1471-2105-14-115.

259. P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas, and A. A. Lee. "Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction". *ACS central science* 5:9, 2019, pp. 1572–1583.

260. M. Shaha and M. Pawar. "Transfer learning for image classification". In: *2018 second international conference on electronics, communication and aerospace technology (ICECA)*. IEEE. 2018, pp. 656–660.

261. L. Shao, F. Zhu, and X. Li. "Transfer learning for visual categorization: A survey". *IEEE transactions on neural networks and learning systems* 26:5, 2014, pp. 1019–1034.

262. A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. 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.

263. V. Simoncini and D. B. Szyld. "Recent computational developments in Krylov subspace methods for linear systems". *Numerical Linear Algebra with Applications* 14:1, 2007, pp. 1–59.

264. A. Singhal, P. Sinha, and R. Pant. "Use of Deep Learning in Modern Recommendation System: A Summary of Recent Works". *International Journal of Computer Applications* 975, p. 8887.

265. A. Smola, A. Gretton, L. Song, and B. Schölkopf. "A Hilbert space embedding for distributions". In: *Algorithmic Learning Theory: 18th International Conference, ALT 2007, Sendai, Japan, October 1-4, 2007. Proceedings 18*. Springer. 2007, pp. 13–31.

266. J. Snell, K. Swersky, and R. Zemel. "Prototypical networks for few-shot learning". *Advances in neural information processing systems* 30, 2017.

267. E. Soares and P. Angelov. *A large dataset of real patients CT scans for COVID-19 identification*. Version V1. 2020. URL: https://doi.org/10.7910/DVN/SZDUQX.

268. "Some studies in machine learning using the game of checkers". *IBM Journal of research and development* 44:1.2, 2000, pp. 206–226.

269. H. M. Sosik and R. J. Olson. "Automated taxonomic classification of phytoplankton sampled with imaging-in-flow cytometry". *Limnology and Oceanography: Methods* 5:6, 2007, pp. 204–216.

270. H. M. Sosik and R. J. Olson. "Automated taxonomic classification of phytoplankton sampled with imaging-in-flow cytometry". *Limnology and Oceanography: Methods* 5:6, 2007, pp. 204–216. DOI: https://doi.org/10.4319/lom.2007.5.204. eprint: https://aslopubs.onlinelibrary.wiley.com/doi/pdf/10.4319/lom.2007.5.204. URL: https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.4319/lom.2007.5.204.

271. P. Srivastava. *Multi-class Weather*. 2020. URL: https://www.kaggle.com/pratik2901/multiclass-weather-dataset/download.

272. E. Strubell, A. Ganesh, and A. McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 3645–3650.

273. F. Sultana, A. Sufian, and P. Dutta. "Evolution of image segmentation using deep convolutional neural network: a survey". *Knowledge-Based Systems* 201, 2020, p. 106062.

274. M. Sun, Z. Song, X. Jiang, J. Pan, and Y. Pang. "Learning pooling for convolutional neural network". *Neurocomputing* 224, 2017, pp. 96–104.

275. P. Sunga. *Make up vs No Make up*. 2018. URL: https://www.kaggle.com/petersunga/make-up-vs-no-make-up/download.

276. I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

277. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

278. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

279. N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE transactions on medical imaging* 35:5, 2016, pp. 1299–1312.

280. M. Tan and Q. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

281. A. challenge team. *Best artworks of all time*. 2019. URL: https://www.kaggle.com/ikarus777/best-artworks-of-all-time/download.

282. T. T. Team. *Flowers*. 2019. URL: http://download.tensorflow.org/example_images/flower_photos.tgz.

283. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65:6, 1958, p. 386.

284. R. Tibshirani. "Regression shrinkage and selection via the lasso". *Journal of the Royal Statistical Society: Series B (Methodological)* 58:1, 1996, pp. 267–288.

285. T Tieleman and G Hinton. "Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning". *Technical report*, 2017.

286. I. du Toit. "Enhanced Deep Learning Feature Extraction for Plankton Taxonomy". In: *Proceedings of the International Conference on Artificial Intelligence and its Applications*. 7. Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–8. ISBN: 978-1-4503-8575-6. URL: https://doi.org/10.1145/3487923.3487930 (visited on 01/20/2022).

287. G.-L. Tran, E. V. Bonilla, J. Cunningham, P. Michiardi, and M. Filippone. "Calibrating deep convolutional gaussian processes". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR. 2019, pp. 1554–1563.

288. B. B. Traore, B. Kamsu-Foguem, and F. Tangara. "Deep convolution neural network for image recognition". *Ecological Informatics* 48, 2018, pp. 257–268.

289. A. B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer series in statistics. Springer, Dordrecht, 2009. DOI: 10.1007/b13794. URL: https://cds.cern.ch/record/1315296.

290. O. Ulucan, D. Karakaya, and M. Turkan. "Meat Quality Assessment based on Deep Learning". In: *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. 2019, pp. 1–5. DOI: 10.1109/ASYU48272.2019.8946388.

291. J. E. Van Engelen and H. H. Hoos. "A survey on semi-supervised learning". *Machine Learning* 109:2, 2020, pp. 373–440.

292. G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie. *The iNaturalist Species Classification and Detection Dataset*. 2017. DOI: 10.48550/ARXIV.1707.06642. URL: https://arxiv.org/abs/1707.06642.

293. V. Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

294. V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

295. V. N. Vapnik and A. Y. Chervonenkis. "On the uniform convergence of relative frequencies of events to their probabilities". In: *Measures of complexity*. Springer, 2015, pp. 11–30.

296. L. N. Vaserstein. "Markov processes over denumerable products of spaces, describing large systems of automata". *Problemy Peredachi Informatsii* 5:3, 1969, pp. 64–72.

297. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". *Advances in neural information processing systems* 30, 2017.

298. C. Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

299. A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. "Deep learning for computer vision: A brief review". *Computational intelligence and neuroscience* 2018, 2018.

300. M. Wang and W. Deng. "Deep face recognition: A survey". *Neurocomputing* 429, 2021, pp. 215–244.

301. W. Wang and J. Gang. "Application of convolutional neural network in natural language processing". In: *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*. IEEE. 2018, pp. 64–70.

302. Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. "Generalizing from a few examples: A survey on few-shot learning". *ACM computing surveys (csur)* 53:3, 2020, pp. 1–34.

303. J. Warner, J. Sexauer, scikit fuzzy, twmeggs, alexsavio, A. Unnikrishnan, G. Castelão, F. A. Pontes, T. Uelwer, pd2f, laurazh, F. Batista, alexbuy, W. V. den Broeck, W. Song, T. G. Badger, R. A. M. Pérez, J. F. Power, H. Mishra, G. O. Trullols, A. Hörteborn, and 99991. *JDWarner/scikit-fuzzy: Scikit-Fuzzy version 0.4.2*. Version v0.4.2. 2019. DOI: 10.5281/zenodo.3541386. URL: https://doi.org/10.5281/zenodo.3541386.

304. K. Weiss, T. M. Khoshgoftaar, and D. Wang. "A survey of transfer learning". *Journal of Big data* 3:1, 2016, pp. 1–40.

305. F. Williams, M. Trager, J. Bruna, and D. Zorin. "Neural splines: Fitting 3d surfaces with infinitely-wide neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9949–9958.

306. A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. "Deep kernel learning". In: *Artificial intelligence and statistics*. PMLR. 2016, pp. 370–378.

307. H. Xiao, K. Rasul, and R. Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". *CoRR* abs/1708.07747, 2017.

308. T. Xiao, M. Singh, E. Mintun, T. Darrell, P. Dollár, and R. Girshick. "Early convolutions help transformers see better". *Advances in Neural Information Processing Systems* 34, 2021, pp. 30392–30400.

309. R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. "Convolutional neural networks: an overview and application in radiology". *Insights into imaging* 9:4, 2018, pp. 611–629.

310. R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. "Convolutional neural networks: an overview and application in radiology". *Insights into imaging* 9:4, 2018, pp. 611–629.

311. Z. Yang, M. Moczulski, M. Denil, N. De Freitas, A. Smola, L. Song, and Z. Wang. "Deep fried convnets". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1476–1483.

312. Y. Yao, L. Rosasco, and A. Caponnetto. "On early stopping in gradient descent learning". *Constructive Approximation* 26:2, 2007, pp. 289–315.

313. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

314. D. Yu, H. Wang, P. Chen, and Z. Wei. "Mixed pooling for convolutional neural networks". In: *International conference on rough sets and knowledge technology*. Springer. 2014, pp. 364–375.

315. X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. "S4l: Self-supervised semi-supervised learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1476–1485.

316. D. Zhang, J. Han, G. Cheng, and M.-H. Yang. "Weakly supervised object localization and detection: A survey". *IEEE transactions on pattern analysis and machine intelligence* 44:9, 2021, pp. 5866–5885.

317. Q. Zhang, L. T. Yang, Z. Chen, and P. Li. "A survey on deep learning for big data". *Information Fusion* 42, 2018, pp. 146–157. ISSN: 1566-2535.

318. X. Zhang, Z. Li, C. Change Loy, and D. Lin. "Polynet: A pursuit of structural diversity in very deep networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 718–726.

319. F. Zhao, F. Lin, and H. S. Seah. "Binary SIPPER plankton image classification using random subspace". *Neurocomputing* 73:10, 2010. Subspace Learning / Selected papers from the European Symposium on Time Series Prediction, pp. 1853–1860. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2009.12.033. URL: https://www.sciencedirect.com/science/article/pii/S0925231210001451.

320. Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu. "Object detection with deep learning: A review". *IEEE transactions on neural networks and learning systems* 30:11, 2019, pp. 3212–3232.

321. H. Zheng, R. Wang, Z. Yu, N. Wang, Z. Gu, and B. Zheng. "Automatic plankton image classification combining multiple view features via multiple kernel learning". *BMC Bioinformatics* 18:16, 2017, p. 570. ISSN: 1471-2105. DOI: 10.1186/s12859-017-1954-8. URL: https://doi.org/10.1186/s12859-017-1954-8.

322. Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou. "Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids". *IEEE Transactions on Industrial Informatics* 14:4, 2017, pp. 1606–1615.

323. Y.-T. Zhou and R. Chellappa. "Computation of optical flow using a neural network." In: *ICNN*. 1988, pp. 71–78.

324. Z.-H. Zhou. "A brief introduction to weakly supervised learning". *National science review* 5:1, 2018, pp. 44–53.

325. F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. "A Comprehensive Survey on Transfer Learning". *Proceedings of the Institute of Radio Engineers* 109:1, 2021, pp. 43–76.

326. T. G. Zimmerman and B. A. Smith. "Lensless stereo microscopic imaging". In: *ACM SIGGRAPH 2007 emerging technologies*. 2007, 15–es.