POLITECNICO DI TORINO

Master Degree Course in Computer Engineering

Master Degree Thesis

Deep Convolutional Neural Networks for Document Classification

Supervisor
prof. Bartolomeo Montrucchio

Candidate
Fabio Ellena
student ID: 231614

Internship Tutor
Docapost BPO
Fabien Aïli

April 2018

Abstract

In the last decades the demand for always faster and more optimized business processing management solutions has continuously increased, boosting the growth of completely digital solutions. This, together with the standardization of documents and procedures, allowed to reach excellent throughputs. However, in the document processing system field new challenges are coming regarding the automation of whole procedures. Standardization allowed to split complex procedures into simple and repetitive tasks, while digitalization allowed to collect huge quantities of documents. Many companies are driving the effort to hit the autonomous document processing in upcoming years, that roughly means to build a system which is able to mimic the human operator in all the operations that involve the processing of a document.

In this report, we present LEIA, a software solution that aims to reduce the document processing time. Our approach is to automate the simple and repetitive processes. LEIA uses the ultimate findings in the Artificial Intelligence field and applies them to the whole business process solution. We will describe automatic annotation and classification study on documents, the LEIA classifier design, and test on real applications, where we reach almost human performance in real time.

Acknowledgements

At the end of my internship experience, I really would like to thank my manager Fabien Aïli for giving me, in cooperation with EURECOM and Politecnico di Torino, this amazing opportunity. I would like to thank my tutor Marius Mézerette for all the patience, the time and the experience he put to guide me in my work.
I would like to thank the supervisors from Politecnico di Torino, Bartolomeo Montrucchio for his supervision and very helpful tips.
It was a pleasure to work with the Innovation team at Docapost: Emmanuel, Sébastien, Cyril, Amélie. Thanks for all your support and the work done together.

Contents

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
HDD	**H**ard **D**isk **D**rive
LSA	**L**atent **S**emantic **A**nalysis
OCR	**O**ptical **C**haracter **R**ecognition
OS	**O**perating **S**ystem
PCA	**P**rincipal **C**omponent **A**nalysis
RAM	**R**andom **A**ccess **M**emory
ReLU	**Re**ctified **L**inear **U**nit
SGD	**S**tochastic **G**radient **D**escent
SSD	**S**olid **S**tate **D**rive
SVD	**S**ingular **V**alue **D**decomposition
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

This report was written during a six-month internship in the Innovation Team team of Docapost. All the presented material and results are the outcome of the work done by Fabio Ellena (the writer) during the internship, in collaboration with the other team members. All the work presented in this report was made by Fabio Ellena, except chapter 9. The theoretical chapter 7 uses as references [Goodfellow et al., 2016][Li et al.,][Lecun et al., 1998].

1.1 Docapost and Innovation Team

Docapost is the digital branch of the group "La Poste", providing IT services from 2007. Docapost provides its main services in the following areas:

- Business process management
- Document management
- Mobile services
- Digital transformation

At Docapost the Innovation Team is composed of about ten people. Its role is to enrich the existing offers by proposing technological innovations and uses. The diversity of the profiles and an agile mode of operation allow the group to be able to take charge of projects from their creations (Design and graphic charter, the establishment of the specifications), to the deployment in production. I joined the Innovation Team for a 6 month internship as a Machine Learning Engineer and I worked mainly on the development of the classifier module of LEIA.

1.2 Contents of the thesis

In Chapter 2 we start presenting an overview of the problems related to the document processing domain. We continue with a detailled description of the current state of the art concerning the document classification. We describe the LEIA program of Docapost and more in detail the LEIA Classifier, the main topic of this report. Ultimately the goals of the project are listed.

Chapter 3 describes the hardware and software supplies used for this work. We continue with the description of the experimental protocols used during this work and the methodologies to analyze and optimize the results.

In chapter 4 we present in a detailed way all the datasets that we used in this work, with the objective to make clear to the reader the kind of data involved and their peculiarities. We describe each dataset briefly, and we show their unique peculiarities. Along with the dataset description, the chapter also treats the common problems that occur in the definition of a dataset and describes the possible effects on the classification.

Chapter 5 describes the annotation process for the unlabeled datasets that we use in this work. We first analyze the main manual solution to the annotation problem. Then we present a solution that automatically annotates the dataset with a minor human effort. We show the results of the annotation both regarding accuracy and required time.

In chapter 6 we firstly show the preprocessing operations that we apply to the documents before the classification. Then we treat the topic of data augmentation, with an in-depth analysis of the type of augmentation that fits better the document datasets.

Chapter 7 covers the theory behind modern Neural Networks. We start with a logical and mathematical description of what a Neural Network and we show the role of activation and loss functions. Then we describe the training process and the idea behind the gradient-based learning and the back-propagation. We conclude with the description of the transfer learning.

In chapter 8 we start with a description of the characterizing blocks of CNNs, which are used for all the image classification tasks of this work. We continue with the description of the MobileNet and DenseNet architectures and the training protocol we use. To conclude we show the classification results with internal and external datasets.

Chapter 9 shows the whole pipeline required to classify a document using the text it contains. We briefly describe the technologies to extract the text and then we present the building blocks of the text classifier we used. In conclusion, we present the results on our datasets.

In chapter 10 we explore the techniques that allow creating a model ensemble, or in other words, a model composed of many models. We conclude presenting the results achieved by the ensemble on our datasets.

Chapter 11 presents the problem of the interpretability of deep learning models for image classification. We continue with an example, where we apply an interpretation technique to our model, and we show the insights that it can provide to optimize various aspects of the training pipeline.

In chapter 12 we explore the different deployment scenarios, tools, and libraries. Then we show a real demonstration of the steps needed to deploy a model in a mobile application. We conclude showing the performances of different models and with an analysis of the insights that a mobile demonstration can provide to the model definition.

We conclude our thesis with chapter 13, where we compare our initial objectives with the obtained results. We reached almost human performances in both accuracy and time, as demonstrated in chapters 8 and 9. In this chapter we also discuss the improvement that we plan for the future of LEIA classifier, explaining our perspectives after our study, implementation, and testing of the system.

Chapter 2

Document processing systems

This chapter presents an overview of the problems related to the document processing domain. Then, we will focus on describing the current state of the art, with the work already done related to this problem. We will try to highlight the similarities and dissimilarities of our approach respect to the ones used in current document processing applications. In conclusion, we describe the LEIA program of Docapost and more in detail the LEIA Classifier, the main topic of this report. Ultimately the goals of the project are listed.

2.1 Overview

Document processing involves all the operations of a business process that can be performed on paper-based and electronic documents (e.g., scanned image of a document). These operations include tasks that go from the acquisition to the conversion, classification, and information extraction of the document. Such systems are often part of more complex business processes. Today, many companies decide to outsource their business process to specific companies to gain flexibility, cost reduction, performance boost.

In the last decades, two patterns changed the business process management and his subfields such as document processing: standardization and digitization. Business process standardization is the procedure of establishing a "best-practice" of how to complete a process and making sure that the entire group follows it. Standardization is often a part of a more significant initiative, such as Business Process Management. The main benefit of standardizing processes is higher productivity respect to non-standardized processes.

Standardization involves finding the best way of carrying out a process and then making sure that the entire organization is aware of the best practice and follows

it. In the domain of document processing, standardization is often applied directly at the source. For example, when a company has control over the documents, the most simple and effective solution is to unify the documents' layout, leading to simplified processing. This simplification is what happens in some cases with the public administration, where most documents share a standard layout. At the same time, the public administration is a negative example in all the situations where standardization is not implemented.

In a more general case, where the company has no control over documents, it is still possible to standardize the acquisition methodologies by imposing different constraints (e.g., only scanned documents).

Digitization is the process that allows for the conversion of physical data like a document into a digital format. Digitization is of vital importance for all the following operations that involve documents like data processing, storage, and transmission.

Standardization and digitization are the starting point of RPA (Robotic Process Automation). Many companies are driving the effort to hit the autonomous document processing in upcoming years, that roughly means to build a system which can replicate the actions of a human operator interacting with a user interface. RPA solutions start from the execution of data entry to the implementation of a full end-to-end business process. In a first step, autonomous document processing systems will play the role of virtual assistants for human operators. The idea is to ease and boost the job of the operator by providing him suggestions.

2.1.1 LEIA

LEIA is an enterprise program of Docapost that aims to automate the document processing operations of current business processes. The LEIA project builds on the idea to automate the simple and repetitive processes by using the latest findings of Artifical Intelligence. The LEIA program is arranged into the following modules:

- CAPTURE: Nowadays people start using their smartphones to take a picture of a requested document and then send it. This operation creates a new range of problems, explained in 4.1.3. To alleviate these problems, LEIA delivers to the clients a mobile application that can be used to take an optimized picture. Taking a better picture is an incentive for the user because he will be sure that the document processing system will correctly process his request thanks to the high-grade picture quality.

- ICR: Almost the entirety of documents is composed of words. The capability of extracting text correctly in complicated documents (e.g., handwritten documents) can unlock a whole new range of services. The first use of the extracted text is classification, in fact, it is possible to classify a document

exclusively from the text that it contains. More complex uses of the extracted text involve the EXTRACT and ROBOT modules.

- CLASSIF: Once image and text of a document are available, it is possible to perform classification. The principal applications of a document classifier are the automatic distribution and archiving of documents, image indexing, and document retrieval.
- EXTRACT: Once the text of a document is extracted, it is possible to associate the text with some fields of interest. We only need to define the fields of interest, create a dataset of annotated documents, and train a NER (Named Entity Recognition) engine.
- ROBOT: Document processing systems often generate new documents following precise business rules that can be specified programmatically. Once the rules are defined, and the data are available, this phase can be automated too.

Each module of LEIA is independent and can operate without the other modules. Interoperation of all the modules is required when a whole document processing pipeline is required. For example, the CLASSIF module can work without the two preceding modules because it only needs the image of the document to work. Nonetheless, to achieve better results, the two preceding modules are needed: the CAPTURE module improves the image classification because it improves the image acquisition, while the text extracted with the OCR module can be used to perform a text classification. The two classifier combined to offer more accurate prediction respect to the single output of the image classifier.

2.2 State of the art

Here we describe the state of the art respect to document classification, Robotic process automation tools will not be analyzed since they are not part of this report. Document classification can be performed using directly the image or the text extracted from the document. Image-based methods base the prediction on the local and global features of the document, while text-based techniques rely on the semantics of the text.

2.2.1 Image classification

Image classification is the task of taking an input image and outputting a category or a probability of categories that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly. When we see an image or just when we look at the world around us, most of the time we are able to immediately

characterize the scene and give each object a label, all without even consciously noticing. These ability of being able to quickly recognize patterns and adapt to different image environments is not present with machines. In document classification this approach is preferred because it works directly on digital images, thus avoiding long procedures like the extraction of text. It is important to remember that not all documents can be classified using the text exclusively and that nowadays the extraction of text from handwritten documents is not ready for a real-world deployment.

What we want the classifier to do is to be able to differentiate between all the images it is given and figure out the unique features that make an ID card an ID card or that make a RIB a RIB. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a document, we can classify it as such if the picture has identifiable features, without the need of reading the document. Similarly, the computer can perform image classification by extracting these features from the image and then combining them to make a correct classification. The principal methods for image-based classification can be grouped into three categories:

- Template similarity
- Image Descriptors
- Convolutional Neural Networks

Template similarity

This class of methods exploit the layout/structural similarity of the document images to perform the classification. A first example is the work of [Kochi and Saitoh, 1999], where a person defines a template for each form, and then the identification is made by template matching. In general, the principal difficulties with template image matching methodologies arise from the complicated process of choosing templates, writing rules and validating the system.

Such systems perform well when the document format and the acquisition procedure are well specified and constrained. On the other hand, this system does not work anymore when classes become more complex, presenting multiple templates, and the definition of category becomes more uncertain. Nowadays, image descriptors based systems that do not need a person to specify a feature for each class substituted layout based systems.

Image Descriptors

To solve the problems of template matching techniques, image descriptors based methods have been developed. The document structure is not anymore specified by a person but learned from a model.

A large part of the modern approaches follow the BOW (Bag Of Word) approach, composed of a 4 step pipeline [Kumar and Doermann, 2013b]:

1. Extraction of local image features: Local feature points, such as SIFT [Lowe, 1999], are widely used due to their description capabilities.

2. Encoding of local image descriptors: BOW was originally used to encode the feature point's distribution in a global image representation. Fisher vectors and VLAD later showed improvement over the BOW [Jégou et al., 2012] [Perronnin et al., 2010].

3. Pooling of encoded descriptors into a global image representation: Spatial and feature space pooling techniques have shown to provide improvements [Lazebnik et al., 2006].

4. Training and classification of global image descriptors: Classifiers such as linear SVM (Support Vector Machines) are widely accepted as the reference regarding classification performance.

In 2014, [Kumar et al., 2014] proposed a method for document classification that relies on a codebook of SURF descriptors of the document images, and then it uses the codebook for the classification. Algorithms based on image descriptors have shown great result and robustness respect to little variations in images. Nonetheless, they show some difficulties when the documents present a high intra-class variance and a low inter-class variance. To sum up, it is difficult to come up with handcrafted feature extractors that are specific to document image classification.

Convolutional Neural Networks

Even though image descriptors have shown great potential, they are still based on handcrafted feature extractors that do not adapt to the data. This defect is the direct consequence of the fact thatthe algorithm that extracts the features is always the same, regardless of the data provided.

CNN (Convolutional Neural Networks) have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural networks grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition, dropping the classification error record from 26 to 15, an astounding improvement at the time [Krizhevsky et al., 2012].

This work builds from recent works regarding CNN usage for document classification [Kang et al., 2014], [Harley et al., 2015], [Afzal et al., 2017]. The first work where CNNs have been used in the document classification task is [Kang et al., 2014]. In this work, the author motivates the use of CNN for classifying unconstrained documents with the fact that CNNs automatically learn the hierarchical

layout features of a class. This approach managed to outperform methods based on structural similarity of documents images from the Tobacco-3482 dataset4.2.2.

One year later, [Harley et al., 2015] introduced the RVL-CDIP dataset which provides a large-scale dataset for document classification and allows for training deep CNNs from scratch, obtaining better results than hand-crafted alternatives. Experiments also showed that given sufficient training data, enforcing region-specific feature-learning is unnecessary.

More recently [Afzal et al., 2017] showed a great improvement in the accuracy by applying deeper models and transfer learning from the domain of real-world images to the domain of document images, thus making it possible to use deep CNN architectures even with limited training data.

To conclude, CNN generates highly effective compact description, largely outperforming earlier SIFT-based encoding schemes from the classification performance and run-time point of view [Sicre et al., 2017]. The greatest disadvantage of CNN is that the training process is time-consuming, even using last generation GPUs. Nonetheless, the improvement justifies this additional expenses. Another disadvantage of CNN is that they are difficult to inspect, thus making difficult for the developers to debug them and for persons to trust them. This aspect is treated in chapter 11.

2.2.2 Text classification

Text classification is a fundamental task in the domain of Natural Language Processing, with applications in domains like recommender systems, information retrieval, ranking and document classification [Pang and Lee, 2008].

Efficient and straightforward models like linear classifiers are often used as solid baselines for sentence classification problems [Joachims, 1998]. Despite their simplicity, they often obtain state of the art performances if the right features are used, like a BOW representation of the document. Recently, models based on convolutional or recurrent neural networks have become increasingly popular in this field [Kim, 2014] [Zhang et al., 2015] [Conneau et al., 2016]. All these models have shown great performances, often sacrificing prediction time in favor of slight improvements in accuracy.

Another work, FastText [Joulin et al., 2016], has shown that linear models based on efficient word representation learning [Mikolov et al., 2013a] [Levy et al., 2015] can train on large datasets in a short time while achieving performance similar to state of the art models that takes much more time for training.

2.3 Problem analysis and solution proposal

The LEIA system, being a solution thought to be delivered to multiple clients and use cases, has to take care of the many different possible scenarios.

Nowadays we live in the era of big data, but the reality is that most of the data is collected in such a way that its exploitation is difficult. In a machine learning scenario, high quality data is characterized by the presence of fine grained labels. Unfortunately, the majority of data is collected without associated labels, or with too general ones to be useful. In this optic, one of our missions is to collaborate actively with clients and propose solutions that makes the big data lakes useful again. Therefore, the first part of the project is the definition of an annotation pipeline that accelarates the annotation of large datasets. The annotation process is composed of different steps that exploits the characterizing properties of documents to first cluster them and then annotate them.

Once an annotated dataset is available, we can proceed with the document classification, which uses both the image and the text of the documents. Image classification allows classifying documents analyzing their layout, while text classification operates on the semantic level.

We will not study any image classification method based on image descriptors, whereas we intend to use more modern approaches based on CNN. As we saw in the scientific literature, CNN are capable of modeling complex documents with an unconstrained structure without requiring a manual tuning of the model for making it work with different datasets.

For what concerns the image classification, the main problems to be solved are related to the lack of documents due to privacy issues and the capture bias 4.1.3 present in our training datasets. Our solution will be based on a model able to learn complex datasets with a limited number of documents and will learn robust representation that will be able to transfer well on documents captured with different taking conditions.

Regarding text classification, we will use recent techniques such as FastText that have shown good performances both in prediction accuracy and time. The text will be extracted using Tesseract [Smith, 2007], which is a common tool for OCR and is used every day in a multitude of applications. After the definition of the classification model, we define an ensemble model that increases the prediction accuracy by combining the two base models.

A third part will be the deploy of the model in a mobile application for demonstration purposes of the capability of the system in a constrained environment.

Summing up, our solution proposal point to have those five main features:

- Fast annotation pipeline: For customers with non annotated datasets and without the expertise of this domain, it is essential to provide a solution that accelerates the annotation of the dataset to then start in a short time the definition of the model.

- Minimum human involvement: Once a model architecture is defined, it should be easy to adapt it to different datasets. This means that the model should adapt to the data, instead of relying on hand-engineered features, which are

not directly transferable to new types of documents.

- Reach human performances: We want to create a system that is ideally able to substitute a human operator in classifying documents. A client is not willing to invest money, time and persons in changing a working system if the gain is not significant. In this optic, a system that is subject to failure is not ready to be used in a production scenario.
- Non-intrusive approach: The classifier should be able to work with the current documents submitted to a document processing system, without depending on specific format or constraints that are not currently in place. This also facilitates the integration of the system in already existing applications, with reduced costs of integration for the client.
- Real-time prediction: The model should be able to reach human performances not only regarding the accuracy but also the time. This requirement is even more important in the first phase, where models are meant to assist the human operator.

As we will present in this thesis, we reached all the points. Having seen an overview of the actual state of the art on document classification, next chapter will describe in a detailed way the datasets used for the training.

Chapter 3

Hardware environment

This chapter contains a detailed description of all hardware and software supplies we used during the internship. In particular, we describe the machine that we used for development, evaluation, and testing. Moreover, in this chapter, we the experimental protocols used during this work and the methodologies to analyze and optimize the results.

3.1 Hardware and software environment

What follows is a description of the machine we used, from both the hardware and software point of view.

3.1.1 The machine

All our work has been done on a machine of the Innovation Team at Docapost, called YODA. This machine is comparable to a high-end machine for video rendering/gaming applications. The machine is composed of:

1. HDD: 12TB
2. SSD: 512 GB
3. RAM: 64 GB
4. CPU: Intel® Xeon® E5-1620 v4 3.50 GHz
5. GPU: NVIDIA GTX 1080 Ti: 11.3 TFLOPs, 11GB VRAM

Another machine we used is the iPad Air, iPhone 5 and iPhone 6 for testing the mobile applications.

3.1.2 Software environment

During the internship, YODA has been the only suitable machine for the development of deep learning models and has been used by three members of the Innovation Team. Apart from the physical access, the machine can be accessed via ssh or using the Jupyter Hub interface. While the ssh connection is suitable for maintenance tasks, the whole development is performed using the Jupyter Hub Interface because it provides a more comfortable interface.

3.1.3 Scientific stack

The OS is Ubuntu, the Linux Operating System. The major components of the project are Python and Jupyter. Python is a general-purpose programming language, while Jupyter is a server-client application that allows editing and running notebook documents via a web browser. Jupyter Notebooks are documents that contain both code (e.g., python) and rich text elements (paragraph, equations, figures, links). Notebook documents are powerful because they are bot human-readable documents that contain descriptions and results, as well as executable documents which can be run to perform computations.

All the work is based on Python 3.5.2, which has been chosen because it is the language of choice of the deep learning community and because it has solid scientific libraries. This allows us to experiment solutions and iterate extremely fast. It is to be noted that almost all Deep Learning and scientific libraries use python exclusively as an interface, in fact, the majority of the code is written in C, C++, and Fortran for speed. The main Python libraries used in this work are:

1. SciPy [McKinney, 2010]: A Python-based ecosystem of open-source software for mathematics, science, and engineering.

2. NumPy [van der Walt et al., 2011]: The fundamental computing library of Python. It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical routines.

3. Scikit-learn [Pedregosa et al., 2011]: A machine learning library. It contains various classification, regression, and clustering algorithms and is designed to interoperate with NumPy and SciPy.

4. Pandas [McKinney, 2010]: A data manipulation and analysis library. In particular, it offers an interface to organize and manipulate table and time series.

5. Matplotlib [Hunter, 2007]: A library to plot 2D graphics.

3.1.4 Deep learning stack

Deep learning libraries are particular and merit their section. Even though most of them allows to perform the same set of operations, each library has its strong points. We use Keras and TensorFlow for the research part, while we use MXNet for deploying the models on the servers. For what concerns the mobile scenario, we use CoreML for iOS and Tensorflow for Android.

Tensorflow

TensorFlow is an open source software library for numerical computation that uses data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the (tensors) transferred between them. Tensorflow allows deploying computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

Keras

Keras is a high-level neural networks interface compatible with the TensorFlow and CNTK backends. It was developed with a focus on facilitating fast experimentation. Keras was chosen because it is the "lingua franca" of deep learning, meaning that we can define a model with keras and then convert it to different frameworks that may work better in production.

MXNet

MXNet is a modern open-source deep learning framework. The strong points of MXNet are performances and scalability. MXNet is supported by principal public cloud providers, and it is the deep learning framework of choice at AWS. For us MXNet is the best framework for production because networks are fast and consume fewer resources.

CoreML

CoreML is the machine learning framework that allows iOS 11 applications to run machine learning models locally. Core ML is optimized for on-device performance, minimum memory footprint and limited power consumption.

3.2 Experimental methodology

In this section, we will explain the experimental methodology we followed to make sure that the state, setup and experimental conditions were the same on all models

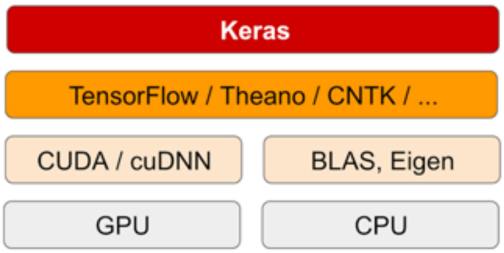

Figure 3.1: Keras software and hardware stack, adopted from [Chollet, 2017]

used. Moreover, we will present the strategy and mathematical means we used for all measures performed in our study.

3.2.1 Experiment reproducibility

Experiment reproducibility is essential because it allows tracking and replicating results, enabling more detailled analysis on the reasons of success and failure of a model, while guaranteeing a proof of work for the client. In a machine learning scenario characterized from its iterativeness, full reproducibility is achieved only if all the steps are constantly tracked. To be more precise, it is necessary "tracking the steps, dependencies between the steps, dependencies between the code and data files and all code running arguments"[Shridhar, 2017].

On the contrary, in the exploratory phase such a protocol is not necessary and it would eventually slows the whole process without clear advantages. Nonetheless, during the definition of the model we may need reproducibility for a limited time. These moments includes the situations where we have to run experiments to find good hyperparameter values for our pipeline. In such situations all things but the target hyperparameter should be fixed, therefore it is not necessary to keep track of anything.

A parameter that we should always take care in a deep learning model is the seed, which controls all the random components of the model. To fix the seed we have to set a seed for the deep learning framework, Numpy, and Python. In this way, we can be sure that all operations that use a random state will be executed under the same conditions.

3.2.2 Resources management

In deep learning experiments, we often use all the available resources, meaning that all CPU cores and the full power of the GPU should be dedicated to a single experiment. When this is not possible, we have seen that the training time does not increase when the CPU is used for other tasks that are not particularly intensive.

An important thing to note is that when the data augmentation is done on the fly, it is important to dedicate all the cores for this tasks because there is the risk to starve the GPU.

For what concerns the GPU, it should be used only by a process at a time. We tried to run multiple models at the same time, and we have seen that the GPU scheduling is not predictable, and it is possible that a model starves the other one. Moreover, we have noted that when multiple models run at the same time the performances decrease severely for both of them, making it faster to train the models sequentially.

3.2.3 Hyperparameter optimization

After the network is defined we proceed with the definition of the hyperparameters of the neural network and the data augmentation transformations.

For what concerns the data augmentation, the transformations are checked manually. The verification is achieved by generating a batch of augmented images before the training phase and modifying the hyperparameters accordingly with the augmentation desired. To do this a complete knowledge of the augmentation procedure is required.

Regarding the network hyperparameters, we usually start with the default ones. After some experiments, we can choose manually reasonable ranges for the hyperparameters. It is important to note that we can make this manual adjustment because we start with good models that usually have built-in mechanisms to prevent overfitting.

3.2.4 Measurement methodologies

During our internship, we faced many different use cases, for which was impossible to apply always the same experimental protocol, especially regarding the number of measures. This was because the datasets were very dynamic, the training of a network required much time, and in some cases having many measures was not important. For what concerns the used metric, we always used the accuracy measure, knowing that it is ill-defined for imbalanced datasets. We decided to use only the accuracy because after an initial phase we have noted that our models were performing well on all classes, also the less frequent ones, making the use of more complex evaluation protocols not essential for our work.

Regarding the dataset split, we always used a 70-20-10 split for training, evaluation, and testing. The split has been performed with the Stratified Oversampling, which enforce some guarantees over the distribution of classes in each split. This was particularly important because of the imbalanced datasets.

To conclude this section, we want to note that this work has been mainly exploratory, with the intention of creating a more refined evaluation pipeline later.

Chapter 4

Datasets

The objective of this chapter is to presents in a detailed way all the datasets that are used in this work, with the objective to make clear to the reader the kind of data involved and their peculiarities. We describe each dataset briefly, and we show their unique peculiarities. Along with the dataset description, the chapter also treats the common problems that occur in the definition of a dataset and describes the possible effects on the classification.

4.1 Dataset biases

The visual world is so complex that any finite set of samples ends up describing only some of its aspects. Moreover, in the case the samples are collected for a particular task, they will unavoidably cover just some specific visual region. Consequently, it is not surprising that pre-defined image collections, like existing computer vision datasets, present such specific bias to be easily recognizable [Torralba and Efros, 2011]. The main causes have been named in [Torralba and Efros, 2011][Søgaard et al., 2014].

4.1.1 Selection bias

Selection bias can be defined as the bias introduced by the selection of datapoints. Whenever proper randomization is not achieved, we can state that the subset selected is not representative of the real distribution.

The selection bias is usually the result of a manual selection of the datapoints. In the case of images, a human operator will always generate a bias if not forced to do otherwise because he will choose the images that best represent each class, thus excluding specific images. The primary solution to selection bias is to sample the datapoints automatically, possibly from multiple sources. This operation forces the annotator to label images that may represent corner cases of the data distribution that otherwise would be ignored.

4.1.2 Temporal bias

The temporal bias is related to changes in the datapoint distribution due to the temporal difference between the training dataset and the deploy conditions. In our case, such differences can be at the level of class frequencies, at the document level, or at the capture level. In the case of class frequencies, the problem is related to the portion of documents that are part of specific classes. When exposed to class imbalance, models learn to predict better frequent classes, and when they are uncertain, the most common class often wins. Class distribution can be useful when it represents the real document distribution, but when it changes it contributes to increasing the portion of misclassified documents. A possible solution would be to balance the dataset, but this is not always possible.

Regarding the temporal bias at the level of documents, it is the consequence of modifications of formats of documents. Whenever a new format is introduced, it is possible that the model, trained with older documents, misclassifies it. The problem stands again in the distribution of documents, but now we should consider the format, related to the emission period, of a document. The majority of documents used in a dataset are old, while only a minor part is recent. This is in complete contrast with the deploy conditions, where most documents are recent, and only a portion have an old format. Again, the solution is to rebalance the dataset, which requires a considerable effort.

After documents, also capturing conditions change over time. In this case, the solution would be to rebalance the dataset with images captured in the new conditions.

(a) Old Italian ID card (b) New Italian ID card

Figure 4.1: Temporal bias due to a change of format

4.1.3 Capture bias

The capture bias is related to how an image is acquired both regarding the used device and the capturing conditions, which includes the point of view and the

lighting conditions. Photographs appear to suffer considerably from the capture bias. The most well-known bias is that the object is almost always in the center of the images.

When we analyze capture bias in the document domain, we can describe the phenomenon even more precisely. Historically, copies of documents have been created with a copy machine. Copy machines produces exact copies of the documents, thanks to uniform lightning and the flattening of the original document. The main biases were respect to the position of the documents. A4 documents showed slight rotation and translations, while smaller documents like ID cards and passports were located in random positions of an A4 paper. Document processing systems were often designed to deal exclusively with these type of documents, where the capturing conditions were consistent.

Nowadays, with the evolution of technology and the digitization of administrative services, people start to send electronically all the documents. Dealing with digital images have brought different problems. The first one is that documents are not anymore constrained to the A4 format, this means that it is possible to receive an image entirely filled by a document like a passport. The second problem is that images are not anymore grayscale, but colored.

If the digitization of the document is performed with a scanner, there are no problems regarding the quality of the image. Nonetheless, nowadays smartphones are pervasive, and persons start to send pictures of documents. This creates a new range of problems, related to the capturing device and capturing conditions. Regarding the capturing conditions, usually, the document is put on a plain surface, like a table. This means that the image is often surrounded by a colored background, the lightning is not uniform, and shadows are present. For what concerns the capturing device, the quality is always worse than that of a scanner, and may present imbalanced colors, blurring, and noise.

Current datasets contain mostly old documents, meaning that they are composed mostly of grayscale and color documents digitized with a scanner. This distribution is in contrast with the current documents that are submitted, which leads to a worsening in classification performances.

4.1.4 Label bias

The category or label bias is the consequence of the fact that visual semantic categories are often poorly defined: similar images may be annotated with different class while, due to the in-class variability, the same class can be assigned to visually different images. Label bias is particularly evident when the annotation process is performed by a single person because the annotator uses its criteria to label ambiguous images and these annotation defects have repercussions on the whole dataset. A possible way to reduce this bias constitutes in aggregating the annotation performed by different persons, thus obtaining a more robust model. Even though

majority voting reduces the label bias, there are some cases where the bias is not personal and affects large groups of persons. To correct these biases, guidelines are often created and establish some ground truth. For this reasons guidelines should be clear and undebatable, otherwise, they could introduce even more substantial biases.

4.1.5 Negative set bias

Datasets that only includes the categories they are interested in might not work in a deployment scenario because they are not modeling the rest of the visual world. One remedy is to add negatives from other datasets. Adding negatives is fundamental especially from a deployment point of view because it allows discarding all the documents that are not part of the target categories. Without a negative category, the model would be forced to randomly pick a category as the prediction, with possibly undesired consequences.

4.2 Test datasets

4.2.1 CIFAR-10

CIFAR-10 [Krizhevsky et al.,] is a public dataset that consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The dataset categories are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

The quality of the dataset is very high: the classes are mutually exclusive, they are balanced, and the images are not ambiguous. Other relevant characteristics of the dataset are the number of images, which is big enough to run meaningful experiments, and the size of the images is little, meaning that experiments do not require expensive resources. For these reasons we use this dataset as a benchmark for the ensemble models.

4.2.2 Tobacco-3482

The Tobacco-3482 dataset [Kumar and Doermann, 2013a] is a sample of 3,482 images from the IIT CDIP Test Collection [Lewis et al., 2006], also known as the Tobacco litigation dataset. The original collection contains over seven million high-resolution images of scanned documents, collected from public records of lawsuits against American tobacco companies.

The Tobacco-3482 dataset was used in a number of related papers [Kumar and Doermann, 2013a] [Kumar et al., 2014] [Kang et al., 2014] [Harley et al., 2015] [Afzal et al., 2017]. Each image has one of ten labels. The categories are highly imbalanced, with the same distribution present in the full dataset.

The label classes in the dataset are advertisement, email, form, letter, memo, news, note, report, resume, scientific.

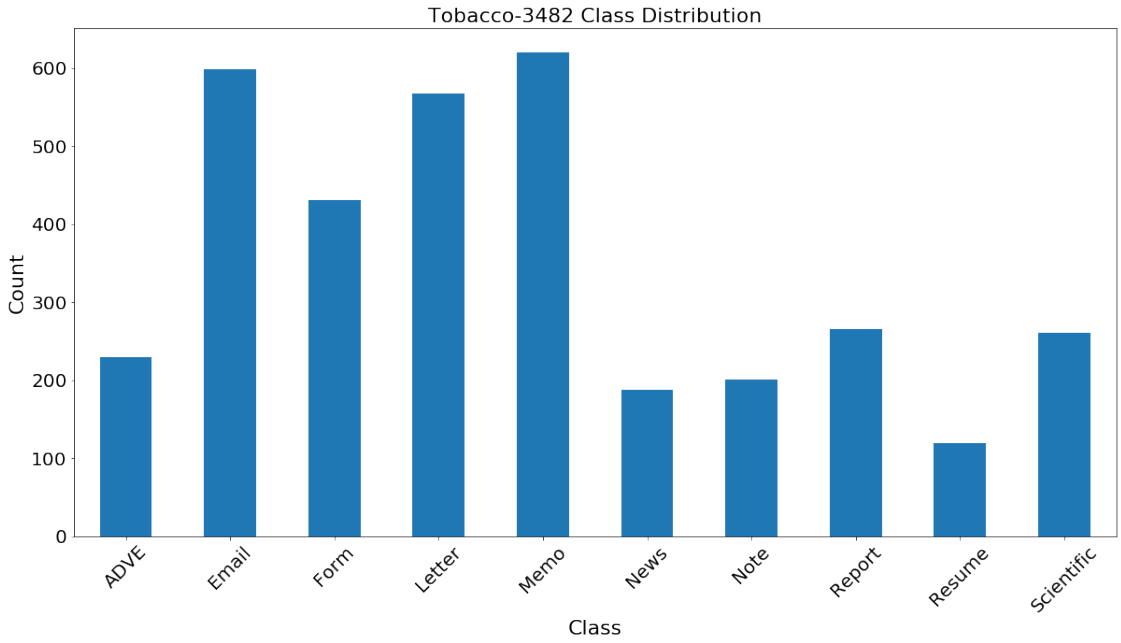

Figure 4.2: Class imbalance of the Tobacco-3482 dataset

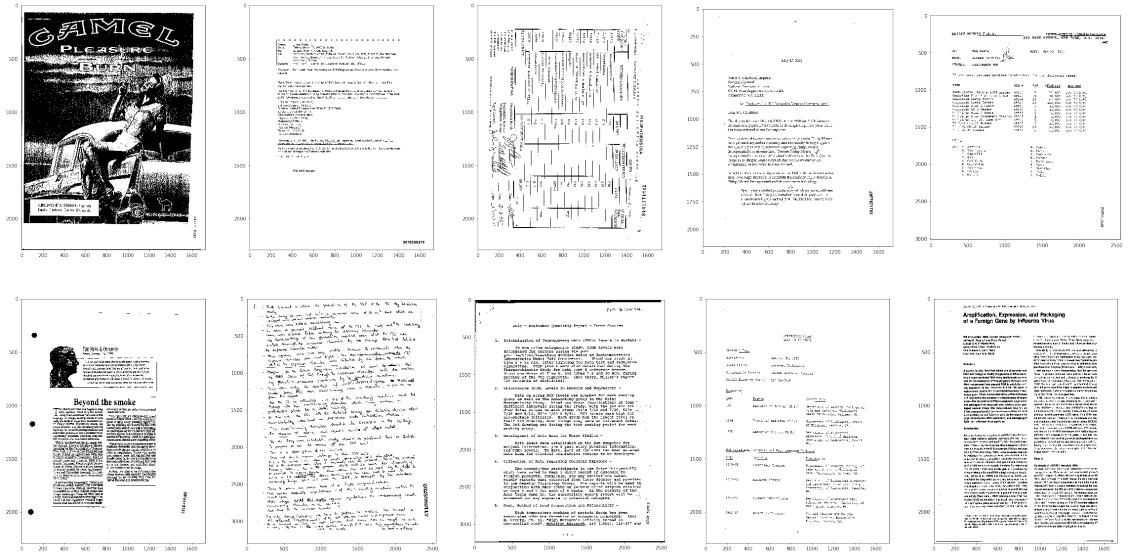

Figure 4.3: Samples of the Tobacco-3482 dataset

In this work, we use the Tobacco-3482 dataset to compare different methods for document classification. Images in categories such as Advertisement, Resume,

Report, News exhibit high variance in structure, while images of different categories such as Report and Scientific may have a similar structure. Another important characteristic is its limited size, especially in the testing scenario, where only 100 images per category are used during the training phase. All these characteristics make the Tobacco-3482 an excellent dataset to evaluate image classifiers that can generalize well with small and complex datasets.

Regarding the experiment setting, we follow the evaluation scheme that was also used in [Afzal et al., 2017] to have comparable results. We randomly create the dataset partitions with the constraints that 100 images per class are used for training while the rest are for testing. The training dataset is again split with an 80/20 ratio so that 20% of the training data are used for validation. Each experiment is repeated ten times with different dataset partitions, and the results are averaged. This is done to have a robust estimate because the dataset is small and good performances may be due to chance.

4.2.3 ADMINISTRATIVE

ADMINISTRATIVE is an internal dataset that contains the documents from a large public administration that processes a multitude of personal documents. The document categories are very extensive and vary from ID cards to pay slips, electricity factures, marriage certificates and so on. The dataset can be seen as public knowledge of administration documents. As expected from such a comprehensive dataset, some classes contain documents that vary a lot in structure. This is mostly due to the absence of standards in the public administration. At the same time, some classes are very similar.

A more serious problem is due to the class imbalance, which is way more pronounced than in the other datasets we use. In the dataset there are classes with more than 10,000 documents, while other classes have less than 10 documents. Such a high ratio is critical for the training of a model because it will tend to learn features useful to classify the more frequent classes.

AVIS D'IMPOT

The "Avis d'impot" is a document for attesting the income declared by a person This category corresponds to a set of specific documents, but due to a lack of standardization, there are different formats, which have been released from different offices at different times. Splitting the "Avis d'impot" category is useful for two main reasons. The first one is related to the business, in fact having a more fine-grained division of categories can empower the product, providing more powerful features to the client.

The second one is related to the development of the model, in fact during the training, the model focuses on large classes because they are the largest contributor

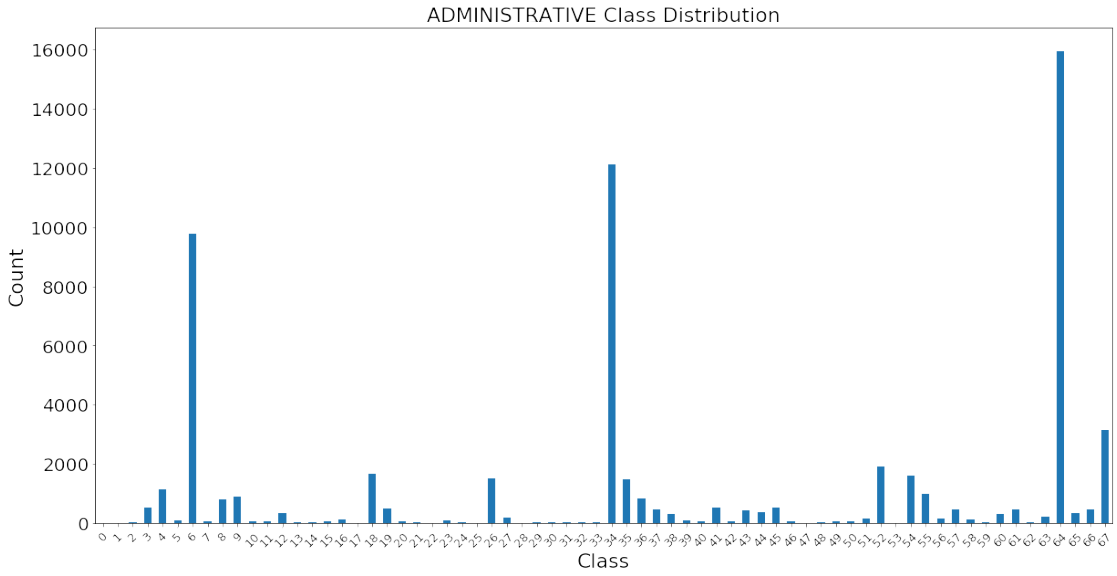

Figure 4.4: Class distribution of the Administrative dataset

to the loss. Dividing large classes allows the model to focus on all classes. Moreover, large classes are usually composed of a multitude of different documents, which the model can learn easier if divided into specific classes.

RIB

RIB (Relevé d'Identité Bancaire) is a document that describes a French bank account. It is one of the most requested documents by administration offices. Thus it is one of the largest categories of the ADMINISTRATIVE dataset. The category is essentially composed of the documents released by the banks, which are different from bank to bank. To facilitate the classification operation and for future initiatives, the RIB class has been divided into subclasses associated to the different banks.

4.2.4 ENTERPRISE

ENTERPRISE is an internal dataset that contains the documents necessary for the opening of a French activity. The documents are printed forms with a fixed structure.

The peculiarity of this dataset is the extremely low intraclass variance, given from the fact that the forms have a static structure. Given a class, the form is always the same, and the only differences are the information written in the fields. The second peculiarity is the low interclass variance. This is given from the fact

that the forms share a common template, with few elements differenciating the different classes.

As a result, this is an exceptional dataset because once the model learns how to differentiate the classes, then it is easy to verify if an item is part of a class or not.

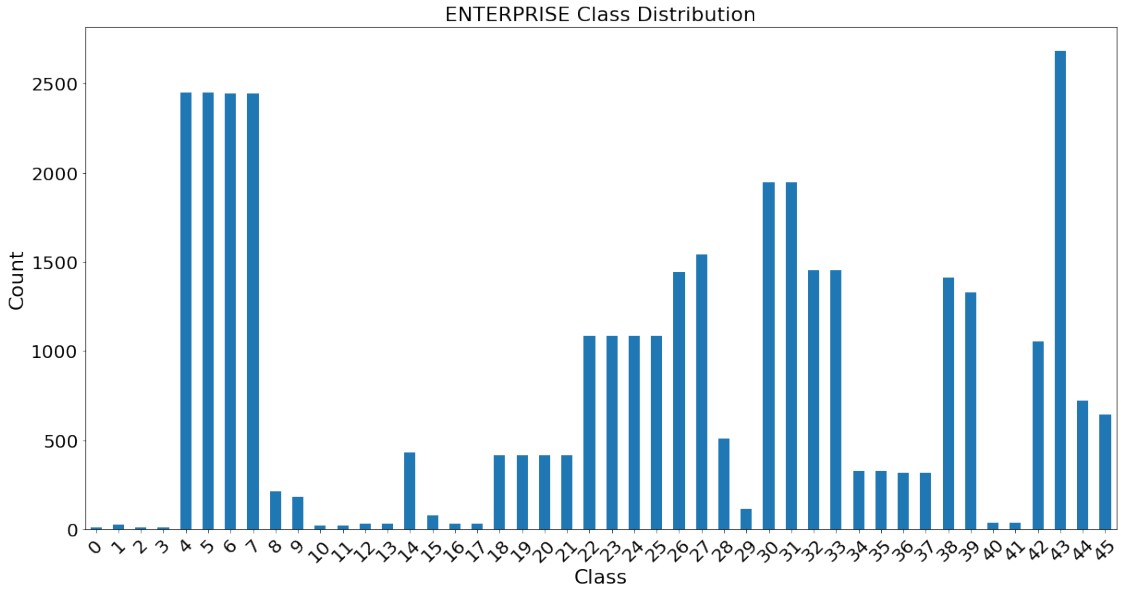

Figure 4.5: Class imbalance of the ENTERPRISE dataset

4.3 Evaluation

A major consequence of the biases in datasets is that the evaluation of a model should be reconsidered. If our data was selected in a biased way, does significance over data points make much sense? In practice, no.

If it is possible to evaluate Capture bias by performing different transformations on the evaluation images, it is impossible to correct the other biases once the dataset is defined. Thus the most practical and straightforward way is to correct the problem at the source, creating an unbiased dataset, which resembles the deployment scenario.

Chapter 5

Dataset annotation

This chapter describes the annotation process for the unlabeled datasets that we use in this work. We first analyze the main manual solution to the annotation problem. Then we present a solution that automatically annotates the dataset with a minor human effort. We show the results of the annotation both regarding accuracy and required time.

5.1 Problem definition

It is well known that in all machine learning projects, the single most time-consuming operation is represented by the annotation of the dataset. The annotation process is time-consuming because a person needs to go through every document and label it manually. The annotation is one of the processes that can not be delegated to a machine because it consists in the definition of an oracle.

The main difficulties related to the annotation are the same that makes a problem difficult for a model:

- Number of features: A large number of features is indeed a problem for both a human annotator and a machine. If a person is simply not able to look at the same time to tens of features, a model has no problem in finding relations between them. At the same time models do not work well with too many features, and the problem is often addressed as the curse of dimensionality. When the data is an image, features have to be extracted and persons have shown to be extremely good at this task. On the other side, the feature extraction is one of the most complex tasks in computer vision. It is interesting to note that when the right features are extracted, also a linear model is sufficient for a correct classification.

- Number of classes: Similarly with the number of features, a human finds very difficult to label a dataset with too many classes. One problem linked to the

number of labels is the creation of an annotation interface, which becomes more complex and slow with the increase of classes.

- Inter-class variance: A low inter-class variance means that different categories show similar documents. To perform a correct annotation, a human person has to spend some extra time to identify the distinguishing feature.

- Intra-class variance: A high intra-class variance means that entirely different documents are part of the same category. For both a human annotator and a model, it becomes more difficult to associate many different features to the same category. Humans, in this case, have an advantage because often the categories are defined semantically, and often the annotator can grasp the semantic features that link many different documents to the same category.

5.1.1 Time estimation

As mentioned above, the main problem of the annotation process is that it requires much time. This represents an important problem from both the economic and managerial point of view. Economically, a lengthy process requires more money, while from the project management point of view the annotation process represents a bottleneck because, without an annotated dataset, the development of the model cannot proceed.

This makes essential to be able to estimate the time required for an annotation because in this way the resources can be allocated more efficiently and the cost can be estimated correctly. As an example, we used the ADMINISTRATIVE dataset, which consists of 60,000 images and it is considered difficult to annotate because it shows most of the characteristics described in 5.1. The definition of the difficulty is essential because it has a multiplicative impact on the annotation time.

Some documents are easy to annotate, while others are more complex. We calculated that most of the document can be annotated on average in 5 seconds. At this moment we can calculate the time required for the annotation of the ADMINISTRATIVE dataset:

$$\frac{60{,}000 \times 5}{3{,}600} = 83 \text{ person hours}$$

If a person spends 7 hours a day annotating the dataset, it will take 12 working days to annotate the dataset. This simply means that if a client wants a solution and the dataset is not annotated, it may take weeks for the annotation if only one person is employed for the annotation. This effort should be made for each new dataset, which means large costs and working hours spent on a trivial task.

5.1.2 Solutions

Reducing annotation time is essential for both industry and research communities, and large effort has been made to reduce the required time.To lower costs and time there are different techniques.

Crowdsourcing

The whole annotation process can be delegated to crowdsourcing internet marketplaces like Amazon Mechanical Turk. These services are enabling individuals and businesses to coordinate the use of human persons to perform tasks that computers are currently unable to do.

The main advantages of these solutions are the following ones:

- Time: The time to annotate the whole dataset can be reduced simply using more annotators.
- Cost: Crowdsourcing marketplaces offer costs per annotation that are way lower than the average salary of a developer.
- Bias: This is linked to the fact that many annotators are used for the annotation. This allows to reduce the label bias 4.1.4 of the dataset because many persons with different biases are used for the annotation task. This is exactly like using an ensemble of models for the classification.

Among the disadvantages we have:

1. Sensible data: When the dataset contains sensible data that the client is not willing to share, the crowdsourcing approach is not possible because we have no control over the annotators.
2. Quality: Whenever we outsource the annotation we take the risk that some annotations may be wrong. With crowdsource annotation services, persons are paid per annotation, which means that quality is often traded for quantity. Performing checks on the annotations is possible but it is a slow process with.
3. No Insights: Most of the times the annotation process is necessary to make the team learn useful insights about the data that later can be used to define the model. If the annotation process is delegated, this learning process is lost.

Ad hoc interface

Creating an optimized annotation interface can speed up the whole annotation time. The main techniques used to improve the annotation speed are:

- Automatic queueing: this is the minimum requirement of an annotation interface. Passing instantly from an image to the next one saves a considerable amount of time respect to opening each image one at a time.

- Shortcuts: they allow to speed up the annotation, but they can be used only when the number of classes is limited.

- Autocompletion: this is necessary in cases where the number of classes is high, or when the categories names are long. A good autocompletion can save seconds for each annotation.

Automation

Not all documents are difficult to annotate, in fact, most of them are very similar. In an annotation system without any automation mechanism, the annotator would lose time annotating very similar images and in cases where the intra-class variance is low, this results in a huge waste of time. All the systems that involve some automation may result in a small number of the wrong annotation, but this is often a trade-off that persons are willing to make because automation allows cutting the annotation time heavily. Automated annotation systems are usually composed of multiple phases. In the first phase, the human annotator annotates a fraction of the dataset. This small dataset is used to train a model that then performs the predictions for the remaining documents. At this point, it is possible to use the predictions as suggestions for the human annotator, in this way an interface can give the annotator the possibility to use the suggested label or to modify the annotation. This system can be seen as a smart interface that allows cutting annotation costs by showing suggestions to the annotator.

Another more powerful approach is to accept the predictions above a given confidence threshold automatically. These techniques allow reducing by a big factor the dataset to be annotated manually. We follow this second approach because our primary interest is the reduction of the annotation time.

5.2 Annotation by clustering

As seen in 5.1, automating the annotation task is difficult because the labels are not available. Our approach to speed up the annotation process is to reduce the number of documents to be annotated. We achieve this by clustering the documents of the same class together, thus avoiding to annotate the documents singularly. For the ADMINISTRATIVE dataset the final goal is to reduce the number of documents to annotate from 60,000 to 68, where 68 is the number of categories of the dataset.

5.2.1 Feature extraction

With both text and images we cannot use the raw features for the clustering phase because they do not summarize the content of the documents. With images, we have millions of pixels that taken singularly do not show any meaningful pattern. With text we have a the raw features that are the character, but they need to be organized and cleaned to communicate any useful information.

For this reason it is necessary to extract meaningful features from the documents, with which we can perform the clustering.

Images

A color image has a number of features equal to $3 \times$ number of pixels. For reference, an A4 paper scanned with a 300 DPI resolution generates an image of 2480×3508 pixels, which corresponds to 26 millions of features. In the Deep learning community, two standard sizes for images are 224×224 and 299×299. With the goal of reducing the image size, while preserving the image details, we choose the larger size. At this point an image counts

$$3 \times 299 \times 299 = 268{,}203 \text{ features}$$

At this points, it is clear that it is impossible to use the raw features of an image because of their dimensionality.

The second reason for which we cannot use the raw features is the fact that the majority of clustering algorithms are built on the hypothesis that features are statistically independent. With images, this condition is not met because nearly located pixels show high correlation. This represents an even bigger problem that the mere large number of features. To facilitate the clustering operation, it is necessary to extract higher level features from the images, which will be used later as the input of the clustering algorithm.nThe ideal features should be high level and rotation, scale, and position invariant, in this way images which are an affine transformation of each other would have almost identical descriptors.

In the computer vision domain, there exists many hands engineered feature extraction methods (e.g., SIFT, HOG, SURF). The main problem of these feature extractors is that the extracted features are complex to use for clustering.

An alternative approach makes use of CNN for extracting features. Recent works demonstrated that the activation vectors near the top of a deep CNN could be used as feature vectors in a variety of tasks [Razavian et al., 2014]. Even more interesting, it has been shown that features learned on the ImageNet dataset transfer well on document images [Harley et al., 2015], meaning that we can use a pretrained model as a feature extractor. Strong of these previous evidence, we use a Xception model [Chollet, 2016] pretrained on ImageNet. The choice of the Xception network was driven by its great performance on ImageNet and its use of a larger input image, which could be beneficial with documents that shows an high density of information.

To use the model as a feature extractor, we remove the last fully connected layer, in this way the output of the neural network is the penultimate feature vector. Before doing the feature extraction, we resize our images to 299×299 pixels, and then we preprocess them using the same preprocessing operations used originally for the Xception model. The preprocessing operation includes average channel subtraction and normalization.

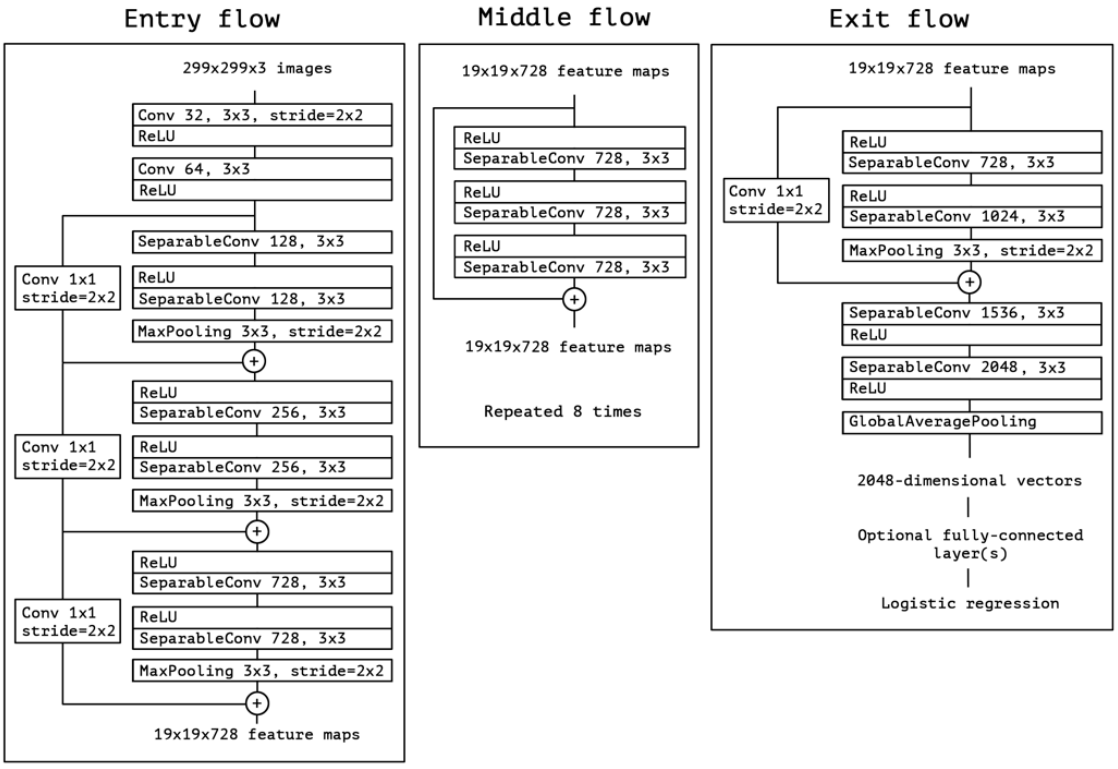

Figure 5.1: Xception Architecture, adopted from [Chollet, 2016]

At this point, we compute the feed forward phase of the model, using as input our preprocessed images. The output of the model is the 2,048 dimensional feature vector associated to each image.

Text

When documents of the same category have a completely different image, it is not possible to use image-specific features for the clustering phase. Nonetheless, it is plausible that documents within the same semantic category contain similar words. Our goal is to find a representation of the documents' text that allows to cluster similar documents.

The first step is the extraction of the text from the documents. We do this using the OCR module described in 9.0.1.

The second step is the tokenization and normalization of the words. Here we remove all non-alphabetical characters; then we remove all the accents and we lowercase the letters. We do not perform stemming operations.

After these operations, we proceed to the construction of the BOW (Bag of words) features. The BOW representation constitutes a term-document matrix where each row is a single document, and each column is a single word. Each cell of the matrix contains the number of repetitions of a token in a document.

The conventional approach is to construct a dictionary representation of the vocabulary of the training set, and then use it to map words to indices. The main problem with this approach is that the dictionary ends up having a huge number of words when large datasets are used[Ganchev and Dredze, 2008]. On the contrary, if the vocabulary is kept fixed and not increased with a growing training set, the model may miss some words that are useful. Instead of maintaining a dictionary, a feature vectorizer that uses the hashing trick can build a vector of with a fixed dimension by applying a hash function H to the features. The resulting hash modulo the dimension of the vector gives us the index in the vector that should be updated. In [Ganchev and Dredze, 2008] it has been shown that text classifiers performances do not decrease when the hashing trick is used with tens of thousands of columns in the output vectors. In our application we start by using 10,000 columns, knowing that later will be reduced with the dimensionality reduction.

Once the term-document matrix is constructed, we apply the tf-idf (term frequency–inverse document frequency) transformation, which reflects how important a word is to a document in a collection. The tf-idf value is proportional to the frequency of the word in the document and is normalized by the frequency of the word in the collection. This helps to adjust for the fact that some words appear more frequently in general. For example, in our dataset of RIB documents, the word RIB is present in all the documents, meaning that it adds no information about a document class.

5.2.2 Dimensionality reduction

The feature extraction procedures applied until now have the characteristic of generating high dimensional feature vectors. High dimensional vectors represent a problem for many machine learning algorithms, especially clustering.

It has been shown in [Beyer et al., 1999][Aggarwal et al., 2001] that when the number of dimensions increases, the nearest and the farthest neighbors to a given point have a similar distance. In this case, the nearest neighbor algorithm becomes ill-defined because it is not possible to define criteria for which a point is near or far. This phenomenon affects all clustering algorithms based on a similarity metric, like the Euclidean distance.

The solution to this phenomenon is the reduction of the number of dimensions under consideration by obtaining a set of principal dimensions.

Image

Even though we managed to reduce the original dimensionality with the feature extraction, 2,048 dimensions are still too many for the clustering phase, especially when we consider that we have 60,000 images.

To further reduce the dimensionality of the feature vectors we use the PCA (Principal Component Analysis), already used in [Babenko et al., 2014] for a similar application.

PCA aims to find a linear mapping of the data to a lower-dimensional space, maximizing the variability of the data in the low-dimensional representation. One method to apply PCA is to compute the covariance matrix of the data and then compute the eigenvectors on this matrix. Among the new dimensions, the principal components are the eigenvectors that correspond to the largest eigenvalues. These components can be used to reconstruct a large fraction of the variance of the original data. In our case, we simply drop the eigenvectors that contributes the least to the variance.

We decide to use only the first two components, which preserve 42% of the variance of the ADMINISTRATIVE dataset and 55% of the variance of the AVIS D'IMPOT subclass. The last step is the whitening of the data, which centers and scale the features. We have seen that haveing normalized components is essential for the following clustering phase.

Text

Similarly with the image case, the raw features extracted until now have 10,000 dimensions, which are too many for the clustering phase.

In this case, we mimic the approach of several works in the natural language processing domain, performing LSA (Latent Semantic Analysis)[Deerwester et al., 1990]. LSA is a technique to find relationships between a set of documents and the terms they contain by producing a set of related concepts. LSA assumes that words that are close in meaning will occur in similar documents. Once the term-document matrix is created, SVD (Singular Value Decomposition) is used to reduce the number of rows while preserving the similarity structure among columns.

In our work we apply LSA with a few differences, the first one being the dimensionality reduction technique, where we use the Truncated SVD for performance reasons. The second one is the normalization of the new vectors. Since clustering algorithms are susceptible to feature scaling, the low-rank components are normalized to have l2-norm equal to 1.

In our application with the RIB dataset, we reduced the dimensions from 10000 to 10, preserving 14% of the original variance.

5.2.3 Clustering

Once the images are mapped to a low dimensional space, we can cluster them. Hypothetically, we would expect a number of clusters equal to the number of classes. In practice, we should define the number of clusters higher than the number of categories. This operation is necessary because of the high intraclass variance: most categories are composed of images that could eventually be grouped in subclusters. As an example, the RIB category is likely to present different clusters associated with the different banks' formats.

Looking at the datapoint distribution in the low dimensional space we can synthesize the main characteristics that we would like the clustering algorithm to have:

- Unknown number of clusters: If it is true that we expect a number of clusters higher of the number of categories, we do not know precisely the exact number. Most of the clustering algorithms require the number of clusters as a hyperparameter, meaning that we should run and evaluate the clustering result several times.
- Density aware: We believe that our dataset shows different densities in the low-dimensional encoding because of the nature of the dataset: there are some classes where the documents show a series of differences, while there are classes where the variance is more limited. We expect this visual characteristic to exist also in the feature space. Clustering algorithms are usually unaware of density, and this may result in inaccuracies of the clusters.
- Noise detection: The whole point of the clustering phase is to find good clusters that can be annotated instantly by a human operator. Clustering perfectly all the points may be unfeasible, but clustering perfectly a subset of them should be easier. An algorithm with the ability to recognize and ignore difficult datapoints would meet our expectations.

These properties bring us to choose a density-based clustering algorithm such as the DBSCAN [Ester et al., 1996]. The DBSCAN is perfect for our scenario because it does not need the number of clusters as input, it is density based, and it detects noise datapoints. One of the problems of the DBSCAN is the tuning of the hyperparameters and the fact that it is necessary to specify a specific density. There are rules to find good values for the hyperparameters, but at the end, they are problem specific.

Moreover, our main problem is the fact that we have many clusters with different densities. Thus we should run multiple times the DBSCAN with different hyperparameters values and then aggregate the results. Our solution to the different density problem is to use a more complex version of the algorithm: the HDBSCAN [Campello et al., 2013], which uses a heuristic to cluster efficiently datasets with different densities in a single run. With HDBSCAN we only have to specify the

minimum number of points to define a cluster. The minimum number of points is a hyperparameter that can be chosen with simple criteria. A lower minimum number of points allows defining tiny clusters, while a higher minimum threshold favors large clusters. Ideally, our objective is to have good little clusters, while not having too many clusters, because then the annotation of the clusters would take too long.

The second parameter that influences the clustering result is the number of features selected after the dimensionality reduction step. This parameter is independent of the clustering algorithm used, but as a general rule the DBSCAN does not work well with high dimensional features. Thus we simply do a grid search over dimensions from 1 to ten. We found that for images, 4 dimensions are enough, while for text ten dimensions are good.

Evaluation

There exist different techniques to evaluate a clustering result, the majority of them based on the true labels associated with the datapoints. Unfortunately, we do not have the labels and we can rely exclusively on metrics based on the datapoints and the clusters.

A popular measure regarding clustering is the silhouette [Rousseeuw, 1987], which coefficients measures how close each point is to his cluster and how far is to the others. The silhouette measure has a [-1,+1] range, and a value of 1 means that the datapoint is in a cluster well separated from the others, while a value of -1 means that point is probably in the wrong cluster. The main drawback of the silhouette is that its value is higher for convex clusters than other concepts of clusters, such as those that characterize density-based clusters, exactly those that we are interested in. Unfortunately, this drawback makes the silhouette score unreliable for our application.

To evaluate the clustering results, we define some metrics:

1. Number of Clusters: Few clusters are bad because they are too generic, while too many clusters make the annotation time-consuming. In this optic, the annotator should define an ideal number of clusters and choose hyperparameter values accordingly.

2. Ratio of Noise Points: Ideally, we do not want noise points, because we know that in reality all points are part of a category.

Empirically we have seen that when the minimum number of points is low, the ratio of outliers is higher. To fix this problem, we run the HDBSCAN two times: the first time we run it on the full dataset, while the second time we run it on the outliers. In this way we obtain the first set of clusters that are well behaved, meaning that they contain almost equal documents. The second set of clusters

contains the documents targeted the first time as outliers. Theoretically, they should contain more diverse documents, and they do in practice.

An additional tool to evaluate the clustering result is the visualization of the clustered datapoints on the two principal components.

Clustering results

In our experiments, we have seen that the percentage of clustered documents is often equal to 50% of the dataset.

For what concerns the RIB dataset we used ten dimensions and 11 min points. We did the clustering only one time, resulting in 74% of the datapoints clustered and 68 clusters. For this dataset, we decided to stop here and eventually annotate the remaining part manually. We have seen an interesting phenomenon with the RIB dataset: a portion of documents had an empty text field. These documents were pictures with bad lightning conditions and we think that the binarization phase of the text extraction damaged the document, making the text extraction impossible.

Regarding the AVIS d'IMPOT subclass, we use four dimensions and 4 min points. We did two runs of the clustering, obtaining the first time 66 clusters and 32% of the documents clustered. Also for this category, we decided to stop at this point and do a manual annotation.

With the ADMINISTRATIVE dataset we use four dimensions and 4 min points. After the first run of the clustering, we obtain 52% of the datapoints in 402 clusters and the remaining document tagged as noise. A second run shows a percentage of clustered documents of 47% and 74 clusters. This brings the total of clusters to 476 and the percentage of clustered points to 74%.

At this point, we can reformulate the time required to annotate the ADMINISTRATIVE dataset as a function of the number of clusters. Given N clusters and a cluster annotation time of 120 seconds (annotating a cluster may take more than a single image), we have

$$\text{time} = N \times 120 \text{ seconds}$$

In our case we had 476 clusters, with an estimated annotation time of

$$\frac{476 \times 120}{3{,}600} = 16 \text{ hours}$$

5.3 Fine annotation

This phase has been performed exclusively on the ADMINISTRATIVE dataset because it is the largest and most important dataset for this work.

After the clustering, the majority of the documents are associated with a provisory label, but there are two problems: some documents are not annotated because

they represent noise in the second iteration of the clustering, and some documents are annotated wrongly.

To solve all these problems we build a model that has the objective to correct the annotations. Then in a second phase, we perform the manual annotation of the remaining documents. The whole approach is an adaptation of [Yu et al., 2015], where a system with human in the loop is used to annotate a huge dataset. The main difference between our system and theirs is the fact that the first set of annotated images is the result of clustering. We were able to do the clustering because many categories have subgroups of similar images.

The system consists of a process that is composed in the following phases:

1. A network is trained from the annotated images
2. The trained network is used to classify all the images
3. The classification results are used to correct wrongly annotated images and to add the newly annotated images to the annotated dataset

Our images are initially divided into two groups: annotated images and not annotated images. These two groups are the results of the clustering mentioned in the preceding section. At this point, the annotated images are used to train a classification model. The model is a Xception network pre-trained on ImageNet, it is the same network used to perform the feature extraction. This part is based on the fact that neural networks are robust to massive noise in the dataset [Rolnick et al., 2017]. This observation leads us to train a classifier on possibly noisy labels, with the guarantee that the result will be acceptable.

Once the model is trained, it is used to classify all the documents, both the annotated ones (74%) and the not annotated ones (26%). At this point, the classified documents can be divided into different groups:

1. Cluster documents correctly classified: 72% of the images are part of this category.
2. Cluster documents wrongly classified: These are documents that may have been put in the wrong cluster. 2% of the images are part of this group.
3. Noise documents classified with high confidence (>80%): These are images that can be added to the set of the annotated images. 16% of the images are part of this group.
4. Noise documents classified with low confidence: These are images that the model is not sure about. We manually annotate them, and then we add them to the set of annotated images. 10% of the images are part of this group.

The final result is that only 12% of the images had to be annotated manually.

Fine annotation results

As we said, we had to annotate manually 12% of the dataset. Considering 5 seconds per annotation, the operation took

$$\frac{7200 \times 5}{3600} = 10 \text{ hours}$$

Summing up, the annotation of the whole dataset took 26 hours instead of 83 hours. In all this, we estimate that the number of wrongly annotated documents is less than 2%. These documents are part of the of the documents initially clustered that were correctly classified with low confidence values. Since we point to a classification accuracy of 95%, this is acceptable.

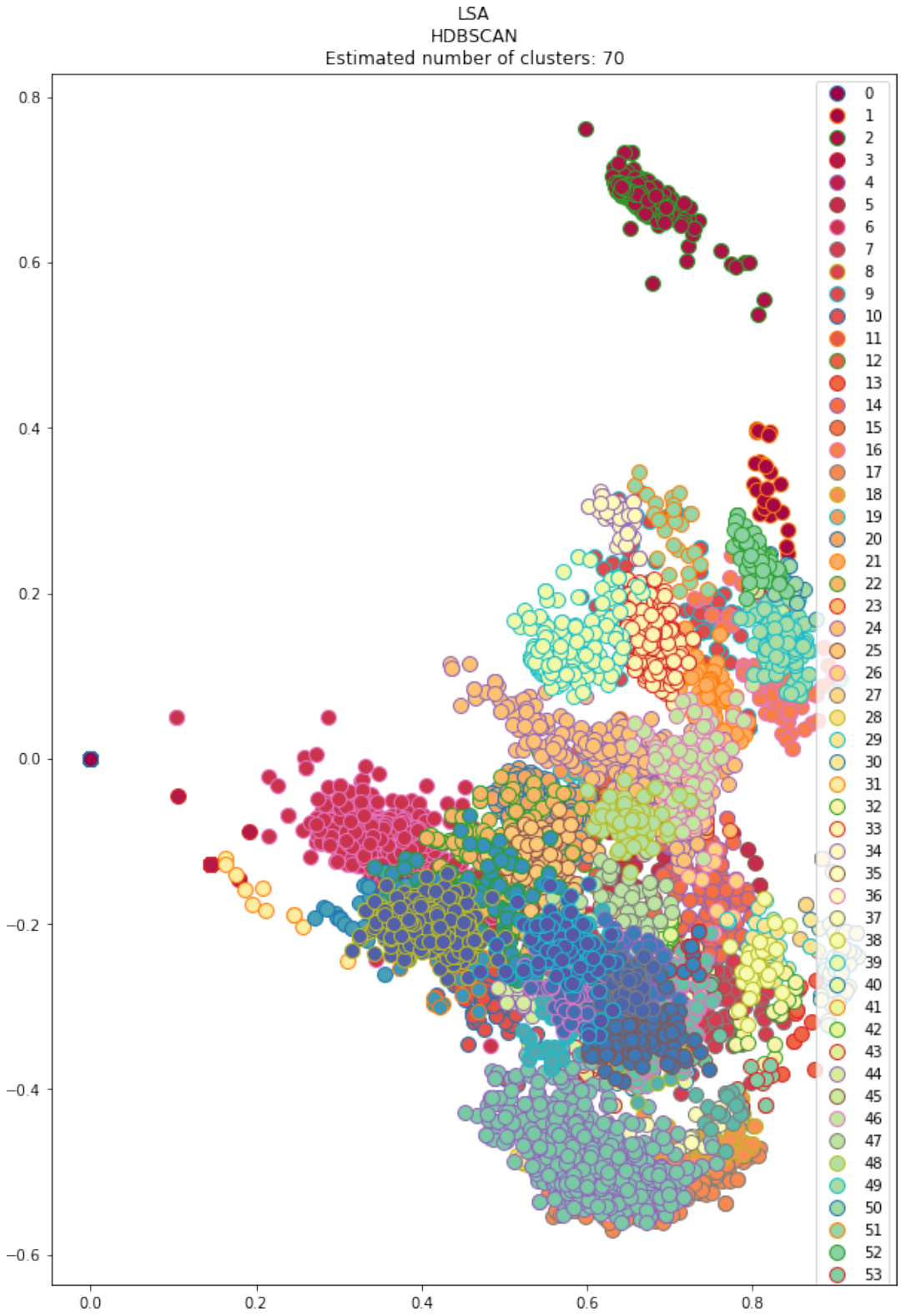

Figure 5.2: Clustering the RIB class

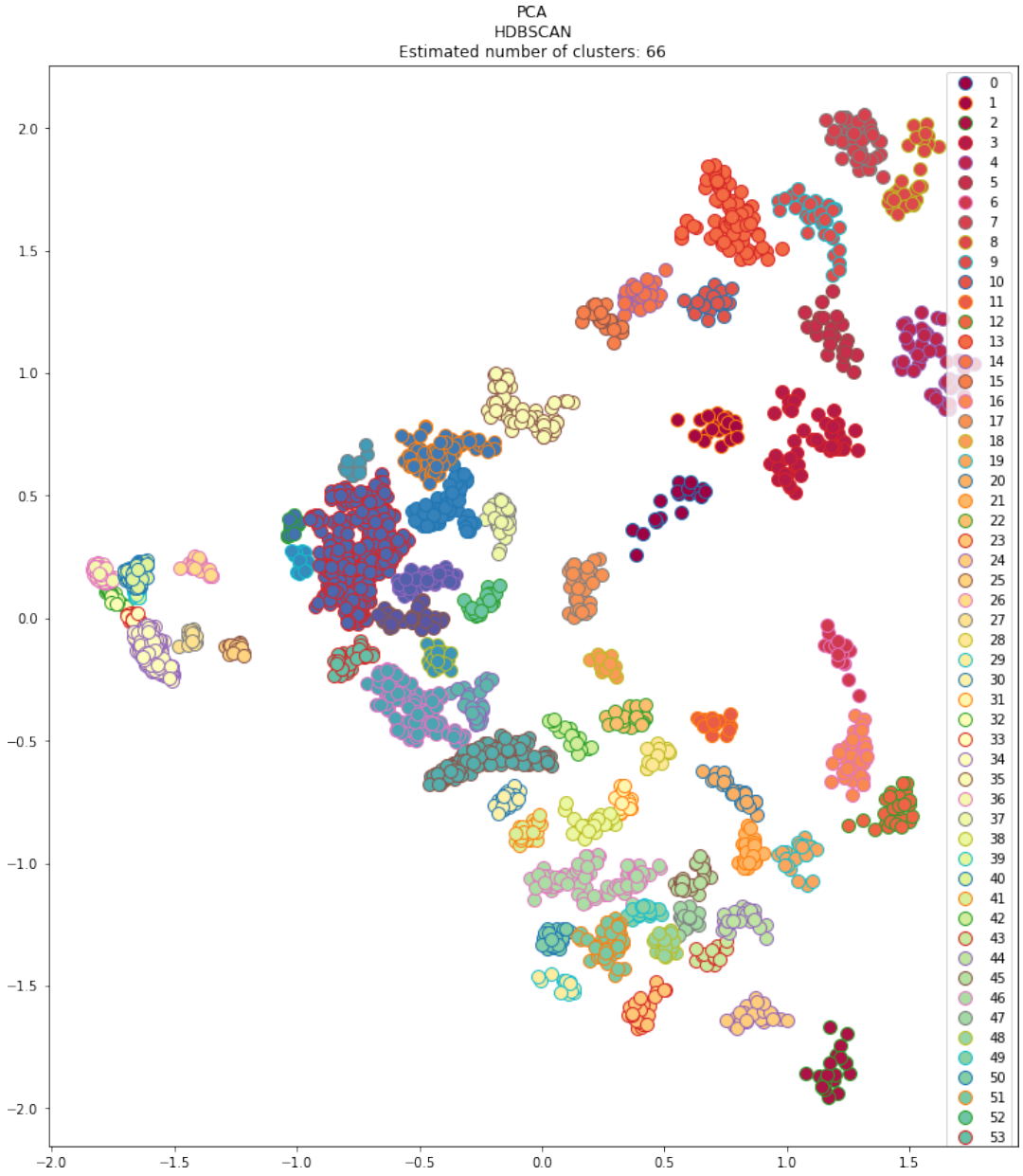

Figure 5.3: Clustering the AVIS D'IMPOT class: the subclasses are clearly visible

Chapter 6

Image preprocessing and data augmentation

In this chapter we firstly show the preprocessing operations that we apply to the documents before the classification to improve model accuracy and training time. Then we treat the topic of data augmentation, with an in-depth analysis of the type of augmentation that fits better the document datasets.

6.1 Preprocessing

Every image can be represented as a set of matrices (tensor) of pixel values. Each matrix corresponds to a channel, the conventional term used to refer to a color component of an image. An image from a standard digital camera has three channels: red, green and blue. Each channel is a 2d matrix having pixel values in the range [0,255]. In order to feed the image to the classifier it is necessary to perform different operations that can include the grayscale conversion, the resizing and the value scaling

6.1.1 Colors

In the more general image classification, colors represent important features that can help the model to discern specific classes. For example, a model trained with landscapes will learn that grass is associated to green tonalities and that a different color is enough to exclude certain classes. With documents, this observation still holds true in those cases where documents have very specific colors, like some forms from the public administration. In our datasets we have exactly a case like this in the 4.2.4 dataset, where forms show different colors. In such a case, colors have been used to facilitate the classification operation of human operators. Humans, like machine learning models, use the color of the image as the first feature to perform

a classification because it is directly available from the image and no processing required.

Until now it seems that colors are a useful feature and we should use them when available. On the other hand, colors may trick a model for the same reason they are so useful: whenever a classifier associates a color with a class, if not forced it does not learn any other feature about that specific class because color is enough. The consequence of this phenomenon is that if a test time we feed an image where colors are different from the usual ones, the classifier makes a completely wrong prediction. In our case we are aware that the testing conditions are different from the training ones, especially regarding the lightning conditions that can vary considerably the perceived color of a document.

In order to solve this color problem we can convert our images to grayscale. In this way the model is forced to learn the structure of the documents and it is not anymore fooled from different colors. In order to perform the grayscale conversion we can simply average the three channels and then stack three copies of the new 2d matrix, in this way we can still use pretrained models that processes RGB images.

The main drawback of using only grayscale images is that we lose some information from the images that could help the model. If we think about our image classification capability, we can note that even though we can still classify grayscale images, the whole process is slower because we have to concentrate on more complex features of the image.

To conclude, deciding wether using color or grayscale images depends from the dataset. In this work we decided to build a more powerful model by using colors, but we managed to solve the robustness problem augmenting the dataset with grayscale images and with images with slight color variations.

6.1.2 Resizing

Resizing is fundamental for a neural network for many reasons, the most important one being performance. Classic image classifier are based on features that are extracted from normal images that are quite large. These methods do not work well when the image resolutions starts decreasing, meaning that only high-res images can be used with the consequence that the feature extraction requires a lot of time.

On the other hand, neural networks are able to work on much smaller images because they look for much deeper features than classical methods. A benefit linked with this is that the inference time of neural networks is really low compared with the classical methods. At the same time, neural networks are also forced to work with low-res images because otherwise the training phase takes too many resources. To understand this phenomenon, we should think that neural networks can be described as a sequence of matrix multiplications and non-linearities. Matrix multiplications are $O(n^3)$, which makes the dimensioning of the input image fundamental. Experimentally it has been shown that convolutional neural networks

perform well with images larger than 224×224, with diminishing returns for bigger images. In our case we resized the image to 224×224 or to 256×256 to perform different tests with different models that were constrained to a specific dimension.

The core of the resizing of an image is the resampling method, which influences the final quality and the time required for the operation. The most common resampling methods ordered by increasing complexity and quality are:

1. Nearest
2. Bilinear
3. Bicubic
4. Lanczos

If it is true that for common image classification the nearest resampling is enough and that the more complex techniques have diminishing returns, for document classification it is not the case. For example, an image of a car is recognizable even when the image quality is low because it has strong characterizing features that are preserved when resized. The same is not true for a document, because the characterizing features are much more subtle and interleaved, meaning that a bad resampling has destructive consequences. In our datasets we have seen that the Nearest resampling can not be used because it completely deteriorates the image. Regarding the other filters, they perform well on our datasets and produce similar images with only minor differences that do not alter the contents. In our work we decided to use the Bicubic resampling, but also Bilinear and Lanczos are valid alternatives. An idea that we did not investigated is feeding to the model images resized with different techniques, with the hope of generating a more robust model.

Apart from the resampling methods, the other parameter that define a risizing is the final aspect ratio. This problem arises because the original image is an a4 paper, while the final image is a square. In order to perform a resize between different aspect ratio there are different techniques. In this case we name the methods like iOS does [UIKit, 2017]:

- ScaleToFill: Scale the content to fit the size of the view by changing the aspect ratio of the content if necessary.
- ScaleAspectFit: Scales the content to fit the size of the view by maintaining the aspect ratio. Any remaining area of the view's bounds is filled by a given color.
- ScaleAspectFill: Scales the content to fill the size of the view. Some portion of the content may be clipped to fill the view's bounds.

In our work we use the ScaleToFill method, which may change the aspect ratio of the image if it does not match the target dimensions. Like the resampling methods, a model would probably benefit from seeing images resized with different methods.

When we deploy the application, if we do not use different scaling options we have to make sure that the deployment methods match those used for training the model. This is absolutely important because they change considerably the image and we do not expect the model to be automatically invariant to different scaling methods.

Efficient preprocessing pipeline

When we train our model we have to decide if the preprocessing operations should be made online or offline. The offline method is optimal when the preprocessing operations are well defined and do not change over time, while the online method is necessary when transformations are applied to images on the fly. Between the two methods the online one is for sure the most flexible, even though it repeats the same operations every time. For this reason, the online preprocessing may slow down the training of the model if not well optimized.

Another observations to be made is that most of the preprocessing operation can be performed online because they are relatively lightweight; they do not have any impact on the training time because it is possible to apply them while the gpu is processing a batch.

Unfortunately the resizing operation is quite onerous, and for this reason we perform it before the training phase. An interesting thing is that even though we apply the resizing only once, it remains a time-consuming operation. The majority of libraries and applications that can resize images using the CPU are single threaded, and they are not optimized to use the special routines of modern processors (SSE4 and AVX2).

In this section, we solve the problem by parallelizing it and by using high-performance libraries. Regarding the high-performance library, we choose Pillow-SIMD. Pillow is the common Python Image Library, and Pillow-SIMD is a fork specialized in doing common operations faster by using SIMD instructions. Even though Pillow-SIMD is fast, it is still single threaded. For this reason, we parallelize the operations. A main thread fill a queue of the filenames to process and then multiple processes consumes the queue. Each process read the image, convert it and store it in the new directory. As expected, the parallelization increase performances in a linear way with the number of cores.

It is important to state that the Pillow-SIMD library has the same interface of Pillow, meaning that no change in the code are made. For now the performances are good enough: we are able to resize the whole TOBACCO-3482 dataset in 30 seconds. The original images have different dimensions, but the most common ones are 2560×3296 and 1728×2292; the images are resized to 256×256 with bicubic

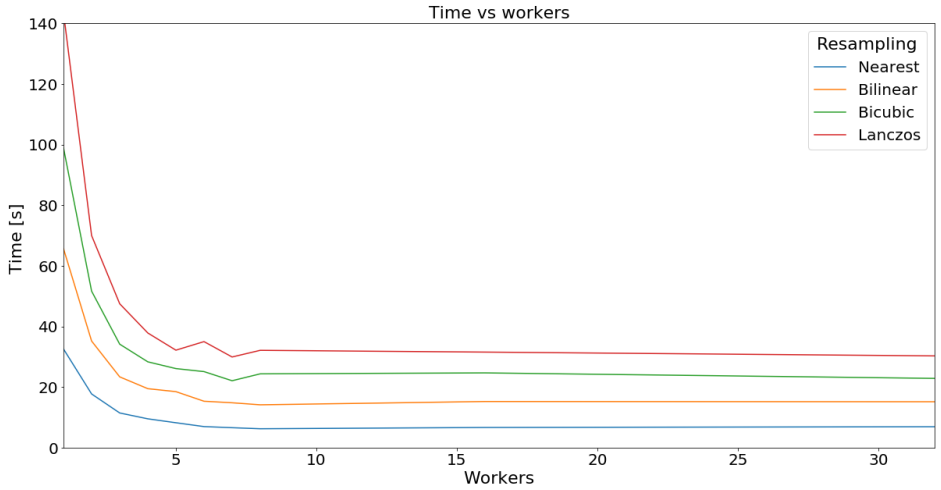

Figure 6.1: Resizing time for different resampling vs number of workers

interpolation and saved in jpg with 100 compression quality.

In future, when the dataset dimensions will grow we will move toward a resizing phase performed on GPU using either OpenCL or CUDA.

6.1.3 Value scaling

The last preprocessing operation to perform on the images is the pixel value scaling. Pixel values of an RGB image range from 0 to 255, but the majority of machine learning systems behave better when values ranges have zero mean and unit variance.

We scale the input values in the range [-1,+1], in this way we are sure that the first layers of the neural network will behave well. We do not use more complex scaling solutions, such as variance normalization because it would introduce parameters different for each dataset and it would complicate the deployment of the model. We should also consider the fact that most neural network architectures use normalization layers that adjust the data distribution during the training, reducing the need for a hand-made scaling. We can define the scaling operations in two ways:

- Mean-Variance: in this definition we first subtract the image mean per channel and then we divide for the variance.

$$\text{scaled image} = \frac{(\text{image} - \text{mean})}{\text{variance}}$$

- Scale-Bias: in this definition we multiply the image for the scale value and then we add the bias for each channel.

$$\text{scaled image} = (\text{image} \times \text{scale}) + \text{bias}$$

The two definitions are interchangeable, but it is important to calculate values for both because in the deployment phase we could be forced to use a specific definition.

6.2 Data augmentation

Deep neural networks need millions of images to be trained. Unfortunately, this is rarely the case because collecting and annotating a large number of images is expensive. This problem is usually solved by generating new images through simple transformations that do not change the semantic class of the image. This approach is called data augmentation.

6.2.1 Data augmentation and image preprocessing

Most of the times the training dataset is coherent with the real distribution, the deployment scenario. With documents, this is not always true, and the main source of difference is the capture bias.

Almost all document preprocessing pipelines try to make a picture of a document similar to a scanned copy of it. Common phases are binarization, deskewing and contouring, which allow to obtain a clear image of the document, similar to the image obtained with a scanner. All these phases are difficult to apply when the original picture shows backgrounds, heavy noise, and brightness effects. If we think that in the future the majority of the documents will be sent as pictures, we can not rely on an image preprocessing pipeline which may degrade the picture.

Instead, we would like to have a model that can take care of all the possible effects that contribute to capturing bias. This makes our work simpler because we do not have a hand engineered pipeline that may require further optimization for each new dataset. A basic fact about capture bias is that if we feed the model with images that are different from those that will be used in the real conditions, the model will not work when deployed.

The approach that the augmentation process proposes is an entirely different solution: when there is a problem with the data we do not fix it, but we teach the network to take care of it. To do this, we have to generate new training images similar to those in the real world.

6.2.2 Online data augmentation

The augmentation process is not done offline because it inherently grows the dataset size. For example, if we have a dataset of 60,000 RGB images with a resolution of 256×256 and 100 epochs of augmentation, we have:

$$\text{memory} = \frac{256 \times 256 \times 3 \times 60{,}000 \times 100}{2^{30}} = 1{,}098\text{GB}$$

This solution, even though the best one from the simplicity and performance point of view, does not scale with the limited systems at our disposal. A better solution is to do the data augmentation online during the training process.

Our Data training pipeline constitutes a set of processes each one having a copy of the dataset. When the training starts, each process shuffles the dataset and generates the batches of filenames that will be used for the training. Each batch is processed sequentially: the process reads the images and then apply the augmentation pipeline. The augmented batch is then put into a global queue that is fed to the GPU. When the list of batches of a process ends, the process restarts the process shuffling the dataset and generating the new batches. This solution is at the same time simple and efficient and allows us maximum flexibility.

6.2.3 Image transformations

For what concerns the data augmentation process, it is a rather old technique that tries to make a model more robust by generating new samples. Samples are generated from existing ones with procedures that are domain dependent (image, text, speech), but in general, the augmented data needs to be part of the distribution that we want the model. From a probabilistic point of view, the model is trying to approximate a distribution, and the lack of samples limits us from achieving this. By augmenting the data, we can extract new samples from the distribution, with the hope that they can ease our work. The most important thing that should always be considered is that the generated samples should be part of the distribution that we want to model. If we generate data different from the real distribution, we end up approximating a different distribution, different from our target and the model will not work in deployment.

Each of these transformations can also be seen differently: we are adding invariance. For example, if we think of a model that is trained with images perfectly centered, we can not expect that it also works in other conditions. Instead, when we train the model with images that are translated, it is likely that the model learns translation invariant features. More generally, each property that is augmented add invariance respect that property.

Rotation

Out of the box, CNNs are invariant to small rotation invariant because of the Max Pooling layer; if the rotation is small enough that the maximum activation of a pooling region does not change, the result is the same. Unfortunately we need a model that is invariant to large rotations. Especially in the document domain, this is a big problem, because people may send pictures of documents that are rotated. To classify rotated images we either use an architecture that is rotation invariant, or we augment the dataset. Previous work have shown that it is possible to create rotation invariant CNN [Gonzalez et al., 2016], but these approaches have the disadvantage of modifying an architecture, with the impossibility to use standard pretrained models. The second and most straightforward approach consists in rotating the training images.

Regarding the implementation of the orientation augmentation, we do not perform the augmentation over 360 degrees. Instead, we first orientate the image of multiples of 90 degrees, and then we rotate it by 10 degrees at maximum. We use this technique to facilitate the training, in fact, we have seen that the when the documents are rotated with all the possible orientations the model has some difficulties in the learning process. Nonetheless, the optimization we did should not have repercussion because usually documents images show little orientations.

As a note, images in the dataset may not be completely aligned, thus leading to the generation of severely rotated images (e.g., 45 degrees). In order to solve this problem we may use rectification techniques for the training images, thus starting from completely straight images for the augmentation.

Scale

Convolutions do not learn scale invariance. The problem of scale is more evident when the document size is small respect to an a4 paper. If we think to an ID card, we can find scanned images where the document occupies a limited portion of the image, while in pictures it occupies the whole image. With neural networks there are two techniques to take care of scale: one is creating a scale invariant model, the other one is to augment the dataset with different scales of the training images. Scale invariant models like [Xu et al., 2014] and [Kanazawa et al., 2014] apply convolutions on different scales of the image, incorporating in this way the scale prior in the model. The second techniques achieves the scaling invariance not at the architecture level, but in the dataset. Showing scaled images to the network is enough to teach the network to detect features at different scales. This technique require more manual tuning of the scaling parameter, but has the advantage of being model invariant: we can use any model, without changing the architecture. This means that we can iterate faster and use pretrained models. Regarding the implementation of the scaling, the scaled image will have a scale between 80% and 120% of the original image. Whenever the scale is less than 100% we have to add a

border to the image and in our case we use a scale of gray with an intensity between 0 and 255. This operation should teach to the model to classify documents with a visible background like a table. On the other side, when scale is greater than 100% there is the risk that the training image is cropped, and in this case the model will learn to classify a documents considering only the content. One drawback of the scaling is that it can generate training images hard to classify even for a human; for example if an ID card that occupies a small portion of a paper is scale down, it will be even smaller and more difficult to classify.

A possible improvement of the scaling operation would be finding the contour of the image and than scaling only the area of the document. In this way we could achieve even better transformations because we would start from a document that occupies the whole image.

Translation

Translation invariance is the ablitity to perform well on inputs that show a translation respect to the training images. CNN are partially translation invariant because they are composed of convolutions and pooling layers. The convolution operator is by definition equivariant respect to translation. This means that if we translate by one pixel right an image, the convolved features will be shifted one pixel right. At the same time, Max pooling achieves partial invariance because the max of a region depends only on the largest element of the region. If a small translation doesn't bring in a new largest element int the pooling region and also doesn't remove the largest element, then the max doesn't change [Goodfellow et al., 2016].

Unfortunately the small invariance provided by CNN is not enough to encounter the large translations that documents can have. Again, the most difficult images are the small documents. For example, an ID card can fill the central part of an image, but also the top or the bottom. In order to add invariance to large translations we can only augment the dataset with translated copies of the original images. This is enough for teaching the model to recognize the same document in different locations.

Regarding the implementation of the translation, we should decide how much we want to translate the images. Small translations do not cover all the cases that the model may face, while large translations may alter completely the semantic class of an image. For example, we can imagine a small ID card that is on the top right corner. If we perform large translation (e.g, 30%) we may put the document outside the image border, generating a white image that for sure we should not classify as a document. At the same time, large translations are not good for documents that cover the entire image because we may cut large portions of the document.

Like the scaling issue, a more refined approach would be starting from a document that cover the whole image. In this way we would have certainties about the

translated image. In this fashion, it is important to relate the scaling and translation transformations because the results strongly depends from the two transformations combined together. For example, when we zoom an image we discard always its borders, thus in these cases the maximum translation permitted should be low. On the other side, when we zoom out an image we should allow large translations to cover all possible positionings of the documents.

Perspective

Whenever the capturing device is not parallel with the document, parallel lines becomes intersecting. This phenomenon can not be reproduced with the transformation that we have seen until now: Rotation, Scale, Translation. This means that a new transformation is needed: the perspective transformation. This transformation is particularly useful to transform scanned images into images taken with a camera because it can convert parallel lines into intersecting lines. Such a transformation can generate completely unrealistic results, thus we use it with parsimony.

Lightning

Shadows and lights are not a problem when an image is scanned; they simply do not exist because the light is uniform. When taking pictures with a camera, instead, they are very common, especially if the pictures are taken in a closed environment with an artificial source of light. Correcting lightning effects is extremely hard, and a simple proof of this is the fact that binarization algorithms do not hand well images with strong lights and shadows

Our solution is to add shadows and lights to an image so that the model will learn to model also this phenomenon.

Histogram manipulation

Histograms show the value distribution of the pixels. They are extremely useful in image processing because most operations on the contrast and brightness can be performed as manipulations of the histogram. In our work we apply mainly contrast normalization and a brightness shift, which allow to generate a large number of images not available in the training set. This group of transformations is probably the one that can benefit the most from further tests because the possible transformations are very numerous.

Color shift

As we described in 6.1.1, colors can be extremely helpful for classifying a document, but they can also harm it if there is a large color shift at test time. Our solution to have the advantages brought by colors and the robustness of grayscale images, we

augment the dataset with images with shifted colors and with grayscale images. We apply the color shifting during the contrast normalization phase, simply doing the operation channel wise with random values. Regarding the grayscale augmentation, we average the values of the channels and then we average the grayscale image and the color image with random weights. These transformations are enough to generate a large number of combinations of colors.

Blur, Noise

Recent works have shown how much the image quality can degrade model accuracy on prediction [Dodge and Karam, 2016]. These findings should be taken seriously in our work because it is likely that because of the capture bias 4.1.3 the test images contains artifacts different from those in the training images. Such artifacts are mainly the presence of blurring, caused by out of focus pictures, and the presence of noise, caused by the capturing conditions and devices.

Random occlusion

Occlusion is a critical factor for the generalization ability of CNNs. When some parts of an image are occluded, a robust model should be able to recognize the category from the visible document structure. However, the collected training samples usually exhibit limited variance in occlusion, meaning that the model will behave well only when the whole object of interest is visible. A possible solution might to manually add occluded images, but as we have seen in 4.1.1 we would probably miss something, adding a bias to our model. Another problem would be related to the time needed to add the occluded images to the dataset. A more efficient solution is to augment the images with random patches of noise.

At first analysis, occlusions invariance does not seem relevant with documents, because when people send them, they make sure that the whole document is visible. On the other hand, experimental tests from [**?**] shows robust improvements on a variety of dataset when the random occlusion augmentation is included. These datasets do not contain occluded images on the test set, meaning that the models learn a more robust representation of the dataset. The random occlusion augmentation can also be seen as a stronger version of the shadow and light transformations. For this reason in our tests we used mainly the random occlusion augmentation.

In 8 we trained the same model using or not the random occlusion augmentation and we found alternating results. We believe that this could be the result of the interaction of the random patches with the other transformations, which could lead to incorrect training images in those cases where fundamental parts of the document are covered.

Code

Augmentation is performed using a python library designed exactly with that purpose: imgAug [Jung et al., 17].

```
1
2  from imgaug import augmenters as iaa
3
4  seq = iaa.Sequential([
5      iaa.Sometimes(1.0,Rotator90()),
6      iaa.Sometimes(1.0,iaa.Affine(
7          scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
8          translate_percent={"x": (-0.20, 0.20), "y": (-0.20, 0.20)},
9          rotate=(-10, +10),
10         cval=(0, 255),)
11         ),
12     iaa.Sometimes(
13         0.5,Eraser(pixel_level=True)
14         ),
15     iaa.Sometimes(
16         0.5,iaa.AdditiveGaussianNoise(scale=(0.*255,0.15*255))
17     ),
18     iaa.Sometimes(
19         0.5,iaa.Sharpen(alpha=(0.,1.0),lightness=(0.5,2.0))
20     ),
21     iaa.Sometimes(
22         0.5,iaa.ContrastNormalization((0.5, 1.0), per_channel=0.5)
23     ),
24     iaa.Sometimes(0.5,iaa.GaussianBlur((0, 2.0))),
25     iaa.Sometimes(
26         0.5,iaa.Grayscale(alpha=(0.,1.0), from_colorspace='RGB')
27     ),
28     iaa.Sometimes(
29         0.5,iaa.Add((-10, 10))
30     ),
31     iaa.Sometimes(0.5,iaa.PerspectiveTransform(scale=(0., 0.03)))
32 ])
```

In the portion of code, we show how imgaug allows defining an augmentation pipeline as a simple sequence of transformations. One of the exciting things is the fact that it is possible to specify the percentage of times that an operation is performed and the fact that the library provides a large number of transformations. This was very useful in our experiments because we have noted that if all the operations are performed together the final result is not realistic and affects negatively the trained model.

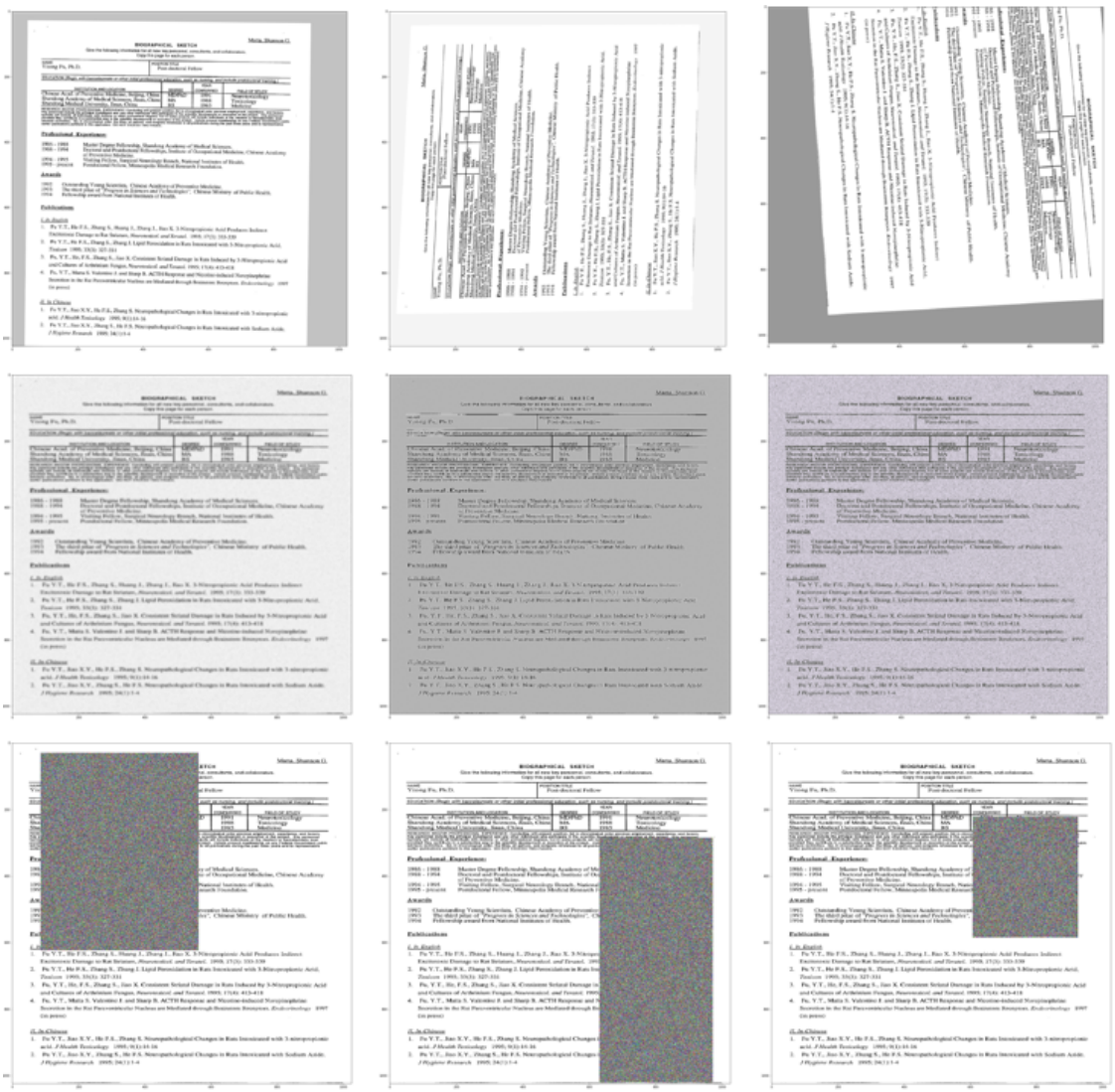

Figure 6.2: Augmentation examples from the Tobacco dataset.
First row: Rotation, Scaling, Translation, Perspective.
Second row: Contrast, Blur, Noise, Sharpen.
Third: Random erasing

Chapter 7

Neural Network Introduction

This chapter covers the building blocks of modern Neural Networks. We start with a logical and mathematical description of what a Neural Network and we show the role of activation and loss functions. Then we describe the training process and the idea behind the gradient-based learning and the back-propagation. We conclude with the description of the transfer learning, which is a practical technique used to improve generalization of models trained with small datasets.

7.1 Feedforward networks

Feedforward networks represent the simplest and most common deep learning models. The goal of a feedforward network is to approximate a function F^*, which in the case of a classifier associate an input X to a class y. A feedforward network defines a mapping $y = F(X, W)$ and learns the value of the parameters W that result in the best function approximation. These models are called feedforward because information flows only one way, from the input X to the output y, through the computations defined by F. The model is often defined as a directed acyclic graph describing how the functions are composed together. When the output is the result of multiple intermediate functions we have a deep feedforward network.

7.1.1 Activations

To understand feedforward networks, we start with a linear model and then we build on top of it until we have a Neural Network. Linear models are fascinating because they fit efficiently and reliably, either in closed form or with convex optimization. Linear models also have the drawback that their capacity is limited

to linear functions, with the consequence that the model cannot model the interaction between multiple input variables. We can define a linear model like this: $y = W^T x + c$ where W is the weight matrix, x the input vector and c the bias vector.

At this point, we may start stacking different layers to obtain a model with higher capacity. Unfortunately, this does not work because a linear combination of a linear combination, is still a linear combination respect the input.

To allow the Neural Network to model a target variable that varies non-linearly with its explanatory variables it is necessary to use non-linear activation functions. Most neural networks do this using an affine transformation controlled by learned parameters, followed by a fixed, non-linear function called an activation function. We use that strategy here, by defining $y = F_n(W^T x + c)$, where F is the activation function.

In modern neural networks, the default recommendation is to use the Rectified Linear Unit (ReLU) function [Hahnloser et al., 2000], defined by the activation function $f(x) = max\{0, x\}$.

7.1.2 Loss function

The majority of modern neural networks are trained using maximum likelihood. This means that the loss function is simply the negative log-likelihood, equivalently described as the cross-entropy between the true distribution p and the model distribution q:

$$H(p, q) = -\sum_x p(x) \log q(x)$$

.

In a classification scenario, the true distribution has all the probability mass on the correct class (i.e. p=[0,. . . 1,. . . ,0]. When we want to represent a probability distribution over a discrete variable with N possible values, we can use for the last layer the softmax function, defined by

$$f_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

.

The softmax function takes a vector of real-valued scores and squashes it to a vector of values in the range [0, 1] that sum to 1, generating a proper probability distribution. Probability distributions based on exponentiation and normalization are common in the statistical modeling literature because the log operation in the loss function undoes the exponentiation of the softmax.

7.2 Training

Neural networks learn through Gradient-based learning, which builds on the fact that the loss function can be minimized by estimating the impact of small perturbations of the parameter values on the loss function.

In a Neural Network the set of parameters W is a vector of real numbers that are initialized randomly, with respect to which the loss function $E(W)$ is continuous and differentiable almost everywhere. In such a scenario, the training aims to minimize the loss in an iterative procedure where W is updated at every step in the following way:

$$W_k = W_{k-1} - \epsilon \frac{\partial E(W)}{\partial W} \tag{7.1}$$

In the most simple version, the ϵ parameter is a scalar constant known as learning rate, but there are more elaborated optimizers that use variable learning rates. In general, with a high learning rate longer steps are taken in the weight updates, and thus, it may take less time for the model to converge. Nonetheless, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal minimum.

We can apply the above equation either on the whole dataset (Batch Gradient Descent) or on a portion of it (Stochastic Gradient Descent). A good minibatch size is small enough to avoid some of the poor local minima but large enough that it does not avoid better minima. One benefit of SGD is that it is computationally faster and it is possible to use large datasets that often do not fit in RAM.

7.2.1 Back-propagation

"The basic idea of back-propagation is that gradients can be computed efficiently by propagation from the output to the input"[Lecun et al., 1998].

Deep Neural Networks are built as a stack of layers, each implementing a generic function $X_n = F_n(W_n, X_{n-1})$. X_n is a vector representing the output of the layer, W_n is the vector of trainable parameters of the layer, and X_{n-1} is the input of the layer, as well as the output of the preceding layer. We refer to E^p as the loss associated to the input pattern Z^p.

The main statement of back-propagation is that if the partial derivative of E^p respect to X_n is known, then the partial derivatives of E^p respect to W_n and X_{n-1} can be computed using the backward recurrence

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n}$$

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1}) \frac{\partial E^p}{\partial X_n}$$

where $\frac{\partial F}{\partial W}(W_n, X_{n-1})$ is the Jacobian of F with respect to W evaluated at the point (W_n, X_{n-1}), and $\frac{\partial F}{\partial X}(W_n, X_{n-1})$ is the Jacobian of F with respect to X. The first equation is responsible for calculating the gradient respect to the model parameters, while the second equation is responsible for the back-propagation.

7.2.2 Transfer learning

"Transfer learning and domain adaptation refer to the situation where what has been learned in one setting is exploited to improve generalization in another setting"[Goodfellow et al., 2016].

The main reason for the success of transfer learning is that models trained on huge dataset learn general features that facilitate the training process with small datasets, often obtaining better performances respect to models trained from random weights.

A first application of transfer learning uses Deep Neural Networks as fixed feature extractor. The last fully-connected layer is removed, and the network is used as a feature extractor. Once the features are extracted, it is possible to train a classifier for the new dataset.

The second strategy is to replace the top classifier and fine-tune the weights of the whole network by continuing the back-propagation. It is possible to fine-tune all the layers, or it's possible to keep the first layers frozen and only tune the higher-level layers. The observation motivates this is that the first layers contain more generic features, whereas the higher-level layers are more specific to the details of the classes of the original dataset[Li et al.,].

Chapter 8

Image classification

This chapter starts with an introduction of the characterizing blocks of CNNs, which are used for all the image classification tasks of this work. We continue with the description of the MobileNet and DenseNet architectures and the training protocol we use. To conclude we show the classification results with internal and external datasets.

8.1 Convolutional Neural Networks

CNNs take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 [Hubel and Wiesel, 1962] where they showed that some individual neuronal cells in the brain responded only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks is one that machines use as well, and is the basis behind CNNs. Modern CNN were proposed by Yann LeCun et al in 1998 [Lecun et al., 1998] and the building blocks have remained unchanged.

CNN share the building blocks of a common Neural Network, with the innovation of using two new operations: convolutions and pooling. As explained in 2.2.1, CNN have shown to be the best tool for document classification.

8.1.1 Convolutions

CNN derive their name from the convolution operator. The primary purpose of convolutions in case of a CNN is to extract features from the input image. Convolutions preserve the spatial relationship between pixels by learning image features

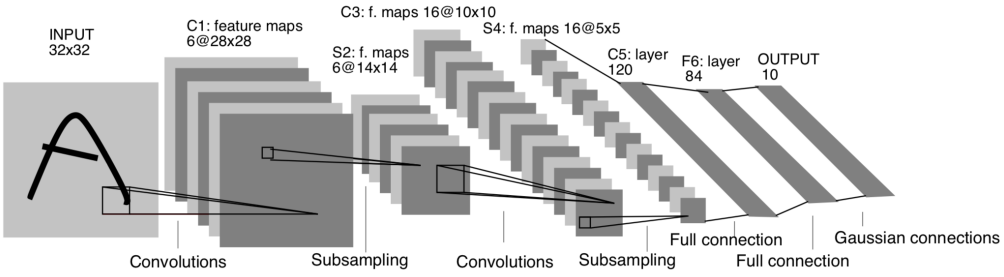

Figure 8.1: LeNet-5 architecture, with convolution and pooling operators shown, adopted from [Lecun et al., 1998]

using small squares of input data.

As we discussed above, every image can be considered as a matrix of pixel values. With a convolution, we slide the sliding matrix over the original image by the stride value; for every position we compute the element wise multiplication between the two matrices and then we add the multiplication outputs to get the final value which forms a single element of the output matrix. In CNN terminology, the sliding matrix is called filter or kernel and the resulting matrix is called the Activation Map or the Feature Map. In practice, a CNN learns the values of these filters on its own during the training process.

The size of the Feature Map is controlled by three parameters that we decide before the convolution step is performed:

- Depth: Corresponds to the number of filters we use for the convolution operation. Each filter produces a feature map of the resulting tensor.
- Stride: This is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time. Having a larger stride will produce smaller feature maps, while a smaller stride will use all the information available from the input features.
- Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix and preserve the image size.

8.1.2 Pooling

Spatial Pooling has the job of reducing the dimensionality of each feature map while retaining the most important information. Spatial Pooling can be of different types: Max, Average, Sum, etc. Pooling makes the feature dimension smaller and more manageable, reducing the number of parameters and computations in the

network, therefore, controlling overfitting. It also makes the network invariant to small transformations, distortions and translations in the input image.

In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window.

8.2 MobileNet

The MobileNet architecture [Howard et al., 2017] is an efficient model optimized for mobile and embedded vision applications. MobileNets are a response to the trend that recently made neural networks always deeper and more complicated to achieve higher accuracies.

8.2.1 Architecture

MobileNets are different from other architectures in their use of depth-wise separable convolutions, which allow building small and fast deep neural networks without sacrificing accuracy.

MobileNets builds up from the observation that the common convolution operation has the effect of filtering features based on the convolutional kernels and combining features to produce a new representation. These two steps can be split into two separate operations. The first operation consists in using depthwise convolutions, which apply a single filter per each input channel with the effect of filtering the input features. The second one is the application of a 1×1 convolution, with the result of creating a linear combination of the intermediate feature maps.

This results in a reduction of computation of 8 to 9 times respect to a standard convolution. Together with the reduction of computation, the model is also lighter: the model we trained weights only 26 MB.

8.2.2 Training

We trained the MobileNet networks using Stochastic Gradient Descent (SGD) with Nesterov momentum [Sutskever et al., 2013] of 0.9 without dampening. When not pretrained, weights are initialized using the weight initialization introduced by [He et al., 2015], also known as “he normal”. The learning rate schedule uses a step decay procedure, with a different initial learning rate for the pretrained case and the random initialized case. For the pretrained case, we start with lr=1e-2, and we halve the learning rate every 20 epoch for 60 epochs. For the not pretrained case, we start with lr=1e-1, and we halve the learning rate every 20 epoch for 60 epochs. The batch size is 40.

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

Figure 8.2: MobileNet Architecture, adopted from [Howard et al., 2017]

8.3 DenseNet

The DenseNet architecture [Huang et al., 2016] builds on top of the ResNet architecture [He et al., 2015]. More precisely, the dense connectivity pattern is an extension of the skip connection mechanism introduced with the ResNet architecture.

DenseNet explicitly differentiates between information that is added to the network and information that is preserved. This improves the information flow throughout the network, which makes them easy to train. Further, dense connections have a regularizing effect, which reduces overfitting on tasks with smaller training set sizes.

8.3.1 Architecture

To understand the DenseNet contribution to the neural networks architectures it is necessary to analyze the preceding architectures. A traditional CNN can be defined as a sequence of layers, each implementing a non linear transformation $F_n(\cdot)$, where

n indexes the layer. We denote the output of the n^{th} layer as X_n and the input image as X_0. In a sequential neural network the output of the n^{th} layer is the input to the following $(n+1)^{th}$ layer [Krizhevsky et al., 2012], which gives rise to the following layer transition:

$$X_n = F_n(X_{n-1}) \tag{8.1}$$

ResNets add a skip-connection that bypasses the non-linear transformations with an identity function:

$$X_n = F_n(X_{n-1}) + X_{n-1} \tag{8.2}$$

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 8.3: DenseNet Architectures, adopted from [Huang et al., 2016]

This so-called skip-connection allowed ResNets to cut consistently the training time and to reach higher accuracies. To improve the information flow between layers, in the DenseNet architecture all the layer with matching feature map sizes are connected. As a consequence, the n^{th} layer receives the feature maps of all preceding layers, $X_0, ..., X_{n-1}$, as input:

$$X_n = F_n([X_0, ..., X_{n-1}]) \tag{8.3}$$

where $[X_0, ..., X_{n-1}]$ refers to the concatenation of the feature maps produced in layers $[0, ..., n-1]$.

Motivated by [He et al., 2016], DenseNet define $F_n(\cdot)$ as a composite function of three consecutive operations: batch normalization [Ioffe and Szegedy, 2015], followed by a ReLU [Glorot et al., 2011] and a 3×3 convolution.

Pooling layers

As in the preceding works, pooling layers are added to reduce the size of the feature maps. In DenseNet each pooling layer delimits a region of the network that can be concatenated with the preceding layers. These regions are called dense blocks and they are limited by transition layers, which do convolution and pooling.

Growth rate

If each function F_n produces k feature maps, it follows that the n^{th} layer has $k_0 + k \times (n-1)$ input feature maps, where k_0 is the number of channels in the input layer. An important difference between DenseNet and existing network architectures is that DenseNet can have very narrow layers, (e.g., $k = 12$). One explanation for this is that each layer has access to all the preceding feature maps in its block and, therefore, to the network's collective knowledge.

Bottleneck layers

Even though each convolution adds only k new feature to the collective knowledge, each convolutional layer processes has to process all its input features, which ultimately grows to a large number. It has been noted in [He et al., 2015] [Szegedy et al., 2015] that a 1×1 convolution can be introduced as bottleneck layer before each 3×3 convolution to reduce the number of input feature maps, and thus to improve computational efficiency. In our experiments, each bottleneck layer produces 4k feature maps.

Compression

In the Densenet architecture, the global state continues to grow layer after layer. This eventually brings to an exploding number of features.

From the growth rate equation, if we use 121 layers, an initial number of features of 64 and a growth rate of 12, we end up with

$$64 + 12 \times (121 - 1) = 1504 \text{ feature maps}$$

With a more complex model we may have 161 layers, an initial number of features of 96 and a growth rate of 48, we would end up with

$$96 + 48 \times (161 - 1) = 7776 \text{ feature maps}$$

Those are too many, especially if we consider that at the end there is usually a fully connected layer. There is more, having a global state that large has repercussions on the speed of the network because it ends up doing huge convolutions. To improve model compactness without sacrificing network depth, it is possible to reduce the number of feature maps at transition layers. In the original work, this

the number of feature maps is multiplied by a factor θ in the range [0,1] and often set to 0.5.

8.3.2 Training

All the networks are trained using stochastic gradient descent (SGD) with Nesterov momentum [Sutskever et al., 2013] of 0.9 without dampening. When not pretrained, weights are initialized using the "he normal" initialization. The learning rate schedule uses a step decay procedure, with a different initial learning rate for the pretrained case and the random initialized case. For the pretrained case, we start with lr=1e-3, and we halve the learning rate every ten epoch for 30 epochs. For the not pretrained case, we start with lr=1e-2, and we halve the learning rate every ten epoch for 80 epochs. The batch size is 8, mainly because of GPU memory constraints.

8.4 Results

CNN have shown good results with all the datasets used for testing. Part of this is the consequence of the fact that in the document domain the intra-class variance is low, so test documents are very similar to the training ones.

An important thing to consider is the dimension of the images: experiments have shown that images smaller than 224×224 are too little, and performances are not acceptable also for the training data. This was already verified in precedent works and other domains (e.g., ImageNet).

The greatest problem is that the test set and the training set have the same capture bias, this means that the accuracy of the model may decrease in a real scenario where documents have a different capture bias.

8.4.1 Tobacco

As a first and baseline experiment, we train a Densenet-121 model, on the Tobacco-3482 dataset following the evaluation scheme described in 4.2.2. All the scores except those of the DenseNet-121 are taken from [Afzal et al., 2017].

In this case, we obtain better results respect to the other models without pretraining: the densenet model achieves 77.28% median accuracy, against the 70.28% of the GoogleNet, the best performing not pretrained model. Nonetheless, the Densenet performance is comparable with the best performing pretrained model: VGG-16, which achieves 77.52% accuracy. This experiment shows the generalizing abilities of the densenet models when trained with little datasets.

In the second experiment, we train a DenseNet-121 model pretrained on Imagenet, on the Tobacco-3482 dataset. We follow the same procedure described above. The results show improvement respect the other pretrained model, meaning that

the good properties of densenet also apply in a transfer learning scenario. As a note, we pass from 77.52% of the VGG-16 model to 83.20% of the DenseNet model.

The architectures in table 8.1 are in chronological order, and show an increasing accuracy from AlexNet to VGG-16, until the ResNet drop. It is to be noted that ResNets achieved way better results than the previous networks on the ImageNet dataset. A possible explanation is that the Tobacco dataset is limited in size, which brings the network to overfit the training set, with the consequence that results on the test set are bad.

DenseNet inverts the trend, and even though it is a more complex architecture, it can generalize better, even with few training samples. This in part confirms the claims of the original papers about a regularization due to the dense connectionism. Unfortunately, we did not have the resources to pretrain a DenseNet on the Big Tobacco dataset and then perform the transfer learning, but we can expect better results than the current networks.

For practical uses, we might still prefer a model trained on ImageNet for his higher generality, our reasoning is that the accuracy given from the Document pretraining is misleading because the RVL-CDIP dataset [Harley et al., 2015] and the Tobacco-3482 dataset have a similar generating distribution. Thus it is quite obvious that a model pretrained on the larger dataset transfers well on the little one. We have doubts that this approach is general since the original work does not provide tests on other datasets. As a proof of the similarity of the datasets, a ResNet-50 achieves 90.40% accuracy on the RVL-CDIP, while the same network pretrained on RVL-CDIP achieves 91.13 % accuracy on the Tobacco-3482 dataset.

Method	Document Pretraining	ImageNet Pretraining	No Pretraining
AlexNet	90.04%	75.73%	62.49%
GoogleNet	88.40%	72.98%	**70.28%**
VGG-16	91.01%	**77.52%**	69.50%
ResNet-50	**91.13%**	67.93%	59.55%
DenseNet-121		**83.20%**	**77.28%**

Table 8.1: Test accuracy for each architecture: bold scores are the current SoA, while the blue ones are our new SoA

8.4.2 ADMINISTRATIVE

As a first experiment, we train a MobileNet-1.0-224 model with the classic augmentation transformations, excluding the more invasive random occlusion. The results are positive, considering the small architecture: we obtain on the training and test set respectively 97.66% and 95.61% of accuracy. The fact that the accuracy on the training data is 2% higher is an index that the model slightly overfit. The obvious solution is to add the random occlusion augmentation. The accuracy of the new

model is 96.10% on the training set and 95.89% on the test set. These results confirm the fact that the random occlusion augmentation improves the robustness of the model.

At this point, we perform the same test with a DenseNet-121, with the hope that a more powerful model achieves better performances. The DenseNet model achieves 98.06% of accuracy on the test set without the random occlusion augmentation and achieves 97.75% when the random occlusion is included. From these results, it seems that the random occlusion is not effective when more powerful and robust models are used. A possible explanation of the decrease of accuracy may be given by a sample of the generated images, where the random patch can cover the whole significative part of a document, generating a wrong training sample.

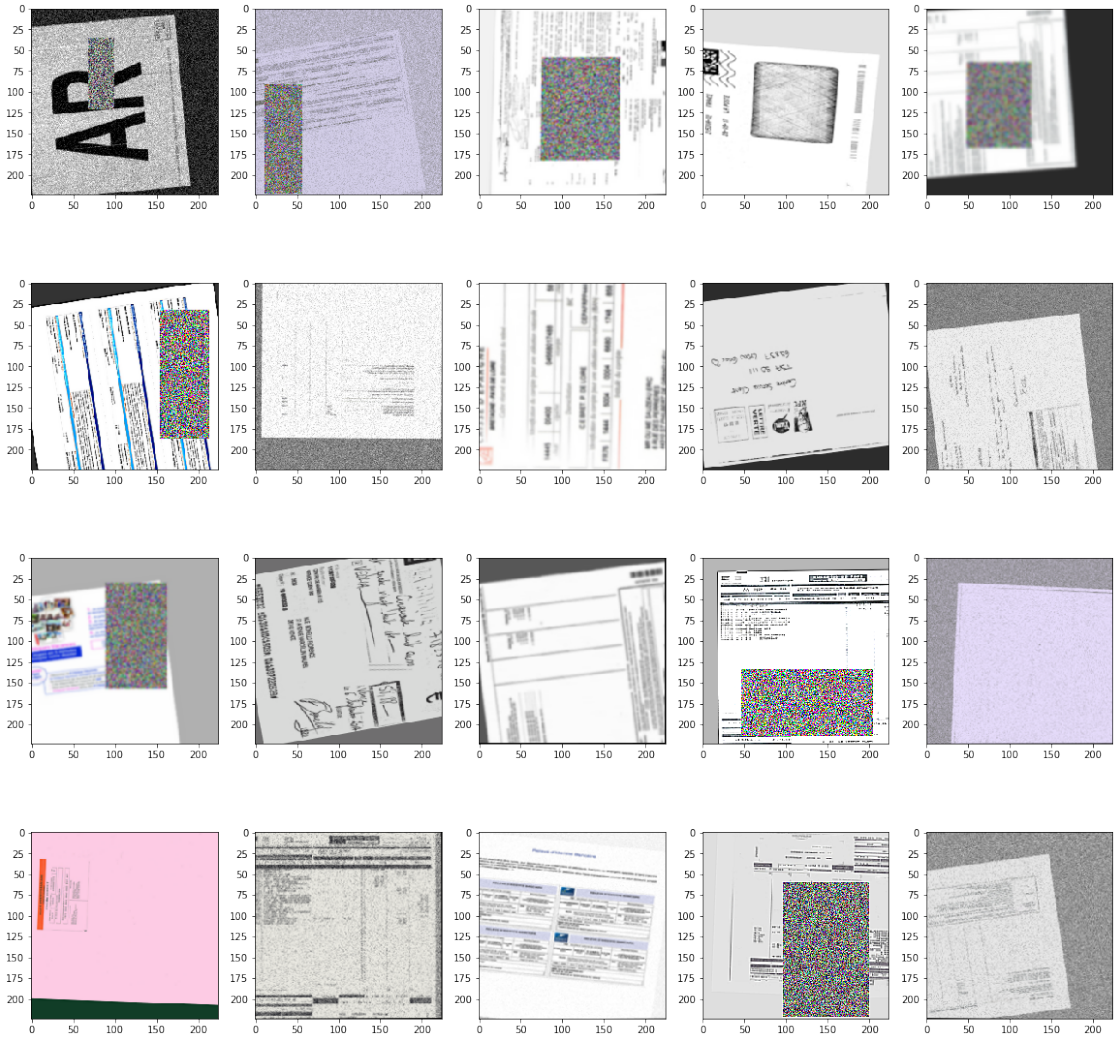

Figure 8.4: Random occlusion on ADMINISTRATVE dataset

Method	Augmentation	Augmentation + Occlusion
MobileNet-1.0-224	95.61%	95.89%
DenseNet-121	**98.06%**	97.75%

Table 8.2: Test accuracy for each architecture: bold scores are the best results

8.4.3 ENTERPRISE

The Enterprise dataset is for sure the one that suffers the most of the capture bias. Thus all our results on the test set have almost no statistical value. We managed to achieve 100% accuracy on the test set with a MobileNet-1.0-224 and a DenseNet-121, which is still impressive because the dataset is heavily imbalanced and the inter-class variance is low in some cases. As a test, we tested our model on a real picture of a document for each class of the model. The results are good, probably because we managed to take a good picture. Looking at the confidence values of the predictions, we have seen that the model can classify correctly real pictures taken in bad condition only if they are part of the classes with more samples. When we try to classify a picture of a document of a little class, it is likely that the prediction will be wrong.

Chapter 9

Text classification

This chapter presents the whole pipeline required to classify a document using the text it contains. We briefly describe the technologies to extract the text and then we present the building blocks of the text classifier we used. In conclusion, we present the results on our datasets.

9.0.1 OCR

OCR (Optical Character Recognition) technology allows the conversion of printed text into digital text that can be edited with a computer. Nowadays OCR is an essential step for all the document classifier based on the text of the documents.

The first part of an OCR engine is a complex preprocessing pipeline of the document with the objective to improve the chances of successful recognition [Optical character recognition, 2017]:

- De-skew
- Despeckle
- Binarization
- Line Removal
- Line and word detection
- Character segmentation
- Normalize aspect ratio and scale

Among the preprocessing steps, the most critical in our domain is the binarization. Binarization is the process of converting an image from color or greyscale to black-and-white. Binarization of scanned document is often successful because the resulting document shows a bimodal histogram. With pictures of a document, the

histogram is not anymore bimodal, and it is possible that the binarization process deletes some portions of text.

The second phase performs the character recognition, where pattern matching algorithms or classification algorithms are used. As an example, we use the 4.00.00 alpha version of Tesseract, which uses a Long Short-Term Memory (LSTM) engine.

9.1 FastText

FastText [Joulin et al., 2016] is a deep learning model based on the recent work in efficient word representation learning[Mikolov et al., 2013a] [Levy et al., 2015]. FastText shows that linear models with a rank constraint and a fast loss function can train on large datasets in a short time while achieving performance on par with state of the art models that takes much more time for training.

9.1.1 Architecture

The FastText classifier can be seen as a simple linear model with rank constraint. The first weight matrix A is a look-up table over the words.

The word representations are then averaged into a text representation, which is then fed to a linear classifier. Ultimately, the softmax function f is used to compute the probability distribution over the predefined classes. At this points it remains only to minimize the negative log-likelihood over the classes:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n log(f(BAx_n))$$

where x_n is the normalized bag of features of the nth document, y_n the label, A and B the weight matrices. This model is trained using stochastic gradient descent and a linearly decaying learning rate.

9.1.2 Embedding

Recent works show that it is possible to learn a dense representation of words [Mikolov et al., 2013b]. This approach differs a lot from a classical representation of words, such as One hot encoding, where the distance between each word is the same, meaning that words are independent features not related to them.

9.1.3 N-gram features

As we saw, BOW is invariant to word order. While many state of the art classifier take order into account, it requires much computation to do it. A simple alternative is to use a bag of n-grams, which allows feeding the model with local word order

information. FastText uses a fast and memory efficient mapping of the n-grams by using the hashing trick [Weinberger et al., 2009].

Code

```
    def fasttext_model(max_words, embedding_dims, maxlen):
        model = Sequential()

        # Embed the vocabulary index into a low-dimensional matrix
        model.add(Embedding(max_words,
                            embedding_dims,
                            input_length=maxlen))

        # Average the features of all words in the document
        model.add(GlobalAveragePooling1D())
        # Use a linear model for prediction
        model.add(Dense(1, activation='softmax'))
```

9.2 Results

FastText has shown impressive results on different datasets. As expected the text classifier achieves high accuracy scores because words can express the semantic category of a document.

9.2.1 ADMINISTRATIVE

Here FastText achieves 93.52% of accuracy. The main problems are hand-written documents and noisy images. There is also another problem: it fails to classify a portion of images, about 1%, because of an error in image binarization that produces blank images. We think that another 2% error is due to partial errors in binarization.

9.2.2 ENTERPRISE

The ENTERPRISE dataset contains forms, which by definitions contains the same words. FastText achieves 100% of accuracy and it is able to differentiate the words that are inserted by the user from the words of the forms, which are fundamental for the identification of the class.

Chapter 10

Ensemble model

In this chapter, we explore the techniques that allow creating a model ensemble, or in other words, a model composed of many models. We conclude presenting the results achieved by the ensemble on our datasets.

A common practice in machine learning is to use cross-validation to give an objective evaluation of each model and then select the best model. This is known as the discrete Super Learner selector [Laan and Dudoit, 2003], which performs as well as the best base model available. The idea to use exclusively one model is a constraint that we impose on ourselves, but this constraint has no practical or theoretical basis. Ensemble learning methods train several models and use some rules to combine them to make predictions. The ensemble learning methods have gained popularity because of their superior prediction performance in practice.

10.1 Classic ensembles

Classic Ensembles includes those ensemble techniques that can be expressed as a sequence of operations that do not depend on the data. The independence from the data makes this methods straightforward to apply, while maintaining good performances. In the following sections, we analyze the most popular classic ensembles.

Max

When a model gives a prediction, it usually outputs a probability distribution over all the output classes. Being a discrete probability distribution, the confidence for each class is constrained between 0 and 1, and the sum of the confidences is constrained to sum up to 1. The Maximum ensemble builds on the idea that the model which is more confident about its prediction wins. Despite being simple to implement, this technique has several weak points. The first one is that each final prediction does not consider the models together. Once the most confident model is identified, the prediction is made exclusively by the winner; this means

that it is not possible to eventually combine multiple predictions. The other weak point is the automatic reliance on the maximum confidence. This automatically advantages models that are overconfident, in other words, models that are sure on all their predictions, also the bad ones. Deep neural networks are known to be often overconfident over their predictions. This has been investigated in [Ju et al., 2017], and it has been shown that some network architectures are more overconfident than others. The results of this are that everytime that we use the Max ensemble, we have to check that one model is not too overconfident respect the others.

Majority voting

Majority voting tries to solve some of the disadvantages of the Max ensemble, such as the reliance on overconfident models and the exclusive reliance on the best model. The Majority voting favors not the most confident models, but the class that has more votes. This makes possible to output a class which is agreed by the majority of models, even though the models are not confident about it. The main disadvantages are the fact that the confidences of the models are less important than before because they are collapsed. This, together with the unweighted average favors scenarios where less accurate models that make the same prediction wins because they are more numerous.

Unweighted average

Unweighted averaging, or soft voting, takes the average of the output probabilities for all the base models and reports it as the predicted probability distribution. Naive averaging, which is largely used, is not data-adaptive and thus vulnerable to a bad selection of base models. On the other hand, this technique is often used when the base models share the same architecture and have comparable performances [Simonyan and Zisserman, 2014]. In machine learning competitions, the same network is trained multiple times, and because of the stochasticity of the training procedure, a different local minimum is obtained at every run. Doing the average of the predictions allows achieving better performances, that exploit the little differences in the neural networks. Since neural networks are similar, it makes sense to use an unweighted average, or a uniform prior, on all of them. As a note, Krizhevsky [Krizhevsky et al., 2012] won the first place in the image classification challenge of ILSVRC 2012, by averaging 7 CNNs with the same structure.

10.2 Stacked ensembles

Stacking, also called Super Learning [Van der Laan et al., 2007] or Stacked Regression [Breiman, 1996], is a class of algorithms that involves training a second-level

"meta-learner" to find the optimal combination of the base learners. Unlike bagging and boosting, the goal in stacking is to ensemble strong, diverse sets of learners together. Due to their high capacity, deep neural networks suit well for a stacked model. In the following sections, we analyze the most popular stacking techniques.

10.2.1 Weighted average

Weighted averaging can be seen as the generalization of the Unweighted average, where the Unweighted average is simply a weighted average where the weights are all equal. The concept of a metamodel that learns the optimal weights was introduced in [Ju et al., 2017] with the name of Super Learner. The Super Learner computes the ensemble weights based on the validation set because the confidence values of neural networks on the training set may be highly biased. Weighted averaging allows telling which models are more influent and which ones are less important. An advantage of the weighted ensemble is that it can mitigate the overconfident phenomenon by multiplying the output probabilities of the overconfident model by a value that scales them. Moreover, the learned weights solve the problem where there are many weak learners: the weights for the weaker models will be lower, thus meaning that they influence marginally the final prediction.

The best weights can be found either with a grid search or minimizing the negative loglikelihood of the weighted output probabilities. This can be done either by backpropagation, learning a 1x1xM kernel, or by directly solving the convex optimization problem. The convex problem can be solved easily using the SLSQP method, which allows to specify constraints on the weight, like being between 0,1 and having sum 1.

10.2.2 Classic classifiers

Stacking can be done using a common model. In our experiments, we used a Logistic Regression, SVM, and Random Forests. All the models have been used with the standard hyperparameters of scikit-learn [Pedregosa et al., 2011], except for the Random Forest Classifier where we used 100 estimators.

10.3 Results

Ensemble models have shown impressive results on different datasets. As expected the ensemble of classifiers is better of the base models, regardless of ensemble technique used.

10.3.1 Cifar-10

On the cifar dataset, we fine-tuned three different models pretrained on imagenet. The models are Inceptionv3, Xception, and ResNet50.

	Method	Accuracy
Base Models	Xception	**96.31%**
	Inception v3	95.93%
	ResNet 50	95.00%
Classic Ensembles	Max	**97.16%**
	Avg	97.11%
	Voting	97.02%
Stacked Ensembles	Weighted Avg	97.19%
	Logistic Regression	97.16%
	SVM	97.26%
	Random Forest	**97.30%**

Table 10.1: Test accuracy for base models, classic ensembles and stacked ensembles. Bold scores are the best of each category.

Starting from the base learners, we see that the Xception model achieves the highest score with 96.31% of accuracy. This was expected because of the more advanced architecture of the model. Moving to the classic ensembles that do not need any training because defined as a simple rule, the Max ensemble performs the best, with 97.16% of accuracy. In this case, the Average and Voting ensembles perform slightly worse but still better than the base models. This is probably because in this last two cases the less performing models influences more the final result. When we move to more complex ensemble models that require a training phase, we see how results improve. In this case, the Weighted Average and the Logistic Regression shows his limits respect more complex models that can exploit relations among the predictions of the single models. In this category, the best performing model is the Random Forest with 97.30% of accuracy, followed by SVM. It is important to remember that these ensemble models should be learned on the evaluation set because the base classifiers are highly biased towards the training data. At the same time, a learned ensemble model amplifies the dataset bias, meaning that it could perform worse in a real testing scenario.

10.3.2 ADMINISTRATIVE

On ADMINISTRATIVE we trained two models, a DenseNet-121 and a FastText model. In cases where the base classifiers are different and rely on different features, ensemble methods should improve even more the accuracy score.

As we have seen, the image classifier performs better than the text classifier. One might think that in this case, an ensemble would not be useful because of

	Method	Accuracy
Base Models	DenseNet	**97.75%**
	FastText	93.52%
Classic Ensembles	Max	**99.16%**
	Avg	99.15%

Table 10.2: Test accuracy on baseline and ensemble models on ADMINISTRATIVE

the performance of the text classifier is way worse than the image classifier. We have found instead that even in such cases an ensemble can improve the prediction power of the model. With the Max ensemble, the accuracy increases to 99.16%, which is 1.4% higher than the image classifier. This result is given from the fact that the two models excel in predicting different categories and by combining them the improvement is more consistent than combining similar classifiers.

Chapter 11

The black box issue

This chapter presents the problem of the interpretability of deep learning models for image classification. We continue with an example, where we apply an interpretation technique to our model, and we show the insights that it can provide to optimize various aspects of the training pipeline.

11.1 Opening the black box

Deep neural networks enable superior performance, but at the same time, their lack of decomposability into intuitive and understandable components makes them hard to interpret [Selvaraju et al., 2016]. To build trust in intelligent systems that integrate into our everyday lives, it is necessary that they are transparent, in the fashion where they explain their predictions. Transparency is particularly useful from the research point of view because allows to developers to understand the failure modes of a model[Hoiem et al., 2012].

11.1.1 Interpretability as explaination

The majority of the persons would say that humans are interpretable because they can explain their actions. This may seem completely acceptable, even though no one is aware of the exact functioning of the brain[Lipton, 2016]. One advantage of this concept of interpretability is that we can interpret black-box models after the fact, without sacrificing predictive performance creating simpler but interpretable models. These interpretations might explain predictions without showing the mechanisms by which models work, but still giving useful insights about a model for end users of machine learning. Since deep neural networks learn rich representations that can be visualized, verbalized, or used for clustering, it seems a good idea to investigate the latest findings on this topic.

Some common approaches to post-hoc interpretations include natural language explanations [Park et al., 2016], and explanations by example. Other techniques

focus on the learned representation of a model, with the hope that by understanding the building blocks, we can grasp the reasoning behind a prediction. Visualizing the convolution filters revealed the role of the first convolutional layers and the intermediate abstraction achieved in the last layers. These techniques allowed to understand the patterns that the neural network learns, but they also revealed that the deepest convolutional level looks for structures that we do not fully understand [Olah et al., 2017]. Other techniques, like the deconvnet [Zeiler and Fergus, 2013] and the guided backpropagation [Springenberg et al., 2014] focus on showing the part of the input image that influences the most the activation of the neuron relative to the class. Recently, a solution called Grad-CAM [Selvaraju et al., 2016] allowed to show the area of the input image that activates the most an output neuron.

To understand how an image interacts with a trained model, we use this last technique.

11.1.2 Grad-CAM

Previous works have shown that the deeper layers in a CNN capture higher-level structures [Bengio et al., 2012]. Furthermore, convolutional features naturally retain spatial information which is lost in fully-connected layers. For this reasons, the last convolutional layers can be seen as the best tradeoff between high-level semantics and detailed spatial information. The neurons in these layers look for semantic class-specific information in the image (e.g., parts of an object). The Grad-CAM implementation can be described in two subsequent steps. At first, the weights relative to a feature map A^k and a class c are computed using the gradient of the class c respect the feature map A^k. Logically, this operation allows understanding the importance of each neuron for a given prediction.

$$\alpha_k^c = \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{11.1}$$

Then, we perform the weighted combination of forwarding activation maps, followed by a ReLU.

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k) \tag{11.2}$$

The weighted average of forwarding activation maps shows the areas of the image that respond positively to the class c.

Grad-CAM results as a useful tool to explore what a model has learned, and it is extremely useful in scenarios where training data contains tricky cases, or when the model is good on train data but does not work on test data.

11.1.3 Grad-CAM experiments

We use the Grad-CAM method to explore the learned representations of our model on the ENTERPRISE dataset 4.2.4. We choose this dataset because it shows extremely low intra-class variance and inter-class variance. In the case where the distinguishing features are not explained, humans find very difficult to identify the categories because different classes look the same.

As a test, we use two DenseNet-121 pretrained on Imagenet. Both models have been trained with minor data augmentation, while only the second model has been trained adding the random occlusion augmentation. This test is intended to show the learned representation of the models on the training images, then how this representation interacts with test images, and to conclude we want to see how the random occlusion modify the model.

Referring to figure 12.1, the first column contains the input images, the second column contains the Grad-CAM images of the model trained without random occlusion, and the third column contains the Grad-CAM images of the model trained without random occlusion, and the third column contains the images. Regarding the rows, each pair contains the input image the training set and the picture of the same document taken with a mobile phone. The first two pairs show documents that have the same exact structure, except the center of the image. The third pair shows a document from a completely different class.

A first analysis we can do is about the differences between the training images and the respective test images. The pictures show a lower contrast, they are darker, and it is possible to see shadows and lights. Regarding the layout of the documents, all the pictures show translations and scale differences respect the training images. Moreover, all the pictures are not aligned, showing slight skewing and an inclination of the plane respect the taking device.

A second analysis we can do is about the learned representation of the two models of the training images. In all three cases it we can appreciate that the model trained with the random occlusion augmentation learns a distributed representation of the image. This was exactly our hope, and we can justify this finding with the fact that during training some parts of the images are occluded, meaning that the model has to learn all the distinguishing parts of the documents.

A third analysis is a comparison between the representation of the training images and the test images. Focusing on the first model, we can see that there are only slight changes between train and test images. On the other side, we see that when we apply the random occlusion, the representations are entirely different between training and test. This phenomenon is not easy to explain because models have shown to react in mysterious ways to slight changes on the input. One possible explanation might be that the second model did not have enough time to train. Both models have been trained for two epochs because after that they reached 100% of accuracy, but it is possible that the second model could benefit from

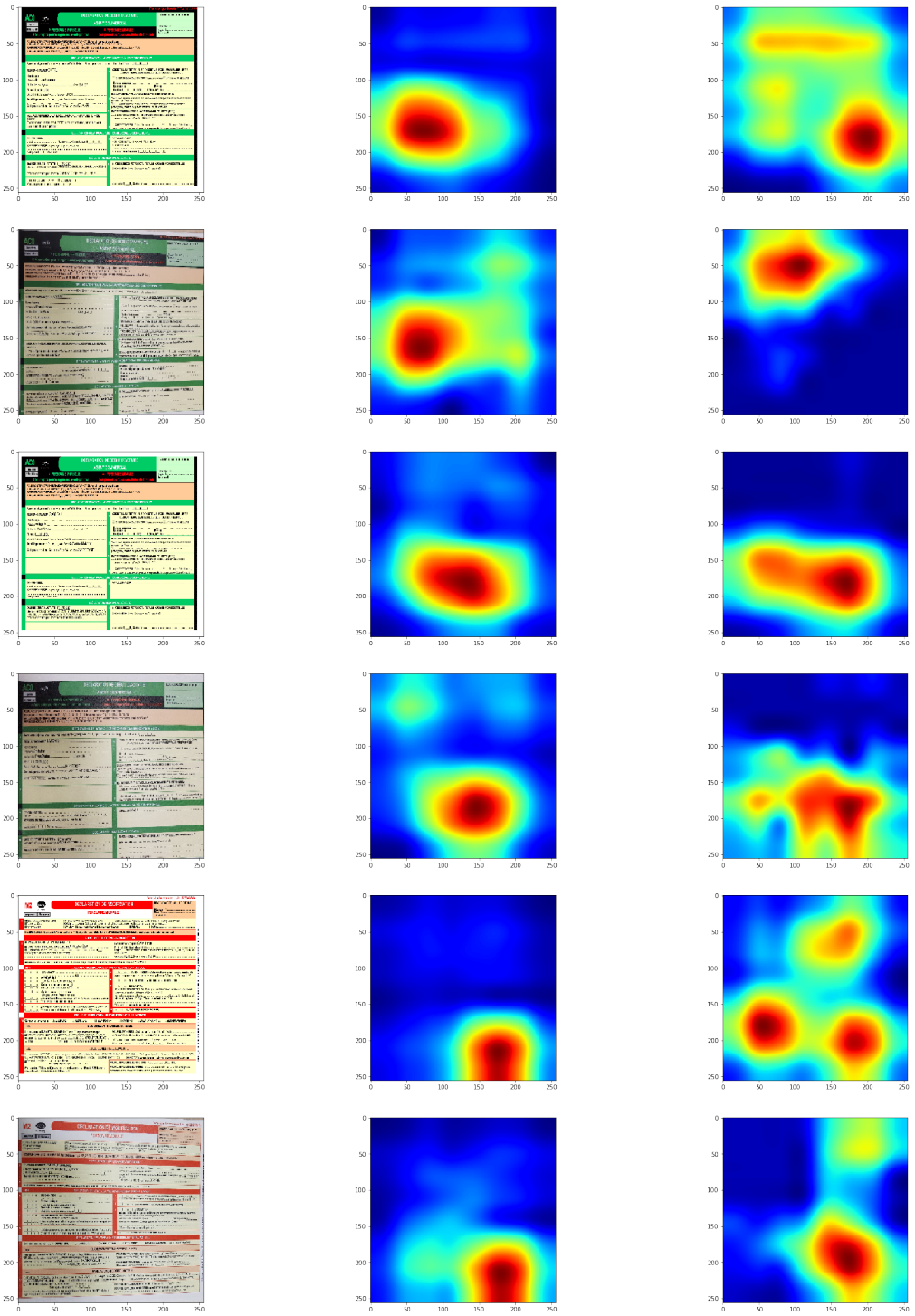

Figure 11.1: Test for no data augmented model

further training.

Ultimately, we can state that from what we have seen the random occlusion learns a distributed representation of the documents that seem not to be robust to changes in the image. On the other side, the model trained with the classic augmentation learns a simpler representation of the documents that seems more robust to changes in the documents.

Chapter 12

Deployment

In this chapter, we explore the different deployment scenarios, tools, and libraries. Then we show a real demonstration of the steps needed to deploy a model in a mobile application. We conclude showing the performances of different models and with an analysis of the insights that a mobile demonstration can provide to the model definition.

12.1 Scenarios

While for the training of a deep learning model is necessary the use of GPUs for time constraints, the inference can happen in way more diverse conditions that include servers, mobile applications, and embedded devices.

- Server: In this scenario, the server often has a REST API that provides inference. The inference can be performed either using the CPU or the GPU. It is to be noted that for inference the CPU is an excellent candidate in that situation where the CPU capacity is not completely used.

- Mobile: Mobile phones specifics continue to evolve, and every year performances of CPU and GPU increases. In this context, it is possible to run some small models in real time using the GPU of the mobile phone. Depending on the operating system, it may be better to use a framework than another; in our case we use CoreML on iOS and TensorFlow on Android.

- Embedded: In this case, special inference chips like the Intel® Movidius™ are used. These chips are optimized for power consumption, meaning that the inferring performances are low and only small models can be used.

12.2 Mobile deployment

The mobile application is intended to be used as a demo, showing the potential of the trained model without requiring the setup of a demonstration server and client.

12.2.1 Model conversion

Since we use different libraries for training and deploying the model, it is necessary to perform a conversion of the trained model from Keras to CoreML. Model conversion between different libraries is not always straightforward because different libraries may have some implementation differences, or some operations may not be implemented in the deployment framework. Regarding this aspect, it should be investigated since the conception of the model. A model that is not deployable has no practical utility, thus meaning that all the time spent in his development would be wasted. In our case, we used well established models that are composed of basic components that are implemented in the majority of the frameworks, thus making the conversion straightforward.

From the practical point of view, the model conversion is done using coremltools, a python library developed from Apple for converting and testing CoreML models.

```
1
2  coreml_model = coremltools.converters.keras.convert(model,
3      input_names = 'image',
4      image_input_names = 'image',
5      image_scale = 2./255,
6      red_bias = -1.0,
7      green_bias = -1.0,
8      blue_bias = -1.0,
9      class_labels = class_labels,
10     output_names = 'classLabelProbs')
11
12 coreml_model.save('DenseNet_{}.mlmodel'.format(image_size))
```

Listing 12.1: conversion of a model from Keras to CoreML

The conversion is straightforward, and from 12.1 we can see that the CoreML model packages not only the model but also the class labels and the preprocessing operations that are expressed as scale and bias. The conversion starts with the preprocessing operations, which in the original model are performed offline respect to the inference. From the deployment point of view, packaging the the preprocessing operations in the model allows to deploy the model easier, since in this way it can operate directly on the input image. Another consideration to be done is respect to the fact that a separation between model and preprocessing is not optimal because the model can not work without the associated preprocessing, which is specific to the model itself. For this reasons, we decide to use an option of coreml that allows

to insert a preprocessing phase in front of the model. In our case we set the image_scale parameter to 2./255 and the bias channels to -1.0. In this way we match perfectly the preprocessing operations described in 6.1.3.

Once the preprocessing phase is converted, the converter converts the model layer after layer. In order to interact with the model, it is necessary to specify a name to associated with the input layer and the output layer. Ultimately, the class labels can be packaged with the model. Like with the preprocessing phase, it makes sense to put the labels in the model and simplify the design of the application.

12.2.2 Mobile Application

The mobile application has the objective to provide an easy to use demonstration of the classification capability of the trained model. For such a simple task, an application that takes a picture and then classifies it would be enough. Actually, since another goal of the project is to provide real time classification, we think that demonstrating it on a computationally limited device would be perfect. To do this, we develop an application that classifies in real time the video output of the camera.

The application can be described as a single view application that shows the video feed from the front camera with the label of the predicted class for the current frame. The application shares the same structure of [yulingtianxia, 2017], with the only difference being the model used for the inference. The core of the application is the model, which is either the DenseNet-121 trained on the ENTERPRISE dataset without doing the random occlusion augmentation or the MobileNet-1.0-224.

Once a frame is isolated, it is resized accordingly and then the classification is handled exclusively by the CoreML API. The CoreML model exposes a class that allows to perform inference requests. The CoreML model performs all the preprocessing operations that we specified during the model conversion: scaling and bias subtraction. At this point the model performs the classification using the GPU of the device, and finally the most probable class label is shown with the associated confidence.

Results

The main metrics we can use to describe the application performances are the accuracy and the inference time. Regarding the accuracy, we have seen that the DenseNet-121 model performs slightly better than the MobileNet-1.0-224 model. This was expected, but we do not have any metric available to make a direct comparison since this is the result of a demo.

Linked with accuracy, we can perform an analysis on the consequences of capture bias on the classification. Generally, we can state that the augmentations allows the model to work on the field, where light is not uniform, color balance is different

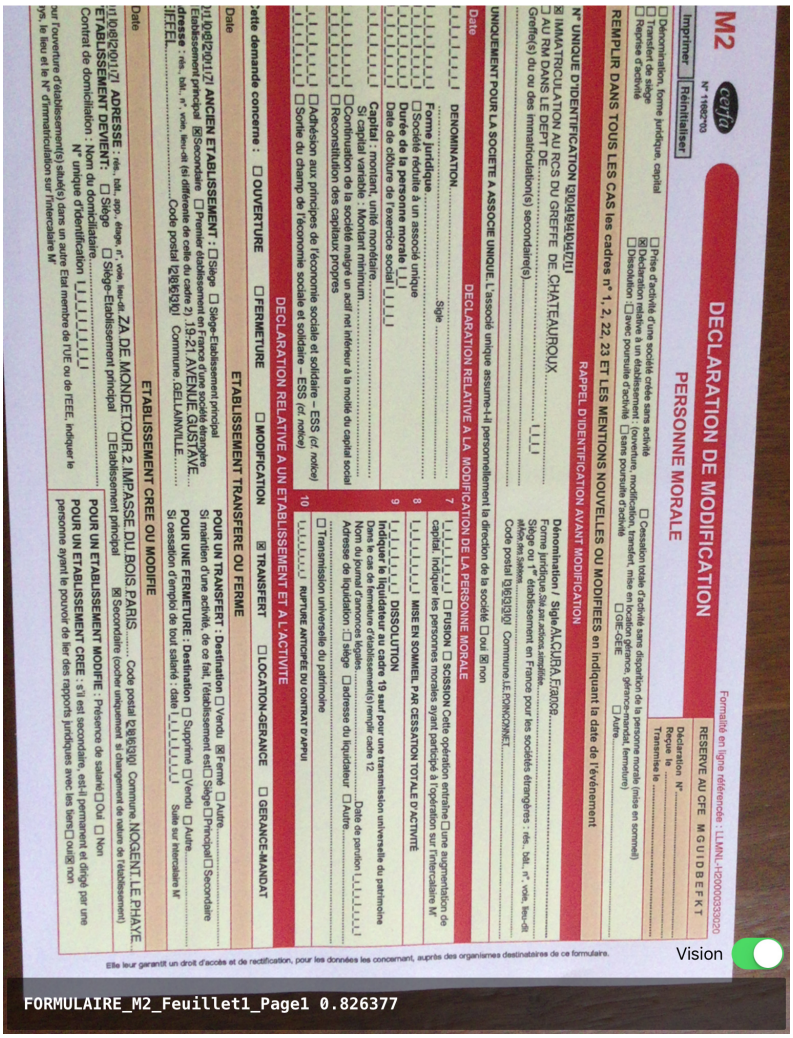

Figure 12.1: View of iOS demo: the predicted class for the current frame is showed at the bottom

and the quality of the image is generally worse. The model showed to perform well also when the image did not match the entire picture, which is particularly important. At the same time, the model performed better when the image was centered and more similar to the training conditions.

The main problem we have noted is with respect of the imbalance of the classes. If it is true that the model is able to classify perfectly also the training images of the infrequent classes, this does not holds true for the real images. More specifically, we have seen that with the frequent classes the model is able to sustain a strong capture bias, while with the infrequent classes the model is more fragile. We believe that this is due to the fact that the model sees many more augmentations of the frequent classes respect those of the not frequent ones. In this optic, a simple rebalance of

the dataset or a longer training may be enough to correct the phenomenon.

During our tests we have seen that the capturing resolution is crucial. With a resolution of 1080p we obtain an image that then is resized and often correctly classified. On the other hand, when we use the 720p resolution the accuracy falls. From this it is clear that the the noise introduced with the 720p video has a negative effect on the model. This is extremely interesting because it shows that even if resized, the original image quality is a fundamental component of the classification model. To fix this phenomenon a more detailled analysis on image quality and noise should be performed.

With respect to the inference time, we measured that the DenseNet model is able to perform 4 fps, while the MobileNet performs 12 fps on an iPad Air 2.

In conclusion, the demo application shows that the model can handle documents that present a severe capture bias. At the same time, we have seen that also complex models can infer in a short time and that the image quality is fundamental.

Chapter 13

Conclusions and future work

In this final chapter we compare our initial objectives with the obtained results. We also discuss the improvement that we plan for the future of LEIA classifier, explaining our perspectives after our study, implementation, and testing of the system.

13.1 Objectives and findings

We started this thesis by taking into account the problem of the document classification on the everyday business process. We designed the LEIA Document Classifier, a system that can classify documents using the latest findings of machine learning. Starting with the annotation pipeline, we managed to build an annotation process that can cut consistently annotation times of datasets that shows visible clusters, like the case of document datasets.

Moving to the classification models, using a deep learning approach and a solid augmentation pipeline we managed to define a model that does not need any change in his definition when data change. Moreover, we reach almost human performances regardless of the difficulties of the dataset and the models are small enough to run seamlessly on smartphones. Regarding the deployment, the image classifier can work directly with the image of a document, without requiring dependencies like OpenCV.

Speaking of our findings, we have seen that it is not necessary to use large models for the document classification task, mainly because of the low intra-class variance. On the other hand, the collection of documents has proven to be the most important phase, mainly because of capture bias and temporal bias.

Data augmentation has proven to be fundamental, especially in cases where the capture bias was important, like the ENTERPRISE dataset. Neural Network architectures proved to be more important in cases where the training samples were limited and complex, like the Tobacco-3482 dataset. Even more importantly,

a good input image has proven to be the key for a good classification. This refers to the image quality and the image size.

Regarding the different classification models, we have seen surprisingly that the image classifier performs better than a text classifier. We think that a major cause of this phenomenon is related to the failures of the text extraction phase.

13.2 Future work

This work has shown the process that in 6 months allowed to annotate a dataset, and design, train and deploy a document classifier. Regardless of the work done there are many areas of improvement.

The first area of improvement is the annotation pipeline, and the first step to do it is the definition of an evaluation procedure. Because of time requirements we have not performed a proper evaluation of the annotation pipeline that we created. In order to improve the whole annotation pipeline it is necessary to define a set of benchmarks composed of different annotated dataset in the document domain. Once the dataset are defined, it would be possible to perform proper evaluations of different components of the pipeline thanks to the available labels. Once it is possible to evaluate the annotation phase, we could test all the components of the annotation pipeline:

- Analysis of CNN feature extractors on documents: We used the penultimate layer of a Xception network as a feature extractor for images, but we have not performed any test using different layers and architectures. In this optic,

- Combination of image and text features: In this work we have shown that depending on the document types, image or text based features can be used for the clustering. What we have not explored is the use of both kind of features, which has the possibility of leading to even butter clusters.

- Analysis of clustering algorithms: After an analysis of the problem we decided to use the HDBSCAN algorithm and it showed good results. Nontheless we did not perform any test regarding different clustering algorithms and we could find better results.

- Fine Annotation improvement: The fine annotation phase allowed us to correct some wrong annotation and minimize the documents to annotate manually. Nonetheless the whole procedure could be refined with a more complex decision function. For example, the annotated documents below a given confidence threshold should be annotated manually, and the thresholds should be evaluated.

A second area of improvement regards the data augmentation of documents. As we said, documents have many characteristics that we could exploit more thoroughly to generate new images and achieve better performances. In each augmentation we specified possible improvements, but we can create more complex augmentation techniques with the use of 3d computer vision techniques.

To conclude, the future research directions in document classification have a lot in common with the more general research in image classification and deep learning.

- Deep learning presumes a stable world: Common systems do not generalize to novel samples and this makes the deployment of deep learning based solutions difficult in ever-changing scenarios. This problem can partially be solved through continuous training, but One-shot learning will be essential to solving this problem.
- Deep learning is resources expensive: In the document domain, we have noted that the quality and size of the image is the key for a correct prediction. The consequence of this is that in order to improve the classification accuracy without increasing the training time, more powerful GPUs are needed.
- OCR for hand-written documents: OCR solutions have shown to be slow and unable to work with handwritten documents. Better solutions would allow us to integrate image and text classifiers also in real times systems.

Bibliography

[Afzal et al., 2017] Afzal, M. Z., Kölsch, A., Ahmed, S., and Liwicki, M. (2017). Cutting the error by half: Investigation of very deep CNN and advanced training strategies for document image classification. *CoRR*, abs/1704.03557.

[Aggarwal et al., 2001] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 420–434, London, UK, UK. Springer-Verlag.

[Babenko et al., 2014] Babenko, A., Slesarev, A., Chigorin, A., and Lempitsky, V. (2014). *Neural Codes for Image Retrieval*, pages 584–599. Springer International Publishing, Cham.

[Bengio et al., 2012] Bengio, Y., Courville, A. C., and Vincent, P. (2012). Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538.

[Beyer et al., 1999] Beyer, K. S., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 217–235, London, UK, UK. Springer-Verlag.

[Breiman, 1996] Breiman, L. (1996). Stacked regressions. *Mach. Learn.*, 24(1):49–64.

[Campello et al., 2013] Campello, R. J. G. B., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Chollet, 2016] Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357.

[Chollet, 2017] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Company.

[Conneau et al., 2016] Conneau, A., Schwenk, H., Barrault, L., and LeCun, Y. (2016). Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407.

[Dodge and Karam, 2016] Dodge, S. F. and Karam, L. J. (2016). Understanding how image quality affects deep neural networks. *CoRR*, abs/1604.04004.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press.

[Ganchev and Dredze, 2008] Ganchev, K. and Dredze, M. (2008). Small statistical models by random feature mixing.

[Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

[Gonzalez et al., 2016] Gonzalez, D. M., Volpi, M., and Tuia, D. (2016). Learning rotation invariant convolutional filters for texture classification. *CoRR*, abs/1604.06720.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Hahnloser et al., 2000] Hahnloser, R. H. R., Sarpeshkar, R., Mahowald, M., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405 6789:947–51.

[Harley et al., 2015] Harley, A. W., Ufkes, A., and Derpanis, K. G. (2015). Evaluation of deep convolutional nets for document image classification and retrieval. *CoRR*, abs/1502.07058.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks. *CoRR*, abs/1603.05027.

[Hoiem et al., 2012] Hoiem, D., Chodpathumwan, Y., and Dai, Q. (2012). Diagnosing error in object detectors. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part III*, ECCV'12, pages 340–353, Berlin, Heidelberg. Springer-Verlag.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

[Huang et al., 2016] Huang, G., Liu, Z., and Weinberger, K. Q. (2016). Densely connected convolutional networks. *CoRR*, abs/1608.06993.

[Hubel and Wiesel, 1962] Hubel, D. and Wiesel, T. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Jégou et al., 2012] Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., and Schmid, C. (2012). Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716.

[Joachims, 1998] Joachims, T. (1998). Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK. Springer-Verlag.

[Joulin et al., 2016] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

[Ju et al., 2017] Ju, C., Bibaut, A., and van der Laan, M. J. (2017). The relative performance of ensemble methods with deep convolutional neural networks for image classification. *CoRR*, abs/1704.01664.

[Jung et al., 17] Jung, A. et al. (2017–). Imgaug: Image augmentation for machine learning experiments. [Online; accessed <today>].

[Kanazawa et al., 2014] Kanazawa, A., Sharma, A., and Jacobs, D. W. (2014). Locally scale-invariant convolutional neural networks. *CoRR*, abs/1412.5104.

[Kang et al., 2014] Kang, L., Kumar, J., Ye, P., Li, Y., and Doermann, D. (2014). Convolutional neural networks for document image classification. In *2014 22nd International Conference on Pattern Recognition*, pages 3168–3172.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

[Kochi and Saitoh, 1999] Kochi, T. and Saitoh, T. (1999). User-defined template for identifying document type and extracting information from documents. In *Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fifth International Conference on*, pages 127–130.

[Krizhevsky et al.,] Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research).

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[Kumar and Doermann, 2013a] Kumar, J. and Doermann, D. (2013a). Unsupervised classification of structurally similar document images. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1225–1229.

[Kumar and Doermann, 2013b] Kumar, J. and Doermann, D. S. (2013b). Unsupervised classification of structurally similar document images. *2013 12th International Conference on Document Analysis and Recognition*, pages 1225–1229.

[Kumar et al., 2014] Kumar, J., Ye, P., and Doermann, D. (2014). Structural similarity for document image classification and retrieval. *Pattern Recognition Letters*, 43(Complete):119–126.

[Laan and Dudoit, 2003] Laan, M. J. V. D. and Dudoit, S. (2003). Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples.

[Lazebnik et al., 2006] Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2169–2178, Washington, DC, USA. IEEE Computer Society.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[Lewis et al., 2006] Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., and Heard, J. (2006). *Building a test collection for complex document information processing*, volume 2006, pages 665–666.

[Li et al.,] Li, F.-F., Karpathy, A., and Johnson, J. Cs231n: Convolutional neural networks for visual recognition 2016.

[Lipton, 2016] Lipton, Z. C. (2016). The mythos of model interpretability. *CoRR*, abs/1606.03490.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA. IEEE Computer Society.

[McKinney, 2010] McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

[Olah et al., 2017] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

[Optical character recognition, 2017] Optical character recognition (2017). Optical character recognition — Wikipedia, the free encyclopedia. [Online; accessed 26-February-2017].

[Pang and Lee, 2008] Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135.

[Park et al., 2016] Park, D. H., Hendricks, L. A., Akata, Z., Schiele, B., Darrell, T., and Rohrbach, M. (2016). Attentive explanations: Justifying decisions and pointing to the evidence. *CoRR*, abs/1612.04757.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Perronnin et al., 2010] Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 143–156, Berlin, Heidelberg. Springer-Verlag.

[Razavian et al., 2014] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382.

[Rolnick et al., 2017] Rolnick, D., Veit, A., Belongie, S. J., and Shavit, N. (2017). Deep learning is robust to massive label noise. *CoRR*, abs/1705.10694.

[Rousseeuw, 1987] Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65.

[Selvaraju et al., 2016] Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391.

[Shridhar, 2017] Shridhar, K. (2017). How to version control your machine learning task. [Online; posted 20-July-2017].

[Sicre et al., 2017] Sicre, R., Montaser Awal, A., and Furon, T. (2017). Identity documents classification as an image classification problem. In *ICIAP 2017 - 19th International Conference on Image Analysis and Processing*, ICIAP 2017: Image Analysis and Processing - ICIAP 2017, pages 602–613, Catania, Italy. Springer.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[Smith, 2007] Smith, R. (2007). An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, pages 629–633, Washington, DC, USA. IEEE Computer Society.

[Springenberg et al., 2014] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806.

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org.

[Szegedy et al., 2015] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567.

[Søgaard et al., 2014] Søgaard, A., Plank, B., and Hovy, D. (2014). *Selection bias, label bias, and bias in ground truth*, pages 11–13. Association for Computational Linguistics.

[Torralba and Efros, 2011] Torralba, A. and Efros, A. A. (2011). Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528.

[UIKit, 2017] UIKit (2017). Ui view content mode. `https://developer.apple.com/documentation/uikit/uiviewcontentmode`. Accessed: 2018-03-12.

[Van der Laan et al., 2007] Van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007). Super learner. *Statistical applications in genetics and molecular biology*, 6(1).

[van der Walt et al., 2011] van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

[Weinberger et al., 2009] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA. ACM.

[Xu et al., 2014] Xu, Y., Xiao, T., Zhang, J., Yang, K., and Zhang, Z. (2014). Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369.

[Yu et al., 2015] Yu, F., Zhang, Y., Song, S., Seff, A., and Xiao, J. (2015). LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365.

[yulingtianxia, 2017] yulingtianxia (2017). Core-ml-sample. `https://github.com/yulingtianxia/Core-ML-Sample`. Accessed: 2018-02-12.

[Zeiler and Fergus, 2013] Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901.

[Zhang et al., 2015] Zhang, X., Zhao, J. J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626.