UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Development of an OCR-based text recognition solution using Deep Learning

Raffaele Perini
2019

TRABAJO DE FIN DE MASTER

Título:	Development of an OCR-based text recognition solution using Deep Learning
Título (inglés):	Development of an OCR-based text recognition solution using Deep Learning
Autor:	Raffaele Perini
Tutor:	Yihwa Kim
Ponente:	Carlos A. Iglesias
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Development of an OCR-based text recognition solution using Deep Learning

Raffaele Perini

2019

Abstract

In this project, a model able to do optical character recognition with a deep learning approach is implemented. The starting point is the analysis and exploration on the current state-of-the-art in computer vision. Following the structure of a Neural Network is proposed, analysed and implemented. Finally this model is tested with several use cases.

Keywords: Optical Character Recognition, Neural Network, Convolutional Neural Network, Recurrent Neural Network, Connectionist Temporal Classification

Acknowledgement

This project is started and has been partially developed in Taiger España company (https://taiger.com/). A special thanks to Yihwa Kim which helped in the establishment of this project and in the creation of this document. A special thanks also to Prof. Carlos A. Iglesias, which supervised the project and helped over the entire university year. A thanks also to the European Institute of Innovation and Technologies (https://www.eitdigital.eu/), which managed and funded the entire Master's Degree. Finally, I want to thanks also my family which supported me in all my academic years.

Contents

CHAPTER 1

Introduction

In this chapter it is explained the context of this project and its goals.

1.1 Context and Introduction

This project is an extension of a work, the author made for Taiger España company (`https://taiger.com/`).

Figure 1.1: Taiger logo

The work consists in the participation in a development processes of an R&D project within the company, from its conception as a proposal to its dissemination or commercialization, through technical execution and administrative thereof. This R&D project consist essentially in a text analytic platform, aimed mainly to document processing, with several components, starting from scanning, optical character recognition, check-boxes recognition, and many other components dealing with automatic information acquisition on one hand, and natural language processing based techniques to enrich this information, such as sentiment analysis, named entity recognition, co-reference Resolution and others on the other hand.

The functionalities of this platform are mainly decouple from one another, this also applies to the optical character recognition which is composed by two smaller parts, a word detector, in charge to identify words within documents and draw bounding boxes on those, and the actual optical character recognition to which these words are fed, one by one, in order to obtain the text. The central component of this project starts from replace this optical character recognition with a new one, implementing the latter with using a deep learning approach. The change was due to the dearth of accuracy of that model in conditions different from the optimal one, i.e. very clean and noise-less documents.

Moreover, since this project remains under an R&D context, it is worth, if not necessary, to explore the current state-of-the-art as regards the treated topic, deep learning for computer vision, in order to explore the actual research status and exploit the potentially of new technologies. Finally, considering also the business nature of the company a strategic benchmark on similar projects available on the market is needed to achieve a potential future productionization.

1.2 Project goals

Given the context of this project, its objectives can be summarized as follows:

- Know the state of the art of Deep Learning for computer vision processes.
- Know the state of the art of Text Analytic platforms that are commercialized.
- Understand how Optical Character Recognition works with a deep learning approach.
- Implement a model able to do OCR with high accuracy on different use cases.

This work is naturally divided into two parts, on one hand there is a part mostly theoretical, focused on understanding what is the current state-of-the-art in deep learning for computer vision, going then more and more specifically in the analysis of OCR-based models passing through the discussion of text-analytic platform, while on the other hand there is the implementation of a system able to effectively do OCR.

Since this project is meant to be included in a bigger one, which is a text analytic platform, it is worth not only to benchmark other similar products available in the market but also to understand how the latest models and studies can be used and exploited. That is why it is important for this project to analyze these areas even though they do not seem specifically related to the problem.

The final goal of this work is to implement a system able to do OCR on words using a deep learning approach. This solution must be decoupled from the text analytic platform and the Word-detector solution, and it should be as independent as possible, in order to make it reusable in others projects and different situations from the one originally planned.

CHAPTER 2

State of the Art

Introduction of the state of the art in the context of deep learning for computer vision.

2.1 Deep Learning for Computer Vision

In this section it is analysed and explained the current computer vision state-of-the-art from a Deep Learning perspective. Starting from a brief introduction to the main works and studies which have been leading the research over the last years, this section goes through a general view and finally analyses specifically in the Opticial Character Recognition fields.

2.1.1 Introduction to Neural Network for Computer Vision

Before going through any specific actual state-of-the-art work and research regarding deep learning for computer vision, text detection and optical character recognition, it is opportune to consider some of the most relevant and popular studies which helped to strongly increase the effectiveness of deep learning for computer vision and object recognition in the last decades.

In the last ten years Neural Networks and especially Convolutional Neural Networks have become more and more popular and they are widely used in many domains but the idea is slightly older and it has been developing and studying since the end of last century.

One of the first and most popular Convolutional Neural Network is **LeNet-5** by LeCun et al.[26], it was published in 1998 and it was able to classify hand-written digits from 32x32 pixel grey-scale input images. The model required high computational power, so this has limited its effectiveness and impact at that time and even now the model is not widely used, since it requires to add more convolutional layers in order to take in input higher resolution images.

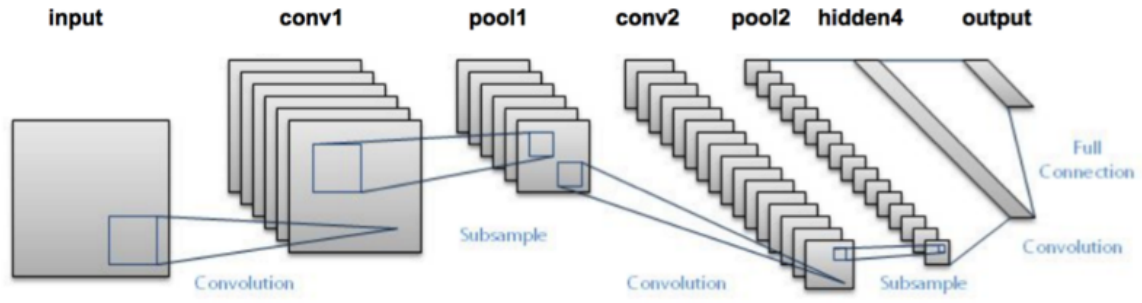

Figure 2.1: The structure of LeNet-5, starting from the input image, to a sequence of convolutional and pooling layers, to a final fully connected layer which leads to the output layer.[26]

Another famous architecture and one of the first ConvNet which has started showing the effectiveness of neural networks, is the so-called **AlexNext** by Krizhevsky et al.[25]

this model consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax for a total of 60 million parameters and 650,000 neurons. It also provides non-linearity attaching ReLU activations after every convolutional and fully-connected layer. It was one of the first trained by GPUs which made the training faster, compared with the models of the previous years. To reduce over-fitting it implements a regularization method called *dropout*[47] in the fully-connected layers that is proved to be very effective. This architecture was published in the 2012 and it significantly outperformed all the prior competitors, in fact it won the ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**) challenge,[43] in which several software compete to correctly classify and detect objects, by reducing the top-5 error from 26,2 to 15.3.

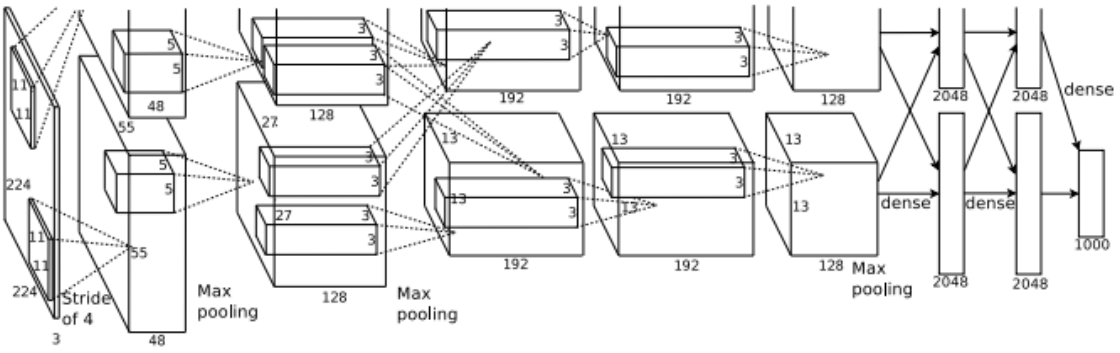

Figure 2.2: AlexNet structure. It is made by five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers. It is divided into two parts, mainly because it has been run in parallel on two GPUs.[25]

The previous model has been developed mainly to solve the problem of *object classification*, which provides for correctly classify an image into a definite class, for instance trying to tell, given a set of dogs images, which breed each dog belongs to. Usually a classification problem looks for one or several but still finite-number objects in an image. A different issue derives from the *object detection* problem, whereby a model tries to draw a bounding box around the object of interest to locate it within the image. In the object detection, there could be many bounding boxes representing different objects of interest within the image, so the effort is made to correctly detect all of them. Using a model such as AlexNet, to try to solve the detection problem may be tricky because the number of occurrences of the objects of interest is not pre-defined and fixed. One approach may be diving the image into small regions, and try to locate a object for each of them, but since there could be several spatial locations and different aspect ratios in which a object is located, this lead to the selection of a big, potentially infinite, number of regions.

To solve the problem, Ross Girshick et al. proposed to select only 2000 regions from the

image, called them region proposals in their model called **R-CNN**[15], in which R means Region. To select these regions a selective algorithm is used, in which it first generates several candidate regions using an initial sub-segmentation, then it combines regions similar among each other using a greedy algorithm. These regions are then used to detect the 2000 region proposals, after that each region proposal is fed into a ConvNet which acts as a feature extractor with a 4096-dimensional feature vector as output, finally this features are given into an Support Vector Machine, which identifies the presence of the object within a region proposal.

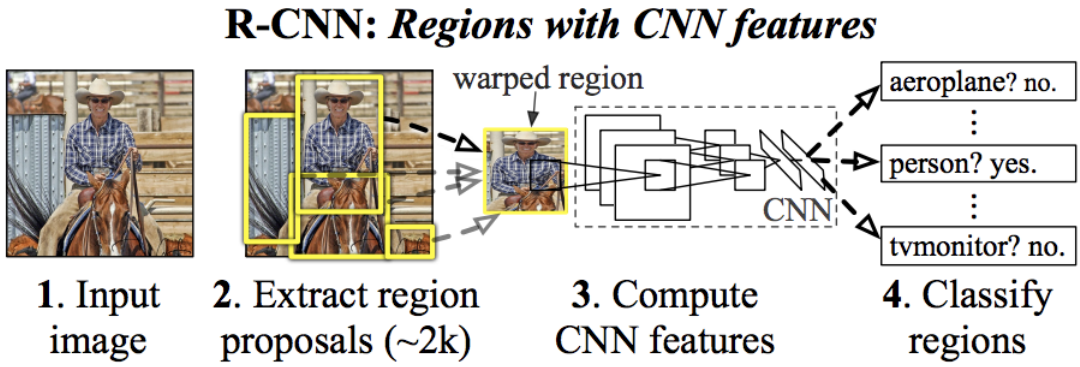

Figure 2.3: R-CNN model. It selects 2000 regions then applies feature extraction to each region to try to identify objects.[15]

There are two main problems with this implementation, first it is considerably slow and therefore cannot be applied real-time and moreover the selective algorithm is fixed, hence no learning is in place.

The years after, 2015, the same author, Ross Girshick, proposed a new model, **Fast R-CNN**[14] where he increased the performance of the previous R-CNN solving some of its disadvantages. Compared to previous work, Fast R-CNN brings several innovations to improve training and testing speed and also to increase detection accuracy. Instead of using a fixed selective algorithm, the image is fed directly into the convolutional neural network to generate a convolutional feature map, from where the feature of proposal are selected. Consecutively, using a Region-of-Interest (RoI) pooling layer, the model is able to reshape and wrap this features and feed them into a fully-connected layer, which in turn, feeds a RoI feature vector, from where, for each RoI, the model uses a softmax layer to identify the output class and the offset values of the bounding box. As also the paper claims, compared to previous work, fast R-CNN is 9-times faster than R-CNN at training time and 213-times faster at test-time.

Despite the great improvement both the previous "R-CNN" models deal with bottlenecks

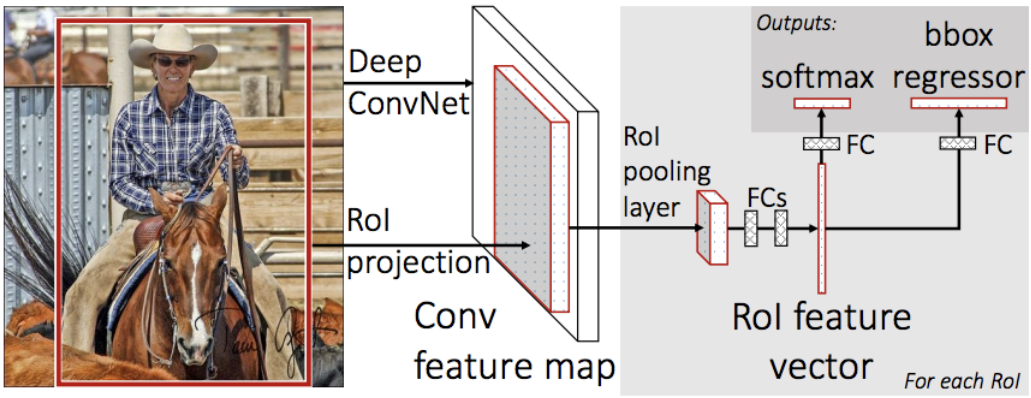

Figure 2.4: Fast R-CNN model. the image is fed directly into the convolutional neural network to generate a convolutional feature map, from where the feature of proposal are selected.[14]

due to the selective search, this kind of technique is a slow and time-consuming process affecting the performance of those networks. In their work, Shaoqing Ren et al. published **Faster R-CNN**[42] introduced a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network which predicts object bounds and objectiveness scores at each position both at the same time, it is also trained end-to-end to generate region proposals, which are used by Fast R-CNN for detection, in such a way the RPN component tells the network where to look.

2.1.2 General View

In the previous subsection it has been discussed about the most notable models which have somehow helped the research in this really fast evolution it has had and still has, especially in the object recognition field. In this section instead the analysis is moved over a wider view. It is necessary to point out Computer Vision is a vast field and covers many sub-fields, i.e. Object Detection, Semantic Segmentation, Image Classification, Image Generation, Facial Recognition and Modelling, Action Recognition, Emotion Recognition and so on. It would be impossible to cover all of them so in this section it will be discussed about the most avant-garde studies in the Deep Learning for Computer Vision, always keep in consideration the current state-of-the-art.

The models introduced in the previous subsection are still very popular nowadays, in fact many of the current more advanced models such as **SNIPER - Efficient Multi-Scale Training**[46], and also the current state-of-the-art model in object detection **Tri-**

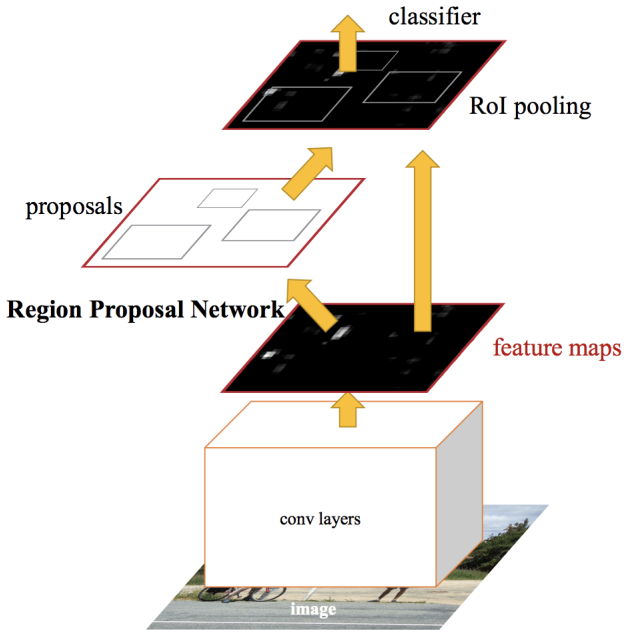

Figure 2.5: Faster R-CNN model. This models tries to overcome the bottlenecks of the selective search introducing a region proposal network, RPN.[42]

dentNet[28], are strongly based on those. Trident, by Yanghao Li et al. is a parallel multi-branch architecture in which each branch shares the same transformation parameters but with different receptive fields. This model has reached the state-of-the-art with the **COCO** database[31]. This database has been made with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding.

The models discussed in the previous paragraphs can also be applied to other computer vision fields, for instance, recently Google in collaboration with the university of Michigan, came out with **TAL-Net**, by Yu-Wei Chao et al. in their paper *Rethinking the Faster R-CNN Architecture for Temporal Action Localization*[8]. The general idea of this paper is to adapt the well-known model Faster R-CNN, in order to identify and categorize actions from videos, by introducing a new method to identify and exact location within a video where a given action occurs. This network gets a sequence of input frames from a video and it converts them into a sequence of 1D feature map. This map is passed to a segment proposal network that generates candidate segments. This input is fed to a Deep Neural Network which applies the representations learned for each candidate segment which will give a score to them. The challenge of recognizing these actions belongs to the field of computer vision known as temporal action localization; this model achieved state-of-the-art performance for action proposal and localization on THUMOS'14 challenge[23]. Moreover

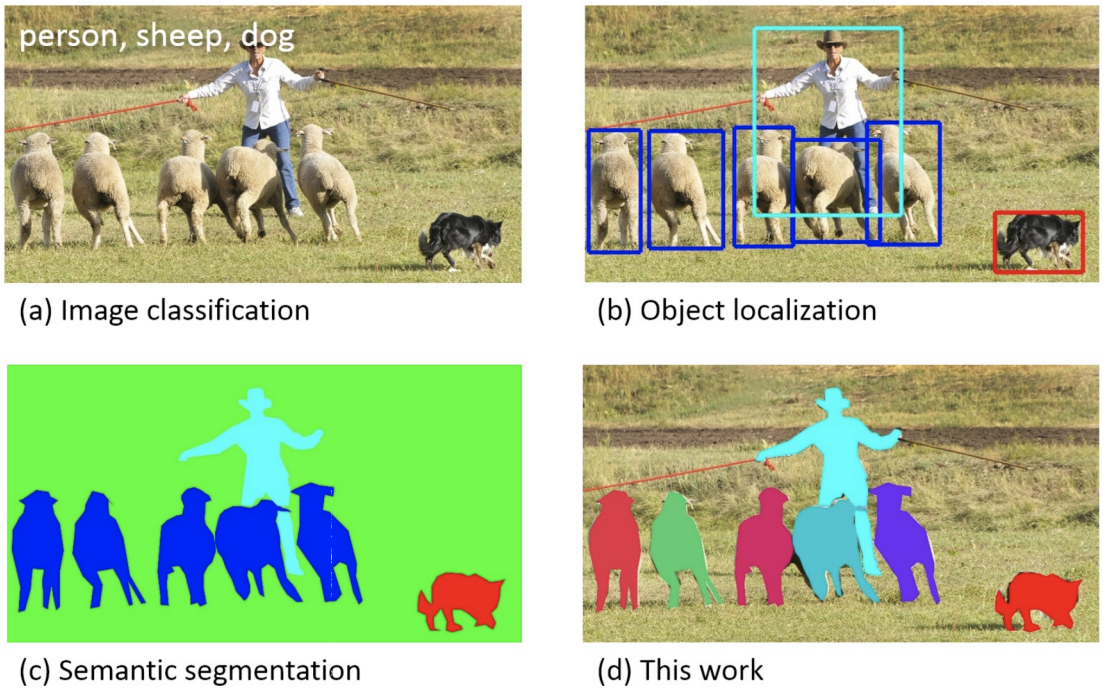

Figure 2.6: Image taken from the COCO paper, it shows what is the focus of the COCO dataset (d) i.e. segmenting individual object instances, compared to the other dataset goals(a)(b)(c).[31]

it is already implemented within Google Photos to recommend the best part of one or more videos such as when a person smiles or a child blows the candles out on its own birthday party.

Another interesting research area is **image styling**, which basically refers to transfer style from a image to another, as it is possible to see better in the figure 2.8.

One problem with style transferring is that the stylized photo may not be remain photorealistic. This is mainly due to the fact they tend to generate spatially inconsistent stylizations with noticeable artefacts. To overcome this problem Nvidia and the University of California, Merced propose a new solution named **FastPhotoStyle**, in the paper "A Closed-form Solution to Photorealistic Image Stylization" by Yijun Li et al.[29]. They propose a method which consists of two steps, stylization and smoothing. Stylization transfers the style of the reference image to content image and the smoothing step is responsible to ensure spatially consistent stylizations. This method has two main advantages compared to the previous state-of-the art, first thanks to the smoothing step it can generate more realistic images and then using a closed-form solution, in fact FastPhotoStyle can produce the stylized image 49 times faster than previous methods. The main achievement of this model is the introduction to a novel image stylization method that is able to outperform

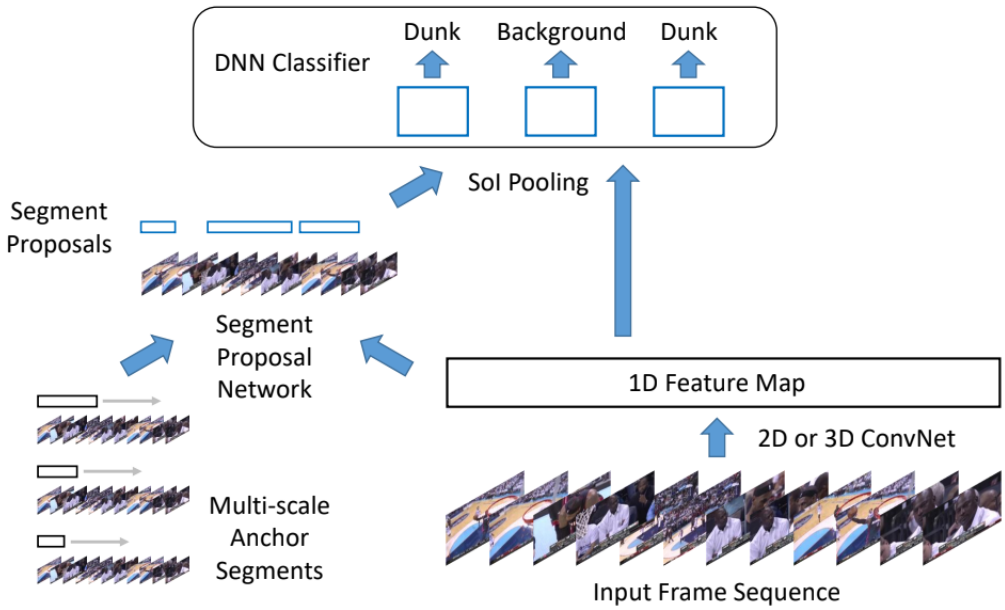

Figure 2.7: TalNet model. It gets a sequence of input frames from a video and it converts them into a sequence of 1D feature map. This map is passed to a segment proposal network that generates candidate segments. This input is fed to a Deep Neural Network which applies the representations learned for each candidate segments which will give a score to them.[8]

Figure 2.8: Different ways for image Image stylization. The first image in the row is the style, while the second one is the content in which we want to apply the style. The third image is from the model by Gatys et al.[13], the fourth is by Luan et al.[33] while the last one is from the FastPhotoStyle model where the image is taken from.[29]

the previous artistic stylization algorithms by rendering much fewer structural artefacts and inconsistent stylizations, and also by synthesizing both colours and patterns in the style images.

Many successful and fascinating studies over the last few years have been achieved in

the field of **image Generation** which, in a nutshell, is the generation of completely new images, starting from a training dataset, that should be similar to the ones in the starting dataset. For example given as input a set of images of human faces, the goal is to generate new faces, different from the input set, which could possibly exist in reality. There are many approaches to address this problem, the most used nowadays are **Generative Adversarial Networks**, (GANs) introduced by Ian Goodfellow et al. in 2014[16]. GANs are deep neural networks composed of two parts, on the one hand there is a Generative network, responsible to generate the "fake" images and on the other hand there is a discriminative network, in charge to detect the authenticity of the images using a probability. GANs often deal with the problem of capturing geometrical and structural patterns in an image. In order to decrease the impact of these issues, Han Zhang et al. introduced **Self-Attention Generative Adversarial Networks** (**SAGANs**) which have achieved the current state-of-the-art results in image synthesis[52]. Self-attention techniques enable the model to design relationships between separated spatial regions even if they are relatively distance among each other. This model calculates response at a specific location, as weighted sum of the features at all the locations. Moreover they used several techniques to improve the performance in their model. First they apply spectral normalization for both generator and discriminator, usually applied only on the discriminator in many GANs models, in this way they can stabilize the gradient preventing the escalation of parameter magnitudes. Furthermore this model utilizes separate learning rates for generator and discriminator in order to counterbalance the slow learning problem in a regularized discriminator and also enabling to use fewer generator steps per discriminator step. SAGAN achieved the state-of-the-art results, increasing the Inception Score from 36.8 to 52.52 and reducing Frechet Inception distance from 27.62 to 18.65 on the challenging ImageNet dataset[44].

There are two main sub-fields in the image synthesis(or generation) field, *conditional* i.e. generating images conditionally from the dataset, based on a label(their class) and *unconditional*, i.e. generating images unconditionally from the dataset. The state-of-the-art in the class-conditional image synthesis has been recently reached by BigGANs[7], achieving an Inception Score of 166.3 and Frechet Inception Distance of 9.6, improving over the previous best of 52.52 and 18.65 respectively. In their paper Andrew Brock et al. demonstrate that if the existing Generative Adversarial Networks are trained at the very large scale, which means increasing the number of parameters and the batch size, they can generate images that look very realistic.

Even here it is possible to move from images to videos to lead to another field of computer vision, **video synthesis**, Ting-Chun Wang et al. recently introduced a new model[49], video to video synthesis capable of synthesizing 2K resolution videos of street scenes up to 30 seconds long. This model is the current state-of-the-art. The method couples generator

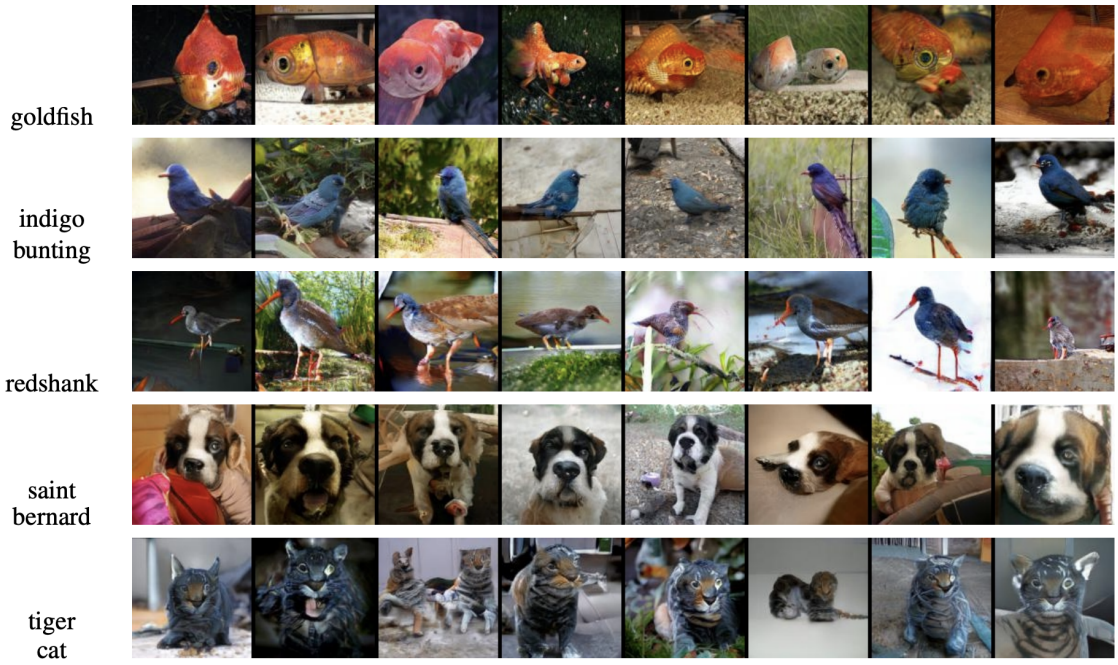

Figure 2.9: Some of the images created by SAGAN.[52]

Figure 2.10: Some images generated by BigGANs.[7]

and discriminator with a spatio-temporal adversarial objective. The main goal of this model is to learn a mapping function from an input source video, for instance a sequence of semantic segmentation masks, to an output photorealistic video that renders the content of the source video. The paper also states that video frames can be generated sequentially, and this generation depends on three factors, current source frame, past two source frames, past two generated frames. It describes also how using multiple discriminators can mitigate the mode collapse problem during GANs training. The paper has been criticised due to the fact it can be used to create deepfakes, and it always useful to keep in mind the ethical considerations about this kind of technologies.

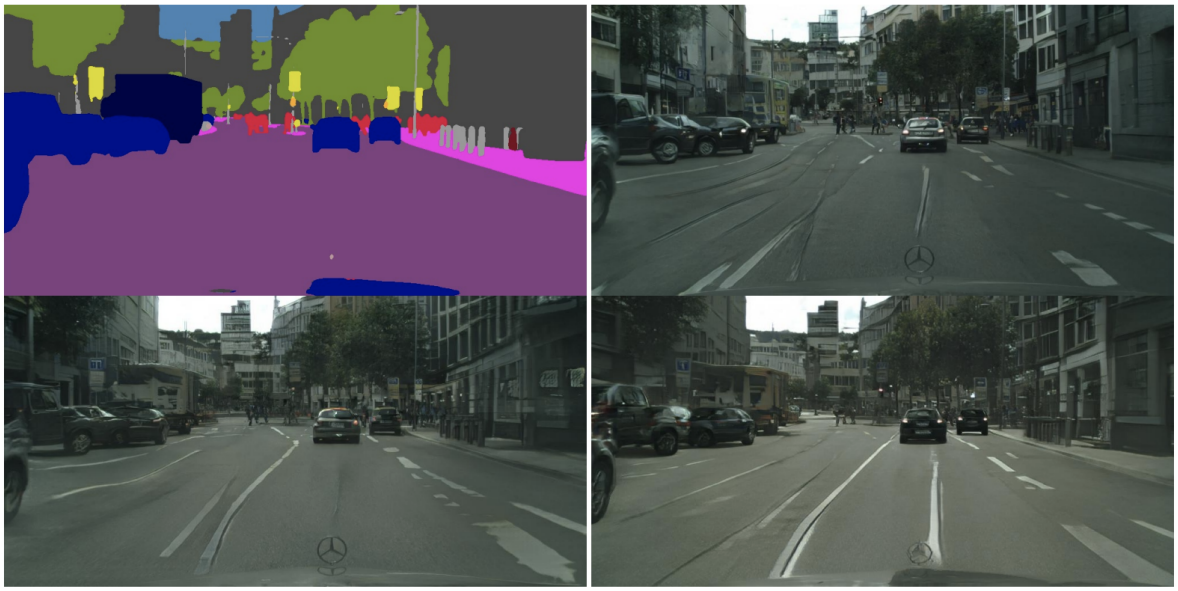

Figure 2.11: Some examples of video synthesis starting from an input segmentation map video(top-left). Pix2pixHD model[50](top-right), COVST model[9](bottom-left) finally the Ting-Chun et al. model[49](bottom-right), from which the image come from.

2.1.3 Optical Character Recognition

In this section it will be explained the current state-of-the-art in the field of Optical Character Recognition. There are many levels in which it is possible to apply OCR, from a scan of a single character, to an entire document, from hand-written texts or historical documents, to detect text in the so-called natural-images. In the last years many good results have been achieved regarding the scan of "normal documents"(i.e. computer written with common fonts like times new roman, etc..), and there are many tools and software out there which can do really good jobs. For this reason the latest researches and studies about Optical Character Recognition are mainly focused in two field analysis of historical documents and scene text recognition i.e. text recognition/detection in images which are not purely text and in which text could be in any shape and format.

2.1.3.1 Text detection and recognition

In common images, for example landscapes, roads, cities, any kind of image which is not a document, OCR task is usually divided in two sub-field, *text detection*, which is the problem to identify text in an image and draw boundaries around it, and *text recognition*, which is the problem to actually read the text from the images. In their first deep learning approaches, OCRs handled these two fields separately, for example by using two independent neural networks. The approaches are different now, for instance Christian Bartz et al. in their

work *A single Neural Network for Text Detection and Text Recognition*[6] have introduced indeed a single neural network to solve the two problems together. They introduced **STN-OCR**, a network that integrates a spatial transformer network, in order to detect text regions in an image, with a text recognition network which takes the identified text regions and recognizes their textual content. The Text Detection part is formed by a Localization Network, a Grid Generator and Image Sampling network which produce an output of N text regions. These N regions are processed independently by the text recognizer, everything training in a semi-supervised way. In 2017 this framework was able to reach state-of-the-art/competitive performance on a range of standard scene text detection and recognition benchmarks.

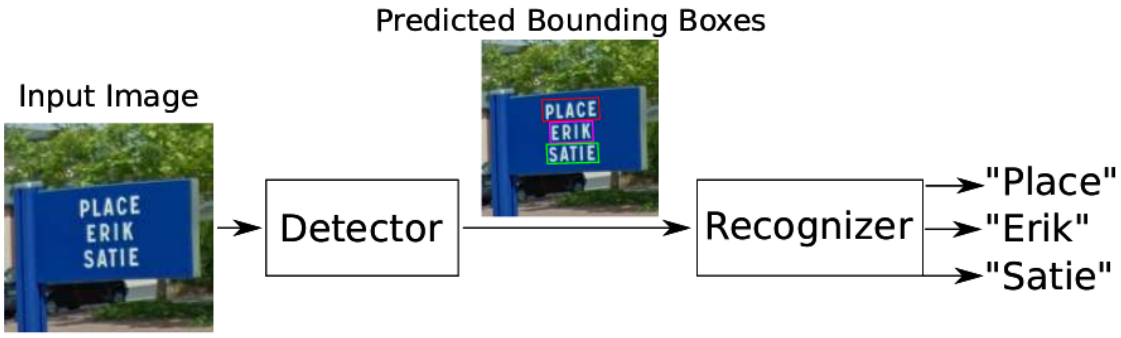

Figure 2.12: Example from Bartz et al., a single neaural network for Text Detection and Text Recognition, it is possible to see that the first neural network, the detector, is responsible to detect text and draw boundaries around it while the second one, the recognizer, actually read the text[6]

Next the focus was aimed to the problem of curved-text recognition and more generally to text with Arbitrary Shapes. The current state-of-the-art has been achieved by Pengyuan Lyu et al. with their model named Mask **TextSpotter**[34], inspired by the work Mask R-CNN[21]. Similar to the previous models, it uses an end-to-end learning procedure but with the main difference that text detection and recognition are acquired via semantic segmentation. This helps the model on finding text in many scenarios as we can see in figure 2.13.

Moreover, it performs better than previous methods in handling text instances of irregular shapes, for example, curved text. Experiments on many databases have been done and this method has achieved the current state-of-the-art results in both scene text detection and end-to-end text recognition tasks.

Figure 2.13: Mask TextSpotter application on different images[34]

2.2 Text analytic platforms

In this section it will be presented the latest technologies and products, available in the market nowadays, considering also the open-source scenario.

2.2.1 On-market View

Text analysis platforms, (or text analytic or text mining) use **Natural Language Processing** and Machine Learning techniques to extract particular information from text such as insights, meanings, topics, key phrases or to perform sentiment analysis, infers language and much else. Moreover text analytic platforms should provide tools for visual representations and interpretation of the data. These platforms may provide tools to consume data from different sources, such as social media, e-commerce reviews, emails, web-sites and any kind of other documents. All of these characteristics will define the quality of a platform.

Following some of the main text analysis features:

- *Language Identification*: Many text analytic platforms provide tools for automatic language detection. This is particularly useful for many reasons, first of all when data comes from multi-modal data sources or social media in order to automatic label and categorize data such as Tweets or Posts, moreover to improve performance of the next steps such as "Part of Speech tagging", etc.

- *Sentiment Analysis*: i.e. try to identify if a particular word, sentence or in general text data, can be labelled with some kind of sentiment or emotional state, usually "Positive"("Happy"), "Negative"("Sad") and "Neutral". It is also possible to enlarge this set of sentiments with other ones, such as "Angry", "Anxiety", "Passion" and so on.

- *Part of Speech Tagging*: Text analysis platforms offer tools to tag words with a part of speech, they can label each word as nouns, pronouns, verbs, adjectives, etc.

- *Syntax Parsing*: The process of analysing text data, conforming to the rules of form grammar. Differs from "part of speech tagging" because instead of analysing the specific word it takes into consideration the whole sentence trying to figure out the inner structure, resulting in a parse tree showing syntactic relations, semantic and other information.

- *Entity Recognition*: or also "Named Entity recognition (NER)" is the process which tries to identify some pre-defined categories into a text, such as "Person", "Location", "Organisations", etc. For instance identify "Mario" as "Person" or "Facebook" as "Organisation".

- *Key-phrase Extraction*: Similar to the previous one, but it tries to identify keywords or key-sentences, which capture the primary topics discussed in the text.
- *Co-reference Resolution*: it tries to find all expressions that refer to the same entity in a text.

In addition some other features are usually integrated into a Text Analytic platform:

- *Metadata management*: which manages the data that provides information on the content of the data itself.
- *Data management*: which includes capabilities of data harmonization and data cleansing to help ensure the quality and integrity of the data.
- *Data lakes*: which provides a large, scalable repository to store data. It's generally based on a high-capacity database.

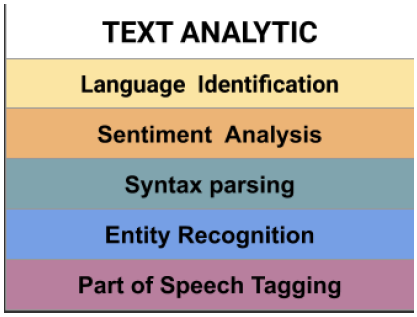

Figure 2.14: Some of the important features which characterize a text analytic platform

Following some of the principal platforms available on the market nowadays:

- **Microsoft Azure**. This platform is developed by Microsoft and present the following features among others. In the first place it includes sentiment analysis, APIs which also support a great number of languages. They use score with a range from 0 to 1; scores close to 1 indicate positive sentiment, and scores close to 0 indicate negative sentiment. Sentiment score is generated using classification techniques. The input features of the classifier include n-grams, features generated from part-of-speech tags, and word embeddings. Moreover, the **key phrase extraction** feature is included, i.e. an API which returns a list of strings which will be the key talking points in the input text. Several languages are supported, English, German, Spanish, and also Japanese. Furthermore, this service includes a *language detector* which is similar to the others the APIs returns the detected language with a score from 0 to 1. Scores close to 1 indicate that the identified language is true. This feature supports more

than 120 Languages. Nonetheless Microsoft Azure includes a NER, APIs to identify all the named entities in the text, such as people, and locations, organisations, and so on. This feature it also able to contextualise, for instance, determine whether a term such as "Amazon" refers to the website or the actual rainforest.[35]

- **Amazon Comprehend**: The service identifies the language of the text, extracts key phrases, (also here places, people, brands, etc.), understands how positive or negative the text is, analyzes text using tokenization and parts of speech, and automatically organizes a collection of text files by topic. It is also possible to use AutoML capabilities in Amazon Comprehend to build a custom set of entities or text classification models that are tailored uniquely to your organization's needs, without being an expert on Machine Learning. The interesting feature in this platform is the possibility of extracting complex medical information from unstructured text, you can use with the so-called Comprehend Medical service. This service is able to identify medical information, such as medical conditions, dosages and frequencies of medicines, from a variety of sources like doctor's notes, clinical trial reports, and patient health records. This service also identifies the relationship between the medication and information on tests, treatment, and procedures. For instance, the service identifies certain doses, levels, and frequencies related to a particular drug from unstructured clinical notes.[3]

- **NVivo**: This platform offers several versions depending on different variables like operative system, amount of features (basic, pro, plus), company versions and so on. The NVivo 12 Plus for Windows offers interesting functionalities such as automate coding, which helps the programmer to code using pattern-based auto coding to automatically categorize and classify data. Furthermore, offers all the tools to analyze social networks(Facebook, Twitter, Linkedin), such as sentiment analysis, NER, discover influencers and opinion leaders. Finally it provides an automatic way to create sociogram visualizations to see network relationships and interactions and use metrics to discover critical network roles like influencers, connectors and brokers.[32]

- **Rapidminer**: This platform includes several services, not really correlated with this topic, but it also offers a Text mining tools with many functionalities such as sentiment analysis and analyze direct feedback from users. A fascinating service relies to improve fraud detection which identifies patterns in written text that indicate fraud may be at play. Dig below the surface of transactional data for signs that could be overlooked.[22]

- **Google Cloud Natural Language API**: Similar to the previous platforms, Google Cluoud Natural Language API provides features for sentiment analysis. An interesting feature this platform provides is that, as well as the entity recognition, it also includes

a function to return information and the sentiment about the entities it recognises. Furthermore there is a syntactic analyzer which extracts linguistic information, breaking up the text into tokens in different ways, given the possibility to perform further analysis on those tokens. Finally on this API is available a content classification which is able to analyze text content and return a content categories from the content itself. Moreover each API can also detect and return the language, if not specified by the user in the initial request.[18]

- **OpenNLP**: The Apache OpenNLP library is not really a platform but a machine learning-based toolkit for the processing of natural language text. Even though it supports the most common NLP tasks, such as language detection, tokenization, sentence segmentation, part-of-speech tagging, named entity extraction chunking, parsing and coreference resolution[12].

2.2.2 OCR Platforms

In this subsection it is going to be introduced the actual OCR-platform scenario of software and API products available on the market and on the open source side. Here some of the most relevant software and API projects:

- **Amazon TextExtract**: this is a service which extracts text and data from scanned documents. It able to identify also the contents of fields in forms and information stored in tables. It uses machine learning techniques to extract text from several types of documents and it can detect layout and understand the data relationships in any embedded forms or tables, and extracts everything with its context intact.[5]

- **Amazon Rekognition**: First of all it is important to notice that this is much more than a simple Optical Character Recognition in fact this set of API is able to do OCR as well as identify text, people, scenes, activities, face expressions and also inappropriate content. This service provides facial analysis and facial recognition on images and video offering services for detecting, analyzing, and comparing faces.[4]

- **Google vision api**: Same as the previous one, this is more than a simple OCR, it can also detect emotion, understand text, and more. It includes two main modules, AutoML Vision which uses users' custom nlp models and vision API which on the other hand is a set of pre-trained machine learning models through REST and RPC APIs.[19]

- **OCR Space**: is a more standard OCR service converts scans or images of text documents into editable files claiming to use the state-of-the-art OCR software. It provides a simple free online version and a pro version for more advanced tasks.[1]

- **Tesseract OCR**: One of the most accurate and probably the most famous in the open-source scenario, Tesseract is an Open-Source Optical Character Recognition engine for various operating systems, developed and sponsored by Google since 2006. The latest version(4), is made by a neural net (LSTM) based OCR engine which is focused on line recognition; it can recognize more than 100 languages and many fonts by default, and it also can be trained in different ways in order to recognize different fonts and styles.[37]

CHAPTER 3

Technologies

This chapter offers a brief review of the main technologies that have made possible this project.

3.1 Programming language and libraries

This section introduces a brief review of the programming language used, Python, and some of the main libraries and frameworks utilized in this project.

3.1.1 Python

The OCR implementation in this work is completely implemented in **Python v3.6**. Python is probably the most popular language for machine learning tasks and most likely one of the most used languages in general. There are many reasons python is so attractive and it became de-facto standard for ML tasks, first of all is very simple to understand and to use, by philosophy of the Python language itself, in fact it was built for readability and less complexity. The second, but not less important reasons, is the large amount of libraries dedicated to machine learning this programming language supports, numpy[36], opencv[38], scikit[39], nltk[40], librosa[30], pandas[51], tensorflow[2], pytorch[41], Keras[11] and so on. Everything a programmer needs to work with images, text, audio, maths, scientific computing, machine learning, data, deep learning, web applications and so on.

Figure 3.1: Python logo

3.1.2 Keras

Keras[11] is the library used in this project to implement the model. It is a high-level neural networks APIs written in Python and supports multiple back-end neural network computation engines. Neural layers, optimizers, cost functions, activation functions, initialization schemes, regularization schemes and so on, are all standalone modules that can be combined to create new models. New modules can be added simply as new classes and functions. Models are defined in Python code, not as separate model configuration files as

many other APIs do.

Its guiding principles start from the concept of being user friendly. Keras offers ease of learning and ease of model building and a wide range of production deployment options, nonetheless it offers integration with several back-end engines, and provides large support for distributed training.

With Keras it is possible to define layers in just one line of code. Core layers are:

- **Dense**: i.e. dot product plus bias.
- **Convolutional**: to filter and create feature maps.
- **Pooling**: for down-scaling, includes most of the types such as max and average.
- **Recurrent layers**: including LSTM, GRU and many others.[10]
- **Dropout**, useful to avoid overfitting.[47]

and many others.

As also the documentation states, there are two main types of models available in Keras the *Sequential* model, and the *Model* class used with the functional API. These two classes are used to build the model in a high-level (and indeed) programmer-friendly approach. The models can be built in two ways, *Functional* and *Sequential*. The latter is simple but limited in model topology while the functional is useful for creating complex models, such as multi-input/multi-output models, directed acyclic graphs and models with shared layers.[11]

In this project we use Keras on top of Tensorflow.

Figure 3.2: Keras logo

3.1.3 Tensorflow

TensorFlow[2] is an end-to-end open source platform for machine learning. It is a symbolic math library, and is primarily utilized for machine learning applications. It is used for both research and production at Google.

In this project TensorFlow it use exclusively as back-end for Keras but it can be used standalone, in fact it offers several interesting functionalities. It provides a large flexibility in its operability, meaning it has modularity in which each part could has been used standalone.

Moreover it is easily trainable, on CPU as well as on GPU for distributed computing and includes pipelining for parallel neural network training i.e. the possibility to train multiple neural networks on multiple GPUs which makes the models very efficient on large-scale systems.

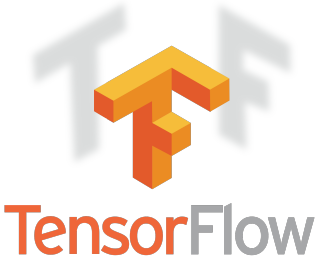

Figure 3.3: TensorFlow logo

3.1.4 Numpy

Numpy[36] is one the most powerful and probably the fundamental package for scientific computing with Python. The main interesting functionalities are the N-dimensional array objects, sophisticated broadcasting functions, linear algebra, Fourier transform and random number capabilities. Beyond, NumPy can also be utilized as multi-dimensional container of generic data. Arbitrary data-types can be defined.

Figure 3.4: Numpy logo

3.1.5 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.[38] This library provides a universal infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. OpenCV supports more than 2000 algorithms, including both classic and state-of-the-art computer vision and machine learning algorithms.

3.1.6 Other libraries

In the previous sections the principal APIs, libraries and tools have been discussed, but many other ones have been used, whereas, each of them, performed a relative small amount

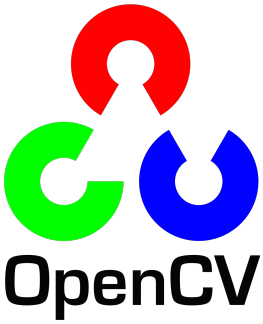

Figure 3.5: OpenCV logo

of operations, but not for this, less important. First **Cairocffi** is a set of Python bindings and object-oriented API for cairo. **Cairo** is a 2D vector graphics library with support for multiple backends such as PNG, PostScript, PDF, SVG, image buffers and so on. Another useful library is **Matplotlib**, which is a Python 2D plotting library which produces publication quality figures in a variety of formats and among many environment. Finally **SciPy** which is a Python-based ecosystem of open-source software for mathematics, science and engineering. In particular, it includes many of the cited above libraries such as numpy, matplotlib and many others.

3.2 Tools

This section offers a brief description of the two main tools used to build, implement and run the project.

3.2.1 Jupyter Notebook

Jupyter Notebook is part of the Jupyter Project[24], and it is an open-source web application in which a programmer can create and share documents that contain live code, equations, visualizations and narrative text. It includes many uses such as data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and so on.

Figure 3.6: Jupyter logo from Jupyter website[24]

3.2.2 Google Colaboratory

Google Colab[17] is a free cloud service in which programmers can develop and deploy deep learning applications. The most important feature that distinguishes Colab from other free cloud services is that it provides GPU. Colaboratory is a Jupyter notebook environment that requires no setup and runs entirely in the cloud. Due to the lack of powerful hardware much of the training has been done with this tool.

Figure 3.7: Google colaboratory logo

CHAPTER 4

Architecture

This chapter describes the overall architecture of the model presented in this project. Each components is described with the connections among other elements involved on the development of the model.

4.1 Overall View

The core of this project relies in the Neural Network, before going through the explanation of the model, is necessary to mention where the inspiration of the latter derived from. First of all the Convolution Recurrent Neural Network (**CRNN**) model explained in the paper "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition" by Baoguang Shi et al.[45] was the primary source from which the process thinking of the model started. The model presented in this project is indeed based on CRNN. Furthermore the architecture provides in this project refers to the one presented by Keras's developers (`https://keras.io/examples/image_ocr/`). Finally part of code and the datasets organization is inspired by the article *latest-deep-learning-ocr-with-keras-and-supervisely-in-15-minutes*[48].

The model presented in this work consists of three main components, a **Convolutional Neural Network** (CNN), in which the input is fed, responsable to extract features from the images. Its output is serve as input to a **Recurrent Neural Network** (RNN) composed by GRU layers. RNN gives in output a set of vectors containing probability distribution of observing alphabet symbols at each time step. Finally there is a **Connectionist Temporal Classification** (CTC) layer which serves two purposes, first calculate the loss value to train the model and decode the matrix to get the text contained in the input image. The model is explained in detail in the table 4.1 and in the image 4.1.

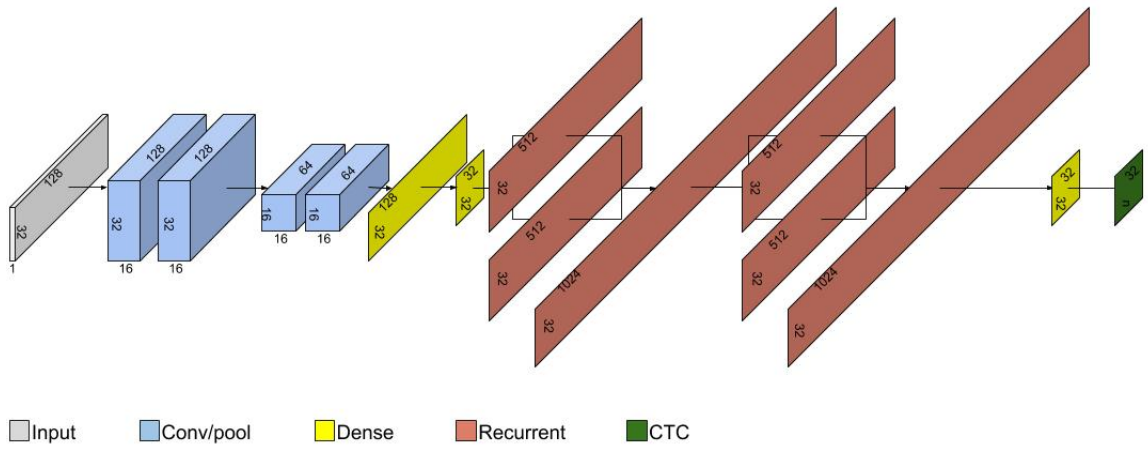

Figure 4.1: Schema of the model layer by layer. more details in the table 4.1

Layer	Type	Out-Shape	Parameters	Connected to
input	InputLayer	(128, 32, 1)	0	-
conv1a	Conv2D	(128, 32, 16)	160	input
conv1b	Conv2D	(128, 32, 16)	2320	conv1a
pool1	MaxPooling2D	(64, 16, 16)	0	conv1b
conv2a	Conv2D	(64, 16, 16)	2320	pool1
conv2b	Conv2D	(64, 16, 16)	2320	conv2a
pool2	MaxPooling2D	(32, 8, 16)	0	conv2b
reshape	Reshape	(32, 128)	0	pool2
dense1	Dense	(32, 32)	4128	reshape
gru1a	GRU	(32, 512)	837120	dense1
gru1b	GRU	(32, 512)	837120	dense1
concat-1	Concatenate	(32, 1024)	0	gru1a,gru1b
gru2a	GRU	(32, 512)	837120	concat-1
gru2b	GRU	(32, 512)	837120	concat-1
concat-2	Concatenate	(32, 1024)	0	gru2a,gru2b
dense2	Dense	(32, 28)	28700	concat-2
softmax	Activation	(32, 28)	0	gru2a-gru2b
CTCloss	Lambda	-	0	-

Table 4.1: Layers description of the model presented in this project. Keras requires the CTC layer to be implemented as Lambda layer. The total number of trainable parameters is 6,435,852

4.2 Achitecture

In this section each component of the model will be explained in detail. In the figure 4.7 it is possible to have an intuition about each major components' task.

4.2.1 Input pre-processing

Before going fed into the model, it is necessary to make some pre-processing on the input image. With OpenCV it is possible to read images from the file system and automatically converts it into a Numpy array. Then, in the second step, images are resized into predefined width and height and turn into a grey-scale. The grey-scale transformation makes possible to move from three channels, Red-Green-Blue, to one, grey-scale. Therefore the input of this model is a Numpy array of dimensions [image-height, image-width, 1]. Moreover the labels need to be transformed as integer vector, starting from the corresponding alphabet, for instance the word "artificial" will have as corresponding label something similar to [1, 19, 23, 4, 2, 4, 16, 4, 1, 11].

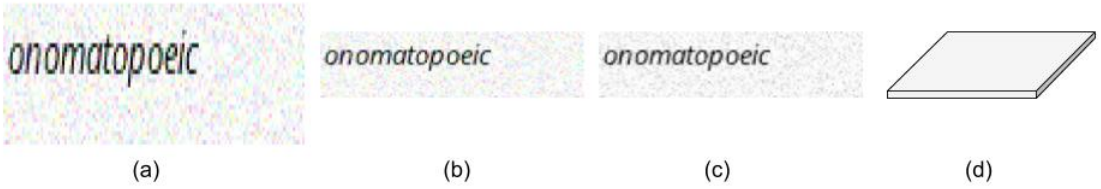

Figure 4.2: Image pre-processing steps, first there is the input image(a), then the image is resized (b)and transformed into a grey-scale(c), finally the shape represents the Numpy array(d) used to feed the model.

4.2.2 Feature Extraction with Convolutional Neural Network

Before going into the detailed explanation of this part, is useful to recap briefly how a **Convolutional Neural Network** works. Normally input images are defined by height, width and number of channels(usually three, RGB but only one in the model proposed in this work, as explained in the previous section). The Convolutional Layer makes use of a set of learnable filters which are in charge to detect the presence of specific features (or patterns) in the input image. Different problems have different amount of filters, and usually each filter has small dimension. To detach features, each filter is convolved (slid) across the input image, and a dot product is computed to give an activation map, giving, in the end, a set of activation maps. There is a simple formula to compute the dimensions of the activation map which is the following:

$$(N + 2P - F)/S + 1 \tag{4.1}$$

where N is the dimension of image (input) file P is the amount of zero-padding, F denotes the dimension of a filter and finally S is the stride which defines how many pixels to slide. For different reasons, sometimes is useful to pad the image with zeros on its border, the size of this pad is defined as zero-padding.

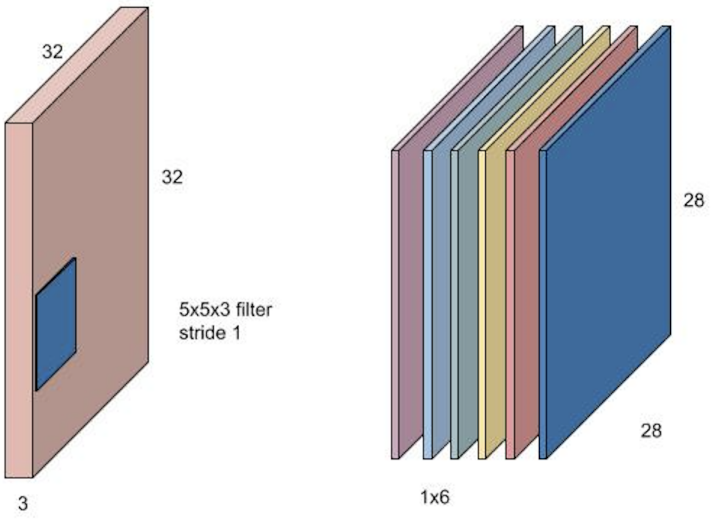

Figure 4.3: A convulutional step over an image 32x32 pixels plus 3 channels (RGB), with 6 filters 5x5x3, stride equals to 1, and no zero-padding produces an activation map 28x28x6.

On the other hand, the Pooling layer basically reduces the amount of parameters and computation in the network, by progressively reducing the spatial size of the network, allowing control of overfitting.

Moreover convolutional layers and pooling layers as the interesting property to be translation-invariant i.e. sliding the object(the word in this case) over within the image space, will have the same output but simply translated over. This is particularly useful because words have different sizes and shapes and may be located in different spatial positions.

In the table 4.1 and in the figure 4.1 it is possible to see the structure of the model and see the Convolutional Network component represented by a pair of convolutional layers followed by a Max-pooling layer, this repeated twice. The work of such components is related to extract feature representations from ther input image. These layers are taken from a standard CNN, but in order to properly feed the Recurrent Network of this model, feature extraction needs to be sequential. Specifically, this convolutional network generates

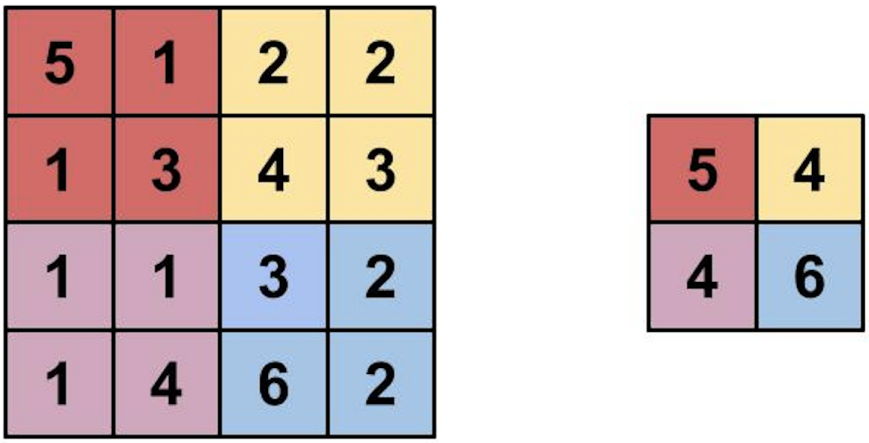

Figure 4.4: A max pooling step, from each region, identified by different colors, is extract the maximum value. In this case there is a 2x2 filter and the slide(stride) is 2. Notice that the resulting matrix changes according to the parameters, if for instance, the stride was 1, the corresponding matrix would be 3x3.

a **vector feature sequence**, in which each sequence vector is generated from left to right, and this sequence is then served sequentially to the recurrent network. Practically each vector of the feature sequence corresponds to a region of the input image, whereby vectors and regions are in the same order from left to right. Therefore each vector from the feature sequence is an image descriptor for a specific region of the input image.

Finally on each layer has ReLU as activation function. Which is defined as follows:

$$f(x) = x^+ = \max(0, x) \tag{4.2}$$

ReLU has the great advantage over other activation functions i.e. it does not activate all neurons at the same time, converting all negative inputs to zero not making activating those neurons. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. It is proved that ReLU converges six times faster than tanh and sigmoid activation functions.

4.2.3 Per-frame Predictions with Recurrent Neural Network

The output of the convolutional network is fed into the recurrent layers. Even in this section is necessary to describe briefly how a recurrent layer, and in particularly the GRU (Gated Recurrent Unit), utilized in this model work.

In a high level, a **recurrent neural network** (RNN) processes sequences one element at a time while retaining a memory (state) of what has come previously in the sequence, this element could be integers, video frames, characters in a word, sentences, or sensor

measurements. In other words this implies the output at the current time step becomes one of the inputs in the next time step. This is true for each unit which composed the recurrent layer as it is possible to see in the figure 4.5. For each element of the sequence, the model considers not just the current input but also all the previous.

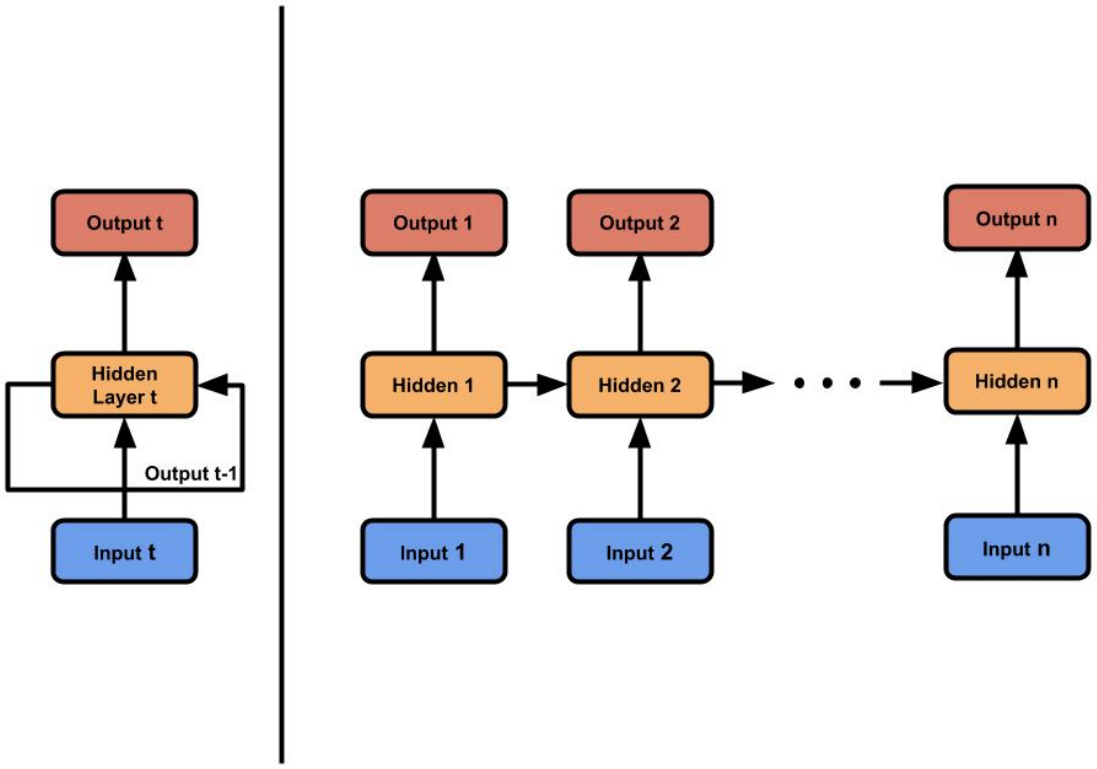

Figure 4.5: On the left a sample of an RNN structure while on the right its unfolded representation.

This particular structure grants the recurrent part to obtain memory allowing the learning of long-term dependencies in a sequence. This means it can take the whole context into consideration when it makes predictions. Each unit of the recurrent layer is a GRU unit. These GRU units are meant to solve the problem of the Vanishing Gradient. In a nutshell this means that when there are layers using certain activation functions(tanh, sigmoid, etc), the gradient of the loss function approaches zero, making the network hard to train. In the next subsection the GRU unit will be briefly introduced, but before that it will be explained the model presented in this project.

In the model introduced in this project there are two bidirectional GRU layers, and the output of those is concatenated and served as input to another pair of bidirectional GRU

layers, in which again, the out is concatenated and serve as input to the decode algorithm, i.e. CTC. The GRU layers are bidirectional. Directional means conceptually it only uses past contexts. However in this case and in general when the problem deals with image-based sequences, contexts from both directions are useful and complementary to each other.

Furthermore, output of the recurrent layers is reshaped into a sequence of n vectors (with $n = 32$ in testing), and on each vector normalized into $(0, 1)$ interval using softmax function. The softmax function is defined as follows on a general vector $\mathbf{z}$:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad i = (1, \ldots, K), z = (z_1, \ldots, z_K) \in \mathbb{R}^K \tag{4.3}$$

Where, standard exponential function is applied to each element z_i of the input vector $\mathbf{z}$ and normalize these values by dividing by the sum of all these exponentials. This normalization ensures that the sum of the components of the vector $\mathbf{z}$ is 1. In this specific case each vector contains probability distribution of observing alphabet symbols at each time step(rnn output).

In this case the output is a vector sequence $y = y_1, ..., y_n$ where T, is the sequence length(and also the number of time steps). Here, each $y_t \in \Re^{|A|}$ is a probability distribution over the set A, where A contains all the encoded alphabet characters(labels characters), as well as a 'blank' label denoted by "-". Notice that the probability distribution does not keep into consideration where each character in the label L is located, which will mean the decode algorithm will not know where each character occurs.

4.2.3.1 Gated Recurrent Unit

Introduced for the first time by Cho et al. in their paper "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", GRU (**Gated Recurrent Unit**[10]) aims to solve the vanishing gradient problem. In order to solve that problem GRU unit introduces two types of gates, **update gate** and **reset gate** which decide what information should be passed or not to the output. Basically, these are two trainable vectors.

The internal structure of a GRU unit is visible in figure 4.6. Following that schema, to compute H_t, it is necessary to start from computing the two gates. Beginning with the update gate Z_t for the time step t with the formula:

$$Z_t = \sigma(W_z I_t + U_z H_{t-1}) \tag{4.4}$$

Where I_t, H_{t-1} are the input of the unit and the information of the previous unit respectively, and W_z, U_z are their weights. Finally the sigmoid activation function (σ) responsible to squash the result between 0 and 1. Abstracting, Z_t is the amount of information from

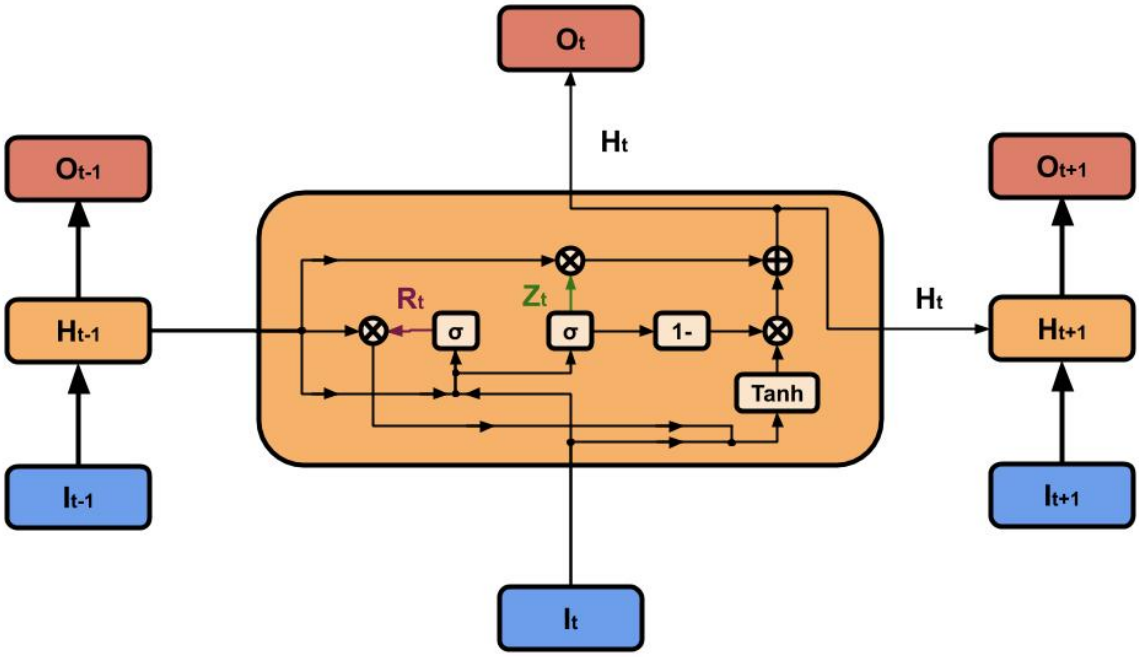

Figure 4.6: Representation of a GRU unit. Zt indicates update gate, Rt is the reset gate. The σ-boxes represents the sigmoid function, while tanh is the homonymous function. The $\bigoplus$ indicates the plus operation and the $\bigotimes$ indicates the Hadamard product.

the previous time steps to pass to the next time steps. On the other hand reset gate R_t is in charge to decide the amount of information from the previous time steps, the model should discard. Following the formula:

$$R_t = \sigma(W_r I_t + U_r H_{t-1}) \tag{4.5}$$

Where W_r, U_r are different weights and σ is the always th sigmoid activation function. Now it is possible to compute the final output H_t as following:

$$H_t = Z_t \otimes H_{t-1} + (1 - Z_t) \otimes H'_t \tag{4.6}$$

Where $\otimes$ indicates the Hadamard product and H'_t is defined as following:

$$H'_t = \tanh(W_h I_t + R_t \otimes U_h H_{t-1}) \tag{4.7}$$

Where W_h, U_h are again different weights and tanh is the Hyperbolic tangent.

4.2.4 Predictions and Loss Calculation with Connectionist Temporal Classification

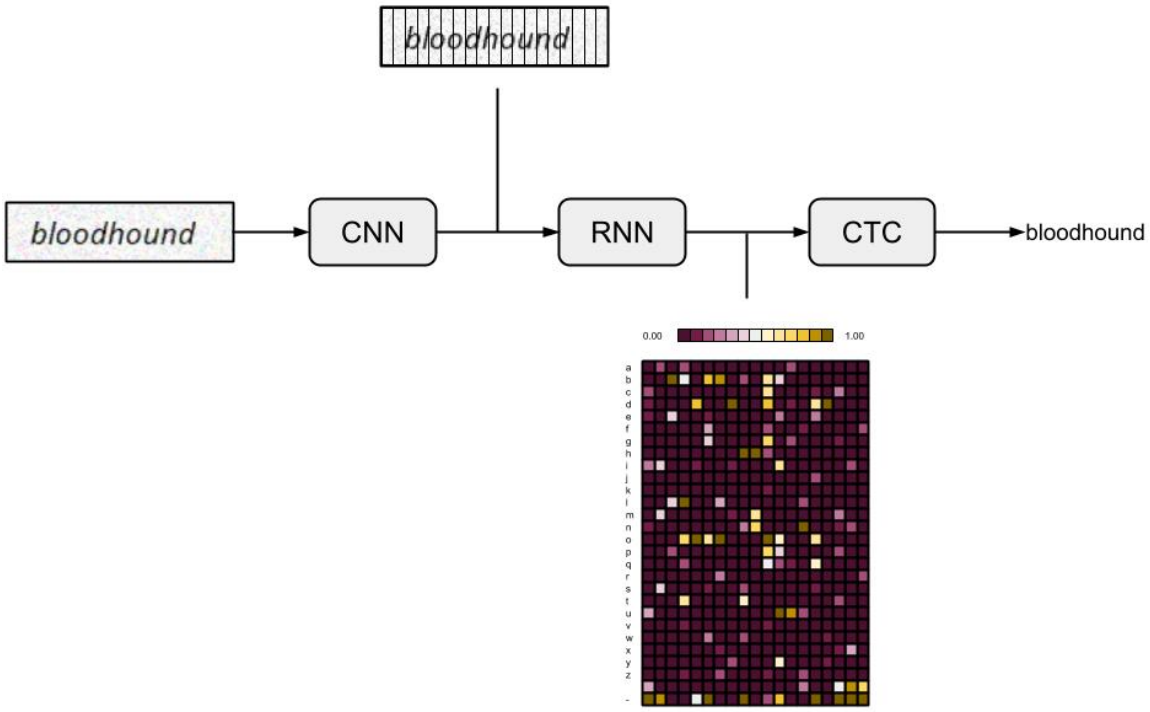

Figure 4.7: Schema of the model's major components: The input image is fed into the CNN, which extracts features and serves the RNN dividing the space into a vector sequence. Then the RNN gets this sequence and computes the probability distribution of the label characters for each vector, resulting in a probability matrix. This probability matrix is served to the CTC which calculates the loss and predicts the word.

The final component of this model's architecture adopts the conditional probability as defined in the **Connectionist Temporal Classification** (CTC) layer proposed by Graves et al.[20]

The task of CTC loss is, first to decode the vector sequence to infer the text contained in the input image, secondly calculate the loss value to train the model.

Recalling that CTC layer has in input the probability distribution over the vector sequence $y = y_1, ..., y_T$ where T is the sequence length (and so, the number of time steps from the recurrent network). In order to decode the sequence and compute the lost, CTC needs also the label L, which holds the encoded characters(labels) of the word.

Therefore, each $y_t \in \Re^{|A|}$ is a probability distribution over the set A, where A contains all the encoded alphabets (labels) characters, as well as a *blank* label denoted by "-". The output of the recurrent network is fixed as T, this means the space is divided into T region, but words can vary in length. This leads into some problems, first a single character can appears in more than one region and the model has to be able to distinguish from one character(e.g. the word "to", with one "o") to more occurrences of the same character(e.g. the word "too", with two Os); Even more there could be region without any character. This is the reason of the '*blank*' label. For example, if from the probability distribution y the decoded character sequence results in [-,-,t,t,t,-0,-,0] the decoding algorithm first merges all the successive occurrences of the same character not separated by a '*blank*' and then removed all the occurrences of the '*blank*' label itself, resulting in the word [t,o,o]. This sequence-to-sequence mapping is defining as a function B on sequence $\pi \in A^T$, so that B maps onto the label L.

Furthermore, the conditional probability is defined as the sum of probabilities of each π mapped by B onto L:

$$p(L|y) = \sum_{\pi:B(\pi)=L} p(\pi|y) \tag{4.8}$$

Where the probability of π given y is:

$$p(\pi|y) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{4.9}$$

with $y_{\pi_t}^t$ is the probability of having the specific character π_t at time step t. At this point the loss is simply the negative logarithm of the conditional probability as shown in eq 4.11.

Moreover, regarding the **predictions**, this model uses the so-called **best path decoding**, this means the possible label L' with the highest probability as defined in Eq. 4.8 is taken as the prediction using the following formula:

$$L' = B(\max_{\pi} p(\pi|y)) \tag{4.10}$$

that is the most probable label π_t at each time step t mapping then the result to L', by B.

An **example** of how CTC works it will be introduced with the figure 4.8. For this simple example only two characters are used so the alphabet is defined as $A = \{a, b, -\}$. The matrix y, with three time steps(columns, so $T = 3$), contains the distribution probability. Let's define the label as $L = ab$, first is necessary to calculate the loss for L. In order to do that it is necessary to find all the possible sequence paths π for that label, i.e. *ab*-, -*ab*,

a-b, *aab* and *abb*. Now using eq 4.9 it is possible to compute $p(\pi|y)$ for each path. For example $p(-ab|y) = 0.2 * 0.2 * 0.9 = 0,036$, (note that the numbers do not fully respect the color scale). Now following eq 4.8, it is necessary to simply sum up all this $p(\pi|y)$. The final step is to calculate the loss applying the negative logarithm on the probability and back-propagate it through the network. Instead, regarding the prediction, following 4.11, by taking the most likely character per time-step (following the yellows), the best path is *a-b* and finally, applying B it results into *ab*

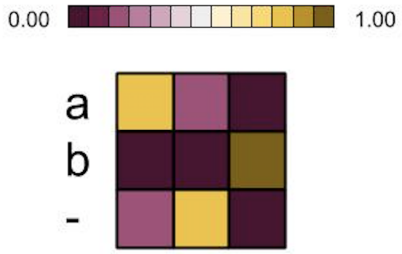

Figure 4.8: Distribution probability: alphabet $A = \{a, b, -\}$ and three time steps $T = 3$

In figure 4.7 it is possible to see a schematic representation of the major model's components.

4.2.5 Training and Optimization

In the previous section it has been declared that the final step is to calculate the loss applying the negative logarithm on the probability and back-propagate it through the network. The objective of the training step itself is indeed to minimize that value defined as:

$$\phi = -\sum_{I_i, L_i \in Tr} \log p(L_i|y) \tag{4.11}$$

Where Tr is the training dataset defined as $Tr = \{I_i, L_i\}_i$ in which I_i, L_i defines images and corresponding labels respectively. Upon the seen before ReLU and softmax functions there are a few more settings to take into consideration. In this model **Stochastic gradient descent** is used, with some optimizations. First of all the learning rate is set to 0.02 with decay rate of *1e-6*. Nesterov momentum equals to 0.9 is applied and finally all parameter gradients are clipped to a maximum norm of 5, which seems to increase the speed convergence.

4.3 Keras implementation

Keras is meant to be simple, in the following piece of code is implemented the definition of the model:

```
1     number_of_conv_filters = 16
2     kernel_size = (3, 3)
3     pool_size = int(downsample_factor ** (1/2))
4     time_dense_size = 32
5     rnn_size =  512
6     input_shape = (image_width, image_height, 1)
7
8     input_layer = Input(name='input', shape=input_shape, dtype='float32')
9     x = input_layer
10
11    for i in range(2):
12        x = Conv2D(number_of_conv_filters, kernel_size, padding='same',
      kernel_initializer='he_normal', activation='relu', name='conv'+str(i
      +1)+'a')(x)
13        x = Conv2D(number_of_conv_filters, kernel_size, padding='same',
      kernel_initializer='he_normal', activation='relu', name='conv'+str(i
      +1)+'b')(x)
14        x = MaxPooling2D(pool_size=pool_size, name='pool'+str(i+1))(x)
15
16    conv_to_rnn_dims = (image_width// (pool_size ** 2), (image_height //
      (pool_size ** 2)) * number_of_conv_filters)
17    x = Reshape(target_shape=conv_to_rnn_dims, name='reshape')(x)
18    x = Dense(time_dense_size, activation='relu', name='dense1')(x)
19
20    for i in range(2):
21        grua = GRU(rnn_size, return_sequences=True, init='he_normal',
      name='gru'+str(i+1)+'a')(x)
22        grub = GRU(rnn_size, return_sequences=True, go_backwards=True,
      init='he_normal', name='gru'+str(i+1)+'b')(x)
23        x = concatenate([grua, grub])
24
25    x = Dense(train.get_output_size(), kernel_initializer='he_normal',
      name='dense2')(x)
26
27    y_pred = Activation('softmax', name='softmax')(x)
```

Listing 4.1: Model implementation in Keras

It is possible to notice how Keras tends to be very high-level. The entire model is

defined in less than thirty lines, parameters and white lines included. Each line corresponds to a different layer, furthermore there are two *for-loops* since the parameters, required to initialize the layer, were the same.

In the code above the last layer is missing, this because CTC is defined differently in Keras:

```
def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)

    ....

loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred
    , labels, input_length, label_length])
```

Listing 4.2: CTC implementation in Keras

As it is shown in the code above, Keras does not support a direct implementation of CTC loss, therefore it is implement as Lambda layer. The important outcome from this implementation is that it certainly requires both train data and corresponding labels, but also their sizes.

Finally the model, and the optimizer, are defined by the following code:

```
model = Model(inputs=[input_layer, labels, input_length, label_length],
    outputs=loss_out)

sgd = SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True, clipnorm=5)

model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=sgd)
```

Listing 4.3: Model definitiona in Keras

CHAPTER 5

Case study

In this chapter it is going to be described the various use cases of this model. Several datasets have taken into account each of them corresponding with different use cases for the model.

5.1 Evaluation methods

In this chapter is going to be described the various use cases of this model, but before starting in the introduction of them, it is necessary to explain the evaluation methodology with which this model will be graded. There are many ways to evaluate OCR models which strongly depends on the problem the latter is facing. Since this model is meant to be flexible and adaptable to many situations, in this testing section two different types of accuracy evaluation are taken.

For the **first evaluation method**, it has been utilized the classical approach i.e. compute precision, recall and F1-score. Following the three formulas:

$$Precision = \frac{\#\ true\ positives}{\#\ true\ positives + \#\ false\ positives} \tag{5.1}$$

$$Recall = \frac{\#\ true\ positives}{\#\ true\ positives + \#\ false\ negatives} \tag{5.2}$$

$$F1\ score = \left(\frac{Recall^{-1} + Precision^{-1}}{2}\right)^{-1} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

Where:

- **true positive** are the characters correctly predicted.
- **false positives** are all the incorrectly predicted characters.
- **false negatives** are all the characters in the labels that are not predicted.

The number(#) refers to all the characters in the dataset taken into consideration.

The **second evaluation method** is different compared with the previous one and it is usually applied to the OCR post-processed output. This measure adopts the *Levenshtein distance*, which expresses the "distance" between two strings. This method is a good way to obtain a more *qualitative* view of the overall accuracy, which is the reason of its use. This second method, called $Accuracy_2$ in this case, is defined as follow:

$$Accuracy_2 = \frac{1}{1 + lev_{s,s'}(|s|, |s'|)} \tag{5.4}$$

Where $lev_{s,s'}(|s|, |s'|)$ is the *Levenshtein distance*, also called *edit distance*, between two strings s, s'(of length $|s|$,$|s'|$ respectively) define as follow:

$$lev_{s,s'}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{s,s'}(i-1,j) + 1 \\ \text{lev}_{s,s'}(i,j-1) + 1 \\ \text{lev}_{s,s'}(i-1,j-1) + \mathbf{1}_{[a_i \neq b_j]} \end{cases} & \text{otherwise.} \end{cases} \tag{5.5}$$

where $\mathbf{1}_{[a_i \neq b_j]}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise. Differently from the F1 score, here the eq. 5.4 is compute for each pair of label and corresponding prediction and the arithmetical mean of all these values is taken as final parameters.

5.2 Use Cases

Each use case is represented with a *different* dataset on which the model has been **trained**. Before starting with the introduction of the use cases, it is crucial to specify that this model is meant to be ductile, flexible, adaptable to various scenarios and different problems. For this reasons instead of test the model over one single dataset has been opted to use several different datasets with increasing complexity:

- **First use case**: Plate recognition, with Supervisely.com dataset.[48]
- **Second use case**: Python-generated images of words, with the same font, different font sizes.
- **Third use case**: Python-generated images of words, with the same font, different font sizes and three different types of noise.
- **Fourth use case**: Similar to third use case, but with different fonts.

In the next sections these use cases will be explained in detail. There are some common characteristics, first the image used was always rescale to 128x32. Furthermore except for the first use case, in which the alphabet is reduced to the alphanumeric characters appear in the training set, the other three use cases, utilize lowercase letters from English alphabet. This is because adding new characters is as easy as change a String containing the whole set of characters.

5.2.1 First use case

As already mentioned, this use case deals with the plate recognition problem, with Supervisely.com dataset.[48]

In the figured 5.1 there are some examples of input images. First fact to notice is that the plates are pretty similar, with different codes. The output is fixed to a 8-character string, with always the same pattern "LIIILLII", where L stays for letter and I for integer. Moreover also the sizes remain always to the same fixed motif. The training set amounts on 10821 images, while the test set is 560.

Regarding the conducted experiment, first no validation set was used, this was because after training for only 1 epoch with the full training set, the accuracy of the model on the testing set was 1, i.e. no errors. This shows the potentially of the model among data with fixed length and patterns. This use case is slightly related to the objective of this model was meant for, however it is useful to prove the potentially of the model itself and it is a simple and practical starting point for the testing section.

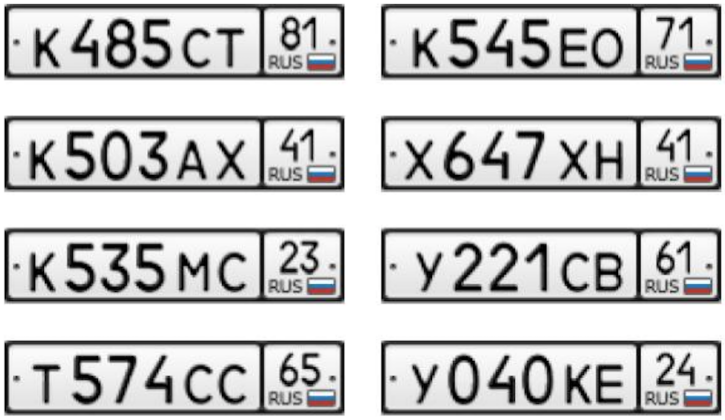

Figure 5.1: Some training images from the dataset used in the first use case.

5.2.2 Second use case

In this second case the scenario considerably changes, running into the detection of words. Comparing to the previous case, here the model is facing words, which means different amount of characters and so different length. In this case, only one font is used (OpenSans-Semibold), but with different font sizes. Furthermore the word is positioned randomly within the iamge space. In the figured 5.2 there are some samples.

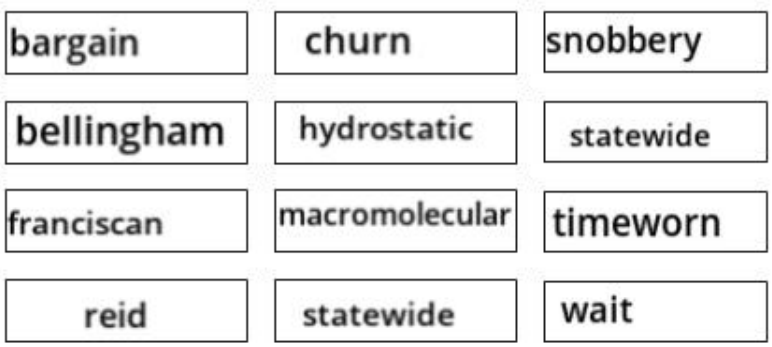

Figure 5.2: Some examples from the second use case dataset. In this dataset, only the (OpenSans-Semibold) is used, but with different font sizes. Furthermore the word is positioned randomly in the rectangle.

The training set amounts on 14000 images, the validation set 1000, while testing set contains 1000 samples.

Regarding the conducted experiment on the testing set the result was very valuable, almost perfect, with an accuracy of $F1\ score = 0.9993$.

5.2.3 Third use case

This use case tries to move the complications a bit further adding a small amount of noise in the images. As in the previous use case, here only one font is used (OpenSans-Semibold), but with different font sizes, and in random positions as well. There are three different types of noise, *Gaussian*, *Salt and Pepper* and *Speckle* noise. In the figure 5.3 there are some examples, in the first column are shown the images with Gaussian noise. In the second column, even if not very notable, there are images, in which have been added a small amount Speckle noise. Finally in the last column is filled by images with a little salt and pepper noise.

From the human perspective, this is just a small amount of noise and the word is easily distinguishable from the background, but from a computer view, which sees images as numbers in an array, the image with noise is completely different from the image without noise. However deep learning models have been proved to act very effectively, and also this model is built to rightly handle this small amount of noise.

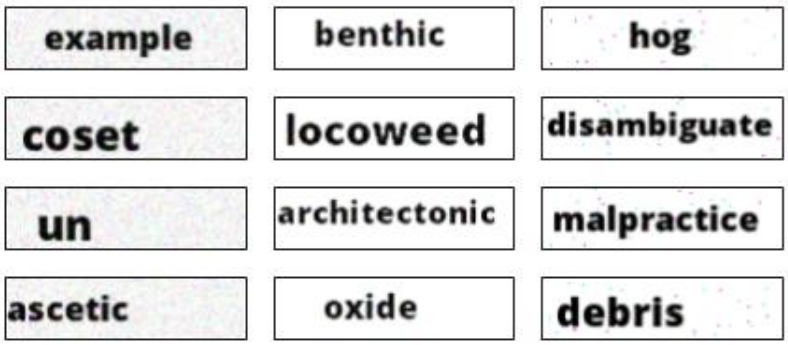

Figure 5.3: Use case 3 samples images. Each column shown different types of noise, *Gaussian*, *Speckle*, *Salt and Pepper* respectively.

The dataset for this use case is composed with these noise images, each class of noise has more or less the same amount of samples. The training set amount on 14000 images, while validation and testing set have both 1000 samples.

In the conducted experiment, the model has been trained for 2 epochs, and it shown pretty good results as expected. In fact the accuracy was almost perfect, specifically $F1\ score = 0.9998$. Which in other words means the model mispredicted only two or three characters out of one thousand word samples.

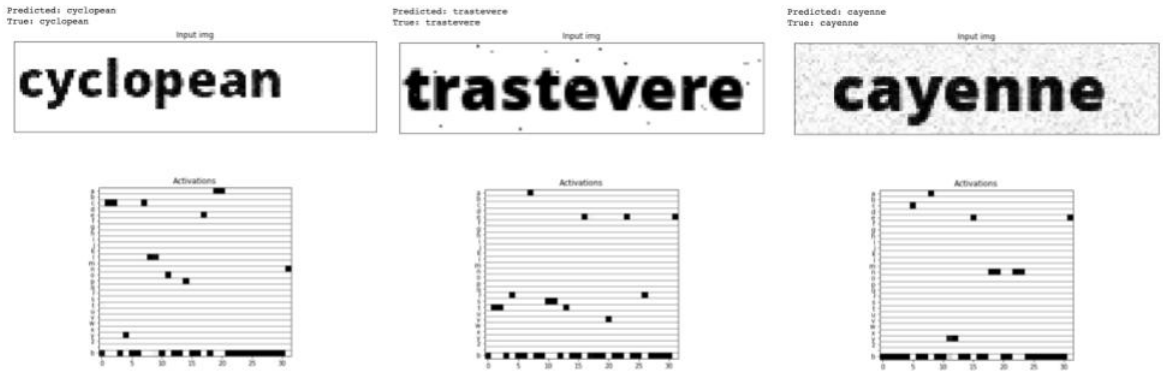

Figure 5.4: Use case 3 activation maps for different images. On the *y axis* there the alphabet plus the 'blank' on the bottom. While on the *x axis* the distribution probability for each time step.

5.2.4 Fourth use case

The size of the dataset remains unchanged from the previous ones, 14000 samples for the training set, 1000 for both validation and testing set. As in the previous use cases, these images brings a small amount of noise and the word is randomly positioned within the area. Even in this use case the complication is moved much further. The major and important difference is, instead of a single fonts, there are five different fonts. In the figure 5.5 there are some examples.

Figure 5.5: Use case 4 samples.

In the figure 5.5, there are both uppercase and lowercase fonts, however in the experiment all the characters are handled as lowercase. This again for increasing the adversity, treating

the uppercase character as a different font.

An **observation** for this use case is that the amount of training set data may be insufficient for the given problem. Notice that there are five fonts and 3 type of noise, so even though slightly similar, it is possible to divided dataset in 3x5 classes, and so 1000 samples for each class, which thinking about the amount of training samples in the previous use cases, this makes this use case even more challenging.

Running this experiments brought interesting results. The model has been trained for 3 epochs. Over the testing set the accuracy dropped by 3% with $F1\ score = 0.9731$. Analysing much deeper, the model demonstrated a good accuracy almost on each font(except one), but analysing the accuracy on each font separately, it was slightly lower than the previous use cases, even for the font previously utilized. The problem was particularly related to one font when the speckle noise was applied on it, examples are in the figure 5.5, images "maverick" and "whimsy". The model mispredicted many of those images characters. This drop even further the $accuracy_2$ value, because the latter looks more word level, instead of character level as $F1\ score$ does.

These results were not unexpected, since it was already anticipated due the lack of training data. However, also another problem turned out during the training phase, in fact the loss on the third epochs was slightly higher on the validation set compared to the training set. This may mean the model is marginally prone to overfitting.

5.2.5 Results Summary

To summurize, all those different use cases, each of them with a different dataset, brought good results, as it is possible to see in table 5.1.

Use Case	# train	# test	Precision	Recall	F1-score	$Accuracy_2$
1	10506	506	1.0	1.0	1.0	1
2	14000	1000	1.0	0.9987	0.9993	0.9960
3	14000	1000	0.9998	0.9997	0.9998	0.9984
4	14000	1000	0.9569	0.9899	0.9731	0.9226

Table 5.1: Accuracy over the different datasets in the various use case. Each use case brings a different dataset. # refers to the number of samples.

The overall testing process has been developed having in mind a growing difficulty. From a "static" problem, i.e. the first use case, to a variegated one with different fonts, shapes

and noise, i.e. the fourth use case. In the first three use cases the accuracy was pretty high and this testing process shown the robustness and effectiveness of the model when it deals with this kind of problems. While on the other hand, although the use case was extremely more challenging compared to the others, the model shown an honestly good accuracy, and most importantly shows the flaws of the model.

CHAPTER 6

Conclusions

This chapter will describe the achieved goals done by the master thesis, following some the key points developed in the project and finally some ideas for future improvements.

6.1 Achieved Goals

In this master thesis several goals have been achieved:

- Researches on deep learning for Computer Vision, looking at the newest achievements and state-of-the-art papers and works.
- Exploration over the state-of-the-art Text Analytic platforms and OCR software which are commercialized.
- Analysis and understanding how Optical Character Recognition works with a deep learning approach.
- Implementation of a model able to do OCR with high accuracy on different use cases.

This work was divided in two parts. The initial sections of this project were focused on exploring and understating the state-of-the-art regarding computer vision and text analytic platform. The starting point was a brief analysis on the most important researches over last year, useful to have a general idea on the overall scenario and introduce to the topic. Following an exploration on the last outstanding research on Deep Learning for Computer Vision view was performed. Upon this, the research moved into a more specific topic i.e. text analytic platforms and OCR software. This part shown a general and interesting scene on the available products.

In its second half, it was introduced the model meant to solve the principal initial requirement, i.e. the implementation of a model able to do optical character recognition with high accuracy over different use cases. All the process led to the creation of a model using a deep learning approach merging convolutional neural network, recurrent neural network and many other recently-discovered technologies. Finally in the experiment section the effectiveness of the model has been proved, and also the flaws have been pointed out, in order to put some guide-lines to improve the model itself in the future.

6.2 Conclusions

In the last decade, many progress on computer vision and machine learning has been achieved thanks to the deep learning approach. New researched and new state-of-the-art are accomplished with an extremely fast rate and keep up with them is difficult but necessary.

It is mainly thanks to these last researches, the project has fulfilled the proposed objectives. First many researches and explorations over the actual state-of-the-art have been completed. Second, using the skills acquired from those, a model able to do optical character recognition with high accuracy has been implemented.

The entire project was created from a real necessity of a company. The real achievement and added value of this project stays on the fact it is able to take a "real" problem and solve it, using already-existing solutions, modelled to this specific purpose. Furthermore this problem-solving process shows how effective the exploring the actual state-of-the-art could be, since the model this project provided is strongly based on those researches adapted to this project necessities.

6.3 Future Improvements

The first idea, from which this project is started, wants this model to be integrated with a word detector. Therefore the very next step is to test model's behaviour integrated with the word detector. Regarding the model itself; it has been shown positive results but it needs to be improved to handle more general problems. In the short time the actions need to be focused on improving the accuracy with different classes and reduced overfitting. For instance, following the heuristic which says "the deeper the model is, the better", may result into an improvement of model's effectiveness. In particular, adding more convolutional layers may help the model to handle more features in the input image, hence more fonts, symbols. Furthermore to prevent overfitting many techniques could be utilized, for example dropout. On the other hand in the long time the decoding algorithm needs to be improved with more advanced decoders such as beam-search decoding, prefix-search decoding or token passing, which also use information about language structure. Moreover the same model may be effectively trained to handle hand-written texts.

Bibliography

[1] a9t9 software GmbH. Ocr.space v3.32. https://ocr.space/. [Online; accessed 13-May-2019].

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Amazon Web Services Amazon. Amazon comprehend. https://aws.amazon.com/comprehend/. [Online; accessed 13-May-2019].

[4] Amazon Web Services Amazon. Amazon rekognition. https://aws.amazon.com/rekognition/. [Online; accessed 13-May-2019].

[5] Amazon Web Services Amazon. Amazon textract. https://aws.amazon.com/textract/. [Online; accessed 13-May-2019].

[6] Christian Bartz, Haojin Yang, and Christoph Meinel. STN-OCR: A single neural network for text detection and text recognition. *CoRR*, abs/1707.08831, 2017.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.

[8] Yu-Wei Chao, Sudheendra Vijayanarasimhan, Bryan Seybold, David A. Ross, Jia Deng, and Rahul Sukthankar. Rethinking the faster R-CNN architecture for temporal action localization. *CoRR*, abs/1804.07667, 2018.

[9] Dongdong Chen, Jing Liao, Lu Yuan, Nenghai Yu, and Gang Hua. Coherent online video style transfer. *CoRR*, abs/1703.09211, 2017.

[10] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[11] François Chollet et al. Keras. https://keras.io, 2015.

[12] The Apache Software Foundation. Opennlp.

[13] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016.

[14] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[15] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[17] Google Google. Colaboratory. `https://colab.research.google.com/notebooks/welcome.ipynb`, 2015. [Online; accessed 28-May-2019].

[18] Google Cloud Google. Ai machine learning products, natural language api. `https://cloud.google.com/natural-language/docs/basics`. [Online; accessed 13-May-2019].

[19] Vision AI Google, Google cloud. Vision ai. `https://cloud.google.com/vision/`. [Online; accessed 13-May-2019].

[20] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[21] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[22] Rapidminer inc. Rapidminer, text mining. `https://rapidminer.com/solutions/text-mining/?utm_term=text+mining`. [Online; accessed 13-May-2019].

[23] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. `http://crcv.ucf.edu/THUMOS14/`, 2014.

[24] Project Jupyter. Project jupyter. `https://jupyter.org/`. [Online; accessed 20-May-2019].

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[27] Xiang Li, Wenhai Wang, Wenbo Hou, Ruo-Ze Liu, Tong Lu, and Jian Yang. Shape robust text detection with progressive scale expansion network. *CoRR*, abs/1806.02559, 2018.

[28] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. *CoRR*, abs/1901.01892, 2019.

[29] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. *CoRR*, abs/1802.06474, 2018.

[30] librosa development team. librosa. `https://librosa.github.io/librosa/`. [Online; accessed 20-May-2019].

[31] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[32] QSR International Pty Ltd. Nvivo. `https://www.qsrinternational.com/nvivo/nvivo-products`. [Online; accessed 13-May-2019].

[33] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. *CoRR*, abs/1703.07511, 2017.

[34] Pengyuan Lyu, Minghui Liao, Cong Yao, Wenhao Wu, and Xiang Bai. Mask textspotter: An end-to-end trainable neural network for spotting text with arbitrary shapes. *CoRR*, abs/1807.02242, 2018.

[35] Microsoft. Microsoft azure, text analitics. `https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/`. [Online; accessed 13-May-2019].

[36] Numpy. Numpy developers. `https://www.numpy.org/`. [Online; accessed 20-May-2019].

[37] Tesseract OCR. Tesseract open source ocr engine, v4. `https://github.com/tesseract-ocr/tesseract`. [Retrieved 29 Oct 2018; Online; accessed 13-May-2019].

[38] OpenCV. Opencv team. `https://opencv.org/`. [Online; accessed 20-May-2019].

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

[40] NLTK Project. nltk. `https://www.nltk.org/`. [Online; accessed 20-May-2019].

[41] PyTorch. Pytorch. `https://pytorch.org/`. [Online; accessed 20-May-2019].

[42] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[45] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015.

[46] Bharat Singh, Mahyar Najibi, and Larry S. Davis. SNIPER: efficient multi-scale training. *CoRR*, abs/1805.09300, 2018.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[48] Supervise.ly. Latest deep learning ocr with keras and supervisely in 15 minutes. *hackenoon.com, medium.comn*, nov 2017.

[49] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *CoRR*, abs/1808.06601, 2018.

[50] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[51] 51-56 Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference. Pandas. `https://pandas.pydata.org/`. [Online; accessed 20-May-2019].

[52] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. *arXiv e-prints*, page arXiv:1805.08318, May 2018.