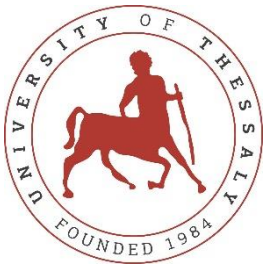


UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Real-time Text Recognition (shop signs) using Neural Networks and GPU Accelerators

Diploma Thesis

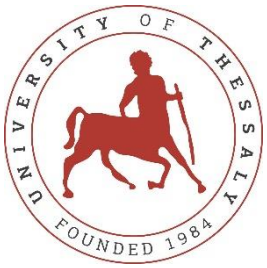
By: Nikolina Eleni Zisouli

Supervisor:
Nikolaos Bellas

2nd Committee Member:
Spyros Lalis

3rd Committee Member:
Gerasimos Potamianos

Volos 2020



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Real-time Text Recognition (shop signs) using Neural Networks and GPU Accelerators

Diploma Thesis

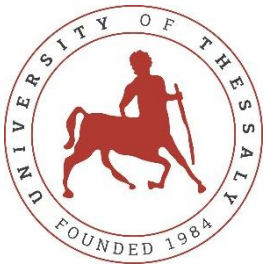
By: Nikolina Eleni Zisouli

Supervisor:
Nikolaos Bellas

2nd Committee Member:
Spyros Lalis

3rd Committee Member:
Gerasimos Potamianos

Volos 2020



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αναγνώριση Κειμένου (επιγραφών καταστημάτων) σε Πραγματικό Χρόνο με Νευρωνικά Δίκτυα και Χρήση Επιταχυντών GPUs

Διπλωματική Εργασία

Της: Νικολίνας Ελένης Ζησούλη

Επιβλέπων:
Νικόλαος Μπέλλας

2ο Μέλος Επιτροπής:
Σπύρος Λάλης

3ο Μέλος Επιτροπής:
Γεράσιμος Ποταμιάνος

Βόλος 2020

“Just because something doesn’t do what you planned it to do doesn’t mean it’s useless.”

–Thomas Edison

ACKNOWLEDGEMENTS

I would like to thank Prof. Christos Antonopoulos, who was always available and willing to help whenever I ran into trouble or had a question about my research. He consistently allowed this Thesis to be my own work but guided me in the right direction whenever he thought I needed it. I would also like to thank my Thesis supervisor Prof. Nikolaos Bellas and other committee members Prof. Spyros Lalis and Prof. Gerasimos Potamianos for the confidence they showed to me.

Additionally, I would like to express my acknowledgements to my parents and my family supporting me and encouraging me throughout the years of studies in the University of Thessaly. This accomplishment would not have been possible without them.

Finally, I could not omit my beloved friends for their unconditional support and belief in my potential all these years. Their motivational words and encouragement have been crucial for this achievement.

Thank you all!

ABSTRACT

Nowadays, when people are more reliant on technology than ever, the capability to search information with the use of smart phones is becoming essential. For this reason, we believe that reverse image search and, especially, their text detection and recognition function is a search engine tool that needs to be further advanced. In the current Thesis we assess two existent NNs, one on text detection and another on text recognition, on multilingual scene text images. The first NN is the EAST text detector, which performs considerably well on locating text in scene text images. For text recognition, we decided to train an end-to-end trainable NN, the CRNN, first on a Greek and English alphabet and, then, on an alphabet containing letters from four Latin languages, digits and symbols. Specifically, we emphasized on all of the training parameters and hyperparameters to achieve the best possible accuracy of correct character prediction, which is 84.6%. Finally, we integrated the two NNs into one system that detects scene text on images and recognizes the words depicted. This system has, additionally, been optimized in order to execute in real time, explicitly at 0.32 seconds per image.

ΠΕΡΙΛΗΨΗ

Στις μέρες μας, οπότε οι άνθρωποι βασίζονται στην τεχνολογία περισσότερο από ποτέ, η δυνατότητα αναζήτησης πληροφοριών με τη χρήση έξυπνων τηλεφώνων γίνεται απαραίτητη. Για αυτό το λόγο, πιστεύουμε ότι η αντίστροφη αναζήτηση εικόνων και, ιδιαίτερα, η λειτουργία τους για εντοπισμό και αναγνώριση κειμένου είναι ένα εργαλείο των μηχανών αναζήτησης που χρειάζεται να αναπτυχθεί περαιτέρω. Στην παρούσα Διπλωματική εργασία αξιολογούμε δυο υπάρχοντα Νευρωνικά Δίκτυα (ΝΔ), ένα για τον εντοπισμό κειμένου και ένα για την αναγνώριση κειμένου, πάνω σε εικόνες πολύγλωσσου σκηνικού κειμένου. Το πρώτο ΝΔ είναι ο ανιχνευτής κειμένου EAST, ο οποίος αποδίδει αρκετά καλά στον εντοπισμό κειμένου σε εικόνες. Για την αναγνώριση του κειμένου, αποφασίσαμε να εκπαιδεύσουμε ένα από άκρη σε άκρη εκπαιδεύσιμο ΝΔ, το CRNN, πρώτα σε ένα ελληνικό και αγγλικό αλφάβητο και, στη συνέχεια, σε ένα αλφάβητο που περιλαμβάνει γράμματα από τέσσερις λατινικές γλώσσες, ψηφία και σύμβολα. Συγκεκριμένα, δώσαμε έμφαση σε όλες τις παραμέτρους και υπερπαραμέτρους της εκπαίδευσης για να πετύχουμε την καλύτερη δυνατή ακρίβεια σωστής πρόβλεψης χαρακτήρα, η οποία είναι 84,6%. Τελικά, ενσωματώσαμε τα δυο ΝΔ σε ένα σύστημα που εντοπίζει σκηνικό κείμενο σε εικόνες και αναγνωρίζει τις λέξεις που απεικονίζονται. Αυτό το σύστημα έχει, επιπλέον, βελτιστοποιηθεί έτσι ώστε να εκτελείται σε πραγματικό χρόνο, αναλυτικά σε 0,32 δευτερόλεπτα ανά εικόνα.

CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

In recent years, people become more and more dependent on technology, especially on smart phones. New applications are being released every day, making the use of smartphones a necessity; people search for information about anything on their mobile phones. According to [1] and [2], 46% of all Google searches are looking for local information and 88% of this kind of searches on a smartphone result in a visit or a call to the business. This fact emphasizes the need to make not only search engines but also informative mobile applications such as Yelp and Foursquare more functional for the users. An efficient way to do this is by introducing reverse image search.

Reverse image search is a query method in which the user is giving the search engine an image file and expects to find results relevant to the image [3]. Many search engines like Google, Yandex and Bing have already incorporated it in their platforms, however we think it is a market demand to introduce reverse image search in all types of mobile applications, but especially in business review platforms such as Yelp, Trip Advisor and Better Business Bureau. That is because, as Watson explains [4], people nowadays tend to prefer using their phone cameras as a note taking tool rather than typing, let alone writing memos and annotations. The same tendency is, also, evident in the way people perform their web searches, especially when they use mobile devices.

Apart from finding information about the image itself, reverse image search can, also, be very enlightening about the content of the image; it can be used to provide information about what is depicted in the image. This is why reverse image search can be quite beneficial for blind and visually impaired people who could use reverse image-enabled applications as assistive technology to identify things around them. A more specific scenario to which reverse image search can be applied is when users are outdoors, in an urban environment and want to find information about the stores around them, such as operating hours, reviews and ratings.

In order to create a system which effectively identifies stores depicted in images it is necessary to have access to an abundance of images with shop signs, namely scene text images. Scene text is a term that describes “the text that appears in an image captured by a camera in an outdoor environment” [5] and scene text images are the images that depict scene text. Nowadays, a plethora of scene text images is available due to the widespread use of smartphones with good quality cameras. In fact, as Ritcher demonstrates in [6] approximately

1.2 trillion photos were taken in 2017, 85% of which were captured by smartphones, and this number keeps increasing every year, as seen in Figure 1. Hence, it is evident that we now have the data needed to produce systems that can recognize stores seen in images.

Shop sign recognition is a task related to computer vision and it includes detecting and recognizing text from the images. In particular, a typical usage scenario has the user capturing an image or a video using a camera and giving it as input to the system. The system, after processing the data, outputs the text of the shop signboards found in the image or each frame of the video. The fact that processing the data is a procedure which encompasses first locating the text present in the image and then recognizing the word or words found in the text, is the reason why the implementation of such systems can be divided into two parts, text detection and text recognition.

When it comes to text detection, the methods used can be either image based or frequency based. More specifically, according to the first method the image is partitioned into segments within which the text is detected [5], and takes advantage of machine learning techniques such as support vector machines (SVMs) [7] and convolutional neural networks (CNNs) [8]. On the other hand, in the case of frequency-based methods, given that the text of the image

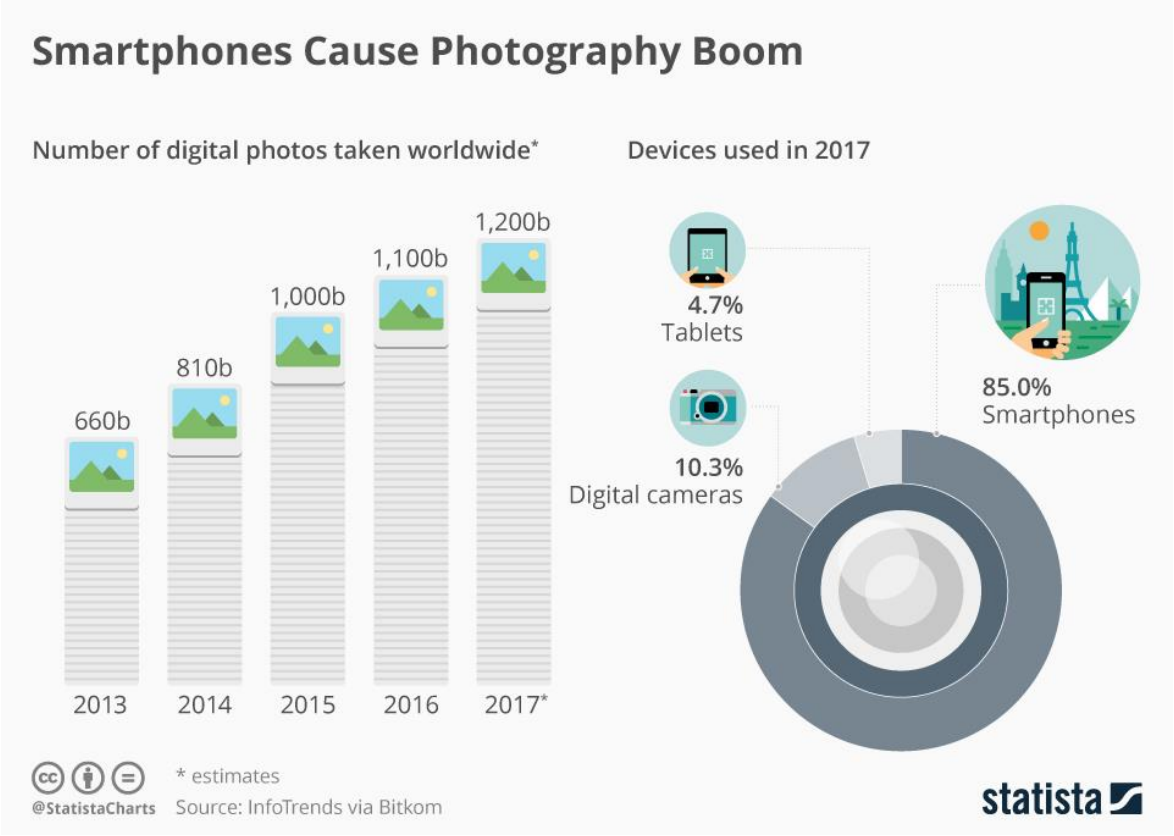


Figure 1. Number of digital photos taken worldwide every year [6]. It is evident that every year this number keeps increasing.

has high frequency components, the approach involves performing discrete Fourier transform (DFT) [9] or discrete wavelet transform (DWT) [10] to separate the text regions from the non-text ones.

Regarding text recognition, the approaches can be either top-down or bottom-up. In the latter case the image is segmented with a binarization algorithm and each segment is going through a recognition network [5]. On the contrary, the top-down approach is segmentation-free and uses a dictionary in order to identify the words.

Bartz *et al.* [15] mention that one of the most recent solutions to text recognition researchers have studied is using deep learning and, specifically, deep neural networks (DNNs). In particular, since the data discussed are images, it is preferred to use CNNs. As seen in Figure 2, CNNs are organized as 3-dimensional structures, which cluster the neurons into groups, each of which examines the image for a certain feature [16]. In other words, a CNN processes the elements of the image in order to logically interpret what they depict [16], in this case words or letters.

Besides the solution mechanism, in text recognition applications it is, also, important to work in real time, to enable users to have the results at the moment they ask for them. In order to achieve that, many optimization methods exist, some of which are algorithmic, that can either be deterministic or stochastic, while others include using special hardware platforms purposed for high performance computing (HPC). More specifically, attaining high performance for the system is a task which necessitates the use of accelerators like GPUs functioning as coprocessors for the extensive data workload [17]. For instance, when using

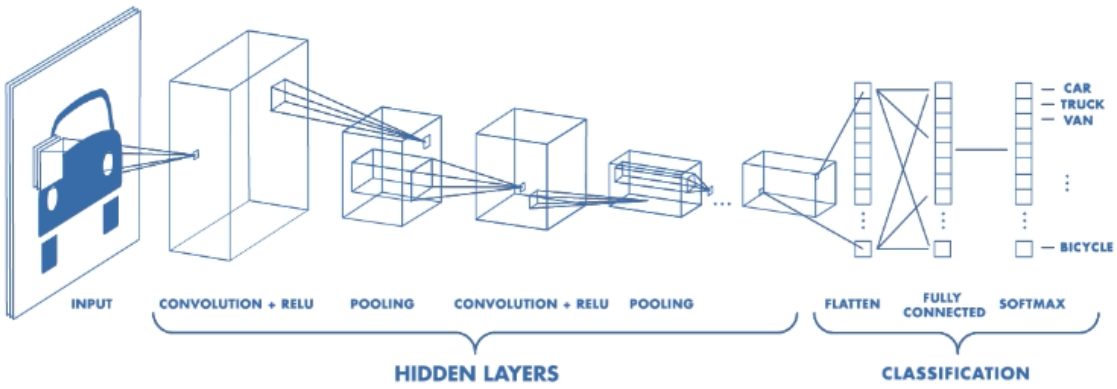


Figure 2. The 3-dimensional structure of a CNN [33]. In this example the CNN is used to classify types of vehicles.

a GPU, the computations that are performed on different variables of the code are being parallelized to kernels. This technique boosts the performance of the code and minimizes the computational time, thus usually producing real time results for the users.

In this Thesis, we approached the issue of text detection and recognition for shop signs with character sets from more than one language. Initially, our target use case was videos and images captured in Greek cities, but, unfortunately, after an extensive search we could not find any background work implemented on scene text recognition that includes the Greek language. Thus, the goal of this Thesis is to assess text detection and recognition algorithms on videos and images captured in Greek cities. However, it was not possible to accomplish this objective due to the lack of labeled scene text image data with Greek characters which would be used for training any text recognition system. Therefore, instead of Greek images, we decided to use a character set that contains symbols and letters from four different languages, English, French, German and Italian.

Another goal of this Thesis is to design a system capable of operating on real life circumstances. This means robustness to bad lighting conditions or motion blurring, which can disturb the clarity of the signboard to be detected. For that purpose, we have used the ICDAR2019 dataset [18] for the robust challenge on multi-lingual scene text detection and recognition. In particular, we used only the images containing text in Latin languages and augmented these data by adding random rotation to the images.

As a result, we have evaluated the potentiality of a system consisted of two neural networks (NNs), one for text detection and another one for recognition, both of which were primarily developed to operate with just the English language. This system attained an inference performance of 3 frames per second on a Tesla K80 GPU and produces results for a character set of 151 letters, numbers and symbols. In particular, we use Zhou's *et al.* EAST text detector [19] in order to detect the parts of the image containing text; that is, the shop signs. Then, we train Shi's *et al.* convolutional recurrent neural network (CRNN) [20] so as to recognize numbers and letters. In order to achieve the latter, we use several different alterations of the ICDAR 2019 dataset, so we can, additionally, propose an optimal way for it to be successfully trained. An example this system's input and output is given in Figure 3, Figure 4.

Figure 3. Example of input images given to the system. The images are taken from the ICDAR2019 test dataset [18].

During our study, we created a dataset with annotated text images and apprehended the importance of the use of augmentation techniques in the dataset. Undoubtedly, data augmentation is a valuable tool when the available data for training a NN are not enough. Another important factor in the process of training a NN is the hyperparameters tuning, which can make the training quite more efficient. In regard to the results of training a NN, we not only comprehended how to discern the differences between having an underfit and an overfit of the data in the model, but also, we understood how the training data have a critical effect on the final output of the NN.

The rest of the Thesis is organized as follows. In Section 2 we describe the specific hardware and software features used during the experiment stage of the Thesis. Then, in Section 3

Figure 4. After input images have been processed, the system gives as output images with annotation seen here.

details about the NNs used are being elucidated and we thoroughly the dataset' s features needed for proper training of the CRNN. Moreover, we elaborate on the way the two NNs used are being combined for the creation of the final system. After that, in Section 4 we show the results of both each NN separately system and assess their accuracy. An additional evaluation is being made on the performance of the system and its parts independently. Finally, we discuss the work of other researchers creating systems for text detection and recognition in Section 5 and in 6 we summarize the whole work done for this Thesis and propose future improvements and upgrades for the system.

2. BACKGROUND

2.1. Software environment

We run our experiments on Ubuntu 18.04 and all the code is written in Python 2.7.

The text detection part makes use of the OpenCV 4.1 library which is an open source infrastructure for computer vision and machine learning applications. It was launched in 1999 by Intel as part of a project aiming to contribute to not only both vision research and knowledge, but also, vision-based commercial applications. As stated in the OpenCV webpage [21], it contains algorithms that can be useable as tools for detection, recognition, classification, even tracking of objects. Some of its image processing features include decision tree learning, gradient boosting trees, artificial NNs, SVMs and DNNs. In our work we have made use of a serialized version of a NN created with OpenCV as well as of functions for drawing and writing on an image. Along with these functions, we, also, adopted OpenCV variable types in our code, for processing the input images and saving the output.

The text recognition component utilizes another machine learning library, PyTorch 1.4. PyTorch is a deep learning framework that features not only tensor computing, but also deep neural networks (DNNs) [22].There are three main machine learning libraries, Tensorflow, Keras and Pytorch [23]. We chose to work with Pytorch, because it is easy to learn and has a simple, yet powerful interface. Another reason for selecting Pytorch over the other frameworks is the fact that it supports better data parallelism, as reported by Rohilla [20], which is very important for this Thesis, since we want to focus on the performance of our code.

In particular, Pytorch provides a low-level API, which is based on the Torch framework and is usually used when processing large datasets with high-performance models. It was primarily developed by Facebook's AI Research lab and it was released in October 2016. Pytorch's advantages include providing flexibility on its uses, great debugging capabilities and short training duration.

For performance optimization, we have made use of functions from OpenCL 1.2 to the text detector and from CUDA 10.2 to the recognition component. OpenCL is a framework which supports functional portability of code, that is the execution of programs on different heterogeneous processors and hardware accelerators [24]. It was initially developed by Apple and it was released by Khronos Group in 2009. It is used to improve both the speed

and the responsiveness of many types of applications by passing the most computationally intensive code on to accelerator devices so that it runs in parallel.

More specifically, OpenCL models the computing system as an assembly of computing devices, as shown in Figure 5.a, and, thus, it executes each of the code's functions in parallel on all or many of the devices' processing elements. This parallelism of the code is modelled, as demonstrated in Figure 5.b, as kernels that are executed in a multi-dimensional index space. OpenCL, also, allows the separate management of device and host memory for the programs it is running. Apropos the OpenCL language, it is a just-in-time compilation programming language, so that the applications using OpenCL are entirely transferable between different host devices.

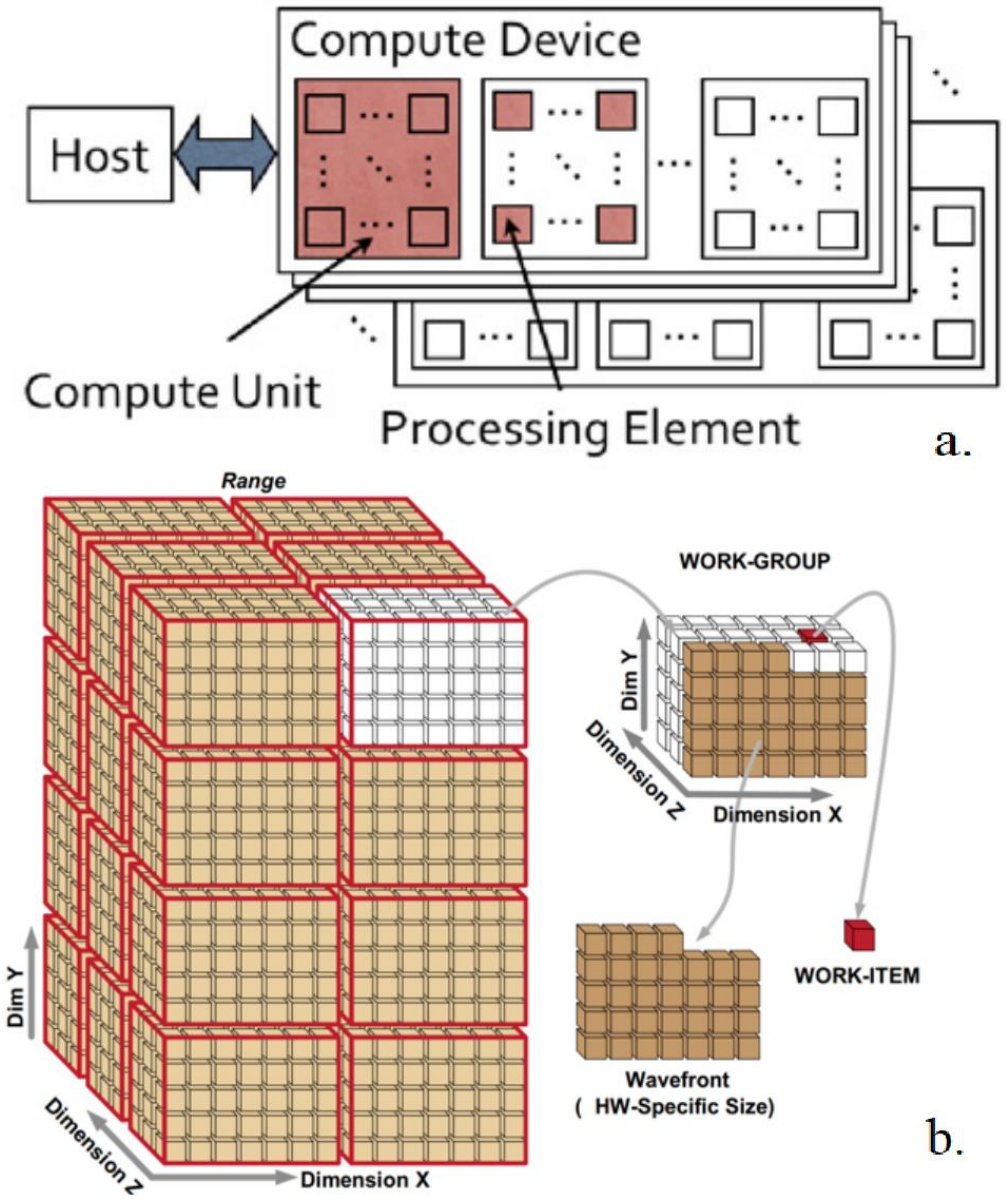


Figure 5. a. OpenCL's hardware view [34]. b. OpenCL's software view. [35]

On the other hand, CUDA is both an execution platform and an API for parallel computing on NVIDIA CUDA-enabled GPUs [25]. It includes not only parallel computing extensions to many programming languages but also drop-in accelerated libraries, like cuDNN and cudaTensor, to help accelerate the execution of specific classes of applications. One of its most remarkable characteristics is the shared memory region that can be shared between threads.

There are applications from many different scientific fields, such as astronomy, biology, physics, data mining and finance, that have been GPU-accelerated with the CUDA ecosystem. Software developers can use either of the two APIs that CUDA provides, the CUDA Driver API, which is low level, and the CUDA Runtime API, which is higher level. There is a specific extension for C/C++ programmers, the CUDA C/C++ language, while Fortran programmers can use the CUDA Fortran extension. Aside these extensions, there are, also, third party wrappers for CUDA support on other programming languages, like Python, Java, Ruby and Perl.

In comparison to OpenCL, CUDA is not supported in as many applications, however CUDA is significantly more optimized than OpenCL for Nvidia GPUs [23]. Although OpenCL is more portable than CUDA, OpenCL programs can be significantly longer, exposing the programmer to more details. Besides that, the two frameworks operate having the same basic hardware and software models, with the difference that OpenCL's support offers lower performance boosts than CUDA.

2.2. Hardware platform

Concerning the hardware platform, we used a server for both training and inference. This server has two Intel Xeon CPUs at 2.30 GHz, that include 28 hyperthreaded CPU cores, with two threads per core, as indicated in Table 1. The RAM in the server is 128 GB and it uses little-endian byte order. Moreover, the server has a NVIDIA Tesla K80 GPU, which we use for optimizing performance. This accelerator is server-optimized with peak single-precision performance of 8.73 Tflops and peak double-precision one at 2.91 Tflops.

CPU	Intel Xeon @ 2.30 GHz
CPU cores	28 (2x14)
Threads per core	2
RAM	128 GB
GPU	NVIDIA Tesla K80
GPU chips	2
GPU memory	24 GB
GPU memory bandwidth	480 GB/s
CUDA cores	4992

Table 1. Hardware specifications of the server used for development and experimental evaluation.

3. TEXT DETECTION AND RECOGNITION IN REAL-LIFE SCENES

The problem is divided into two parts, text detection and text recognition. Accordingly, we decided to use two different NNs, one for each part of the problem, and then combine them in order to create a system that detects scene text from images or videos and recognizes the characters in the text. Figure 6 outlines all the steps discussed in the rest of this Section.

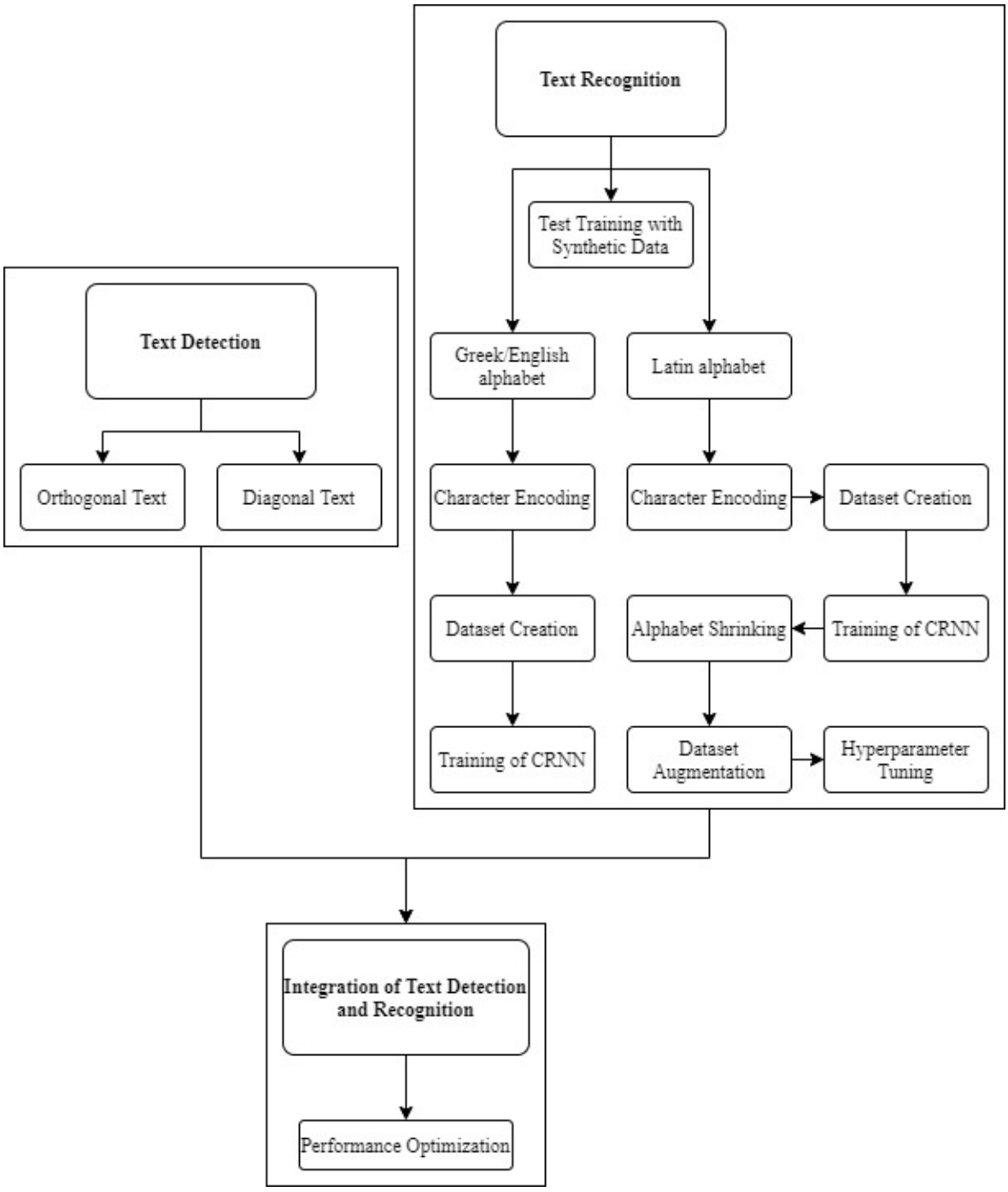


Figure 6. Diagram of all the steps discussed in Section 3.

3.1. Text detection

We decided to assess Zhou's *et al.* EAST NN-based text detector [19]. In particular, Zhou *et al.* have approached the scene text detection problem with a two-stage pipeline. The first stage includes a multi-channel fully convolutional NN (FCN), depicted in Figure 7. It consists of 3 components: the feature extractor stem, the feature-merging branch and the output layer. The FCN produces the test score map and the text geometries which are, in the second stage, given as input to a non-maximum suppression (NMS) algorithm. This NMS algorithm merges the geometries from nearby pixels, by averaging the quadrangles to merge coordinates, and outputs the final text regions of the image.

The reason why we chose to utilize the EAST text detector is the superior accuracy on detecting the existence of text on images and speed it achieves compared to other existing text detectors. In terms of accuracy, the authors emphasize that their detector attains an F-score of 0.7820 in ICDAR 2015 Challenge 4 [27] and an F-score of 0.8072 when tested at multi scales (image is given as input in different sizes, in order for text of different sizes to be detected) in the same challenge. Concerning performance, the EAST text detector performs at an average of 16.8 FPS on a NVIDIA Titan X GPU with images of 1280x720 resolution.

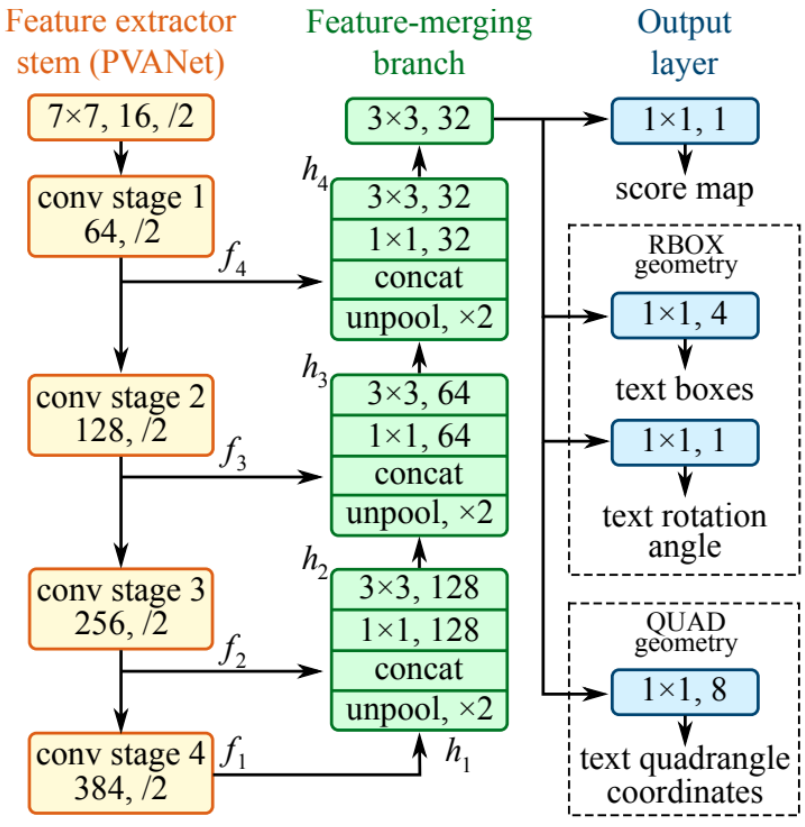


Figure 7. Structure of EAST text detector's FCN.

We reused the serialized EAST model by Rosebrock [28], which is the EAST model with trained final weights in a binary format, and applied it to his OpenCV scripts for image and video inputs. In the case, where the text is in a horizontal or orthogonal orientation we used the script as is and found that it gives the most accurate results with the best speed at minimum confidence of 0.02 and when the image is resized to width and height of 704 pixels. Samples of the output can be viewed in Figure 8.

3.1.1. Diagonal text

In some cases of our test images the orientation of the text is not horizontal, but diagonal, thus a rectangle bounding box cannot cover the whole text. Such cases are depicted in Figure 9. For this reason, we had to alter the shape of the predicted region from rectangle to trapezoid. The serialized EAST model outputs the upper left and the lower right points of the text, so with these two points we created a trapezoid that can contain any type of angled text. This is achieved by keeping the text's height in the right side of the trapezoid and making both the right angles obtuse. In this way, the trapezoid can cover the whole text whether it extends upwards or downwards from right to left. The final results of this script can be observed in Figure 10. Concerning the tuning of the parameters in this case, we experimentally identified that a minimum confidence of 0.02 and the image being resized to

Figure 8. Examples of text detector's output when it is given ICDAR 2019 test images as input.

Figure 9. In some cases the text in the images is neither horizontal nor vertical, so rectangular is not always the apropriate shape for the bounding box.

704x704 dimensions, just like in the case of the rectangles, also makes the script perform best in terms of both accuracy and speed.

3.2. Text recognition

For the text recognition part, we chose Shi's *et al* CRNN [20], a sequence recognition NN. In our case, we want to recognize specific sequences, namely words comprising a multi-lingual character set. The CRNN is a combination of a deep CNN and a recurrent NN (RNN), the architecture of which is depicted in Figure 11. Specifically, it takes as input an image that is first processed by the convolutional layers of the NN, which extract a sequence of feature vectors generated from the feature maps that are produced by the convolutional layers. Then, the feature sequence goes through the network's recurrent layers, which comprise a deep

Figure 10. In the cases of diagonal text, shown in Figure 9, we propose this trapezoid bounding box.

bidirectional long short-term memory (LSTM) network, and they output the label distribution for each frame of the image. In other words, the recurrent layers assign a label from the alphabet to each feature vector from the feature sequence, creating the label distribution. The last layer is a transcription one that translates the LSTM's predictions to the final predicted label sequence.

The fact that the CRNN is created to predict any label sequence coming from a given alphabet is the reason why we found it the most pertinent solution for multi-lingual text recognition from shop signs. Quite often shop names consist of fictional words or surnames that cannot be found in a lexicon. Apart from that, after reviewing the literature [29], [30], [31] we observed that the CRNN achieves accuracies very close to or sometimes even better

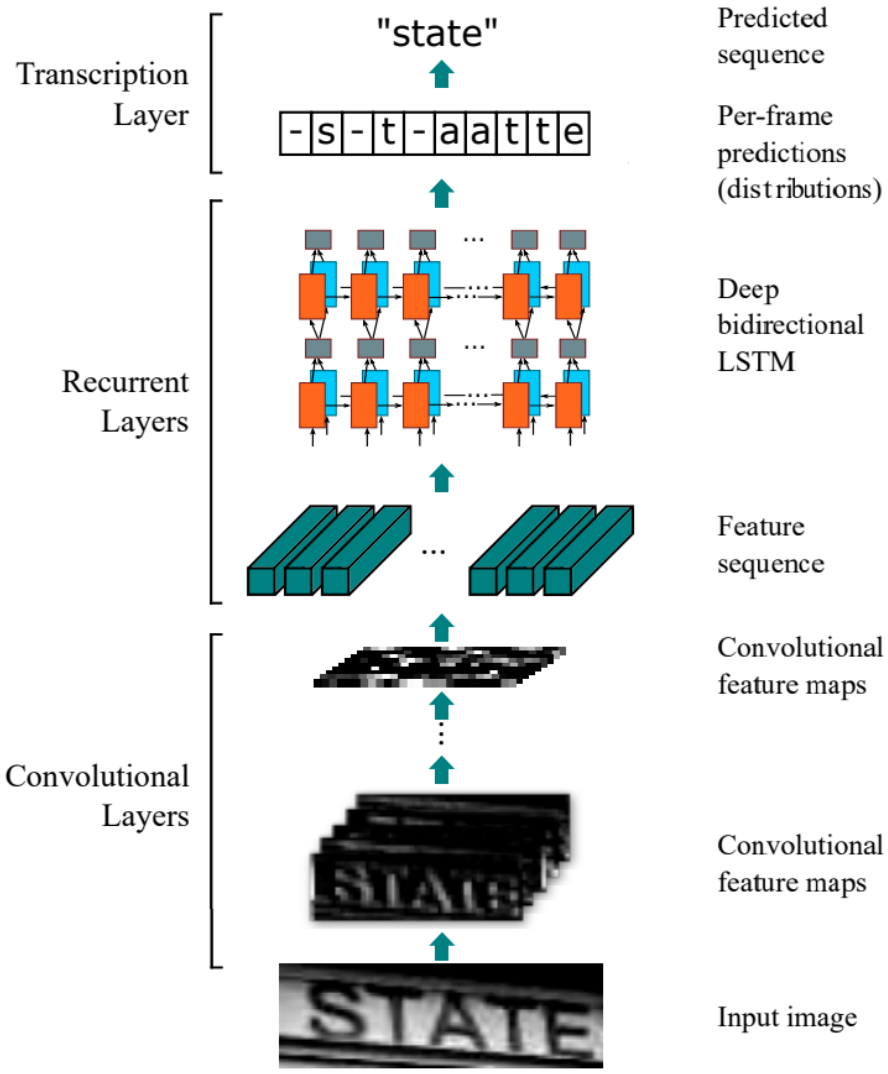


Figure 11. CRNN's structure.

than the reported accuracies of other existing text recognition methods without the use of lexicon.

In order to accomplish the multilingual scene text recognition, we first had to train the network on the particular alphabet we wanted it to recognize. We decided to use Mei's Pytorch code [29] which is fully modifiable. It uses the connectionist temporal classification (CTC) loss as the algorithm's cost function and gives three options of optimizer to use, which we will discuss below. In addition, it normalizes pixel values of the images in the range [0, 1] before having them pass through the CRNN model. This dataset normalization is important because it improves the numerical stability and can, also, reduce the training time. Besides, Mei's code calculates the accuracy of correct word prediction, which is the accuracy we take into consideration for all the training optimizations we perform below.

Since our initial thought was to make a text recognition system usable in Greece, we decided to train the CRNN in order to recognize text from Greek signboards, that is, scene text with Greek and English letters, as well as numbers. Therefore, we set the alphabet to "0123456789abcdefghijklmnopqrstuvwxyzαβγδεζηθικλμνξοπρστυφχψω". Apart from this alphabet, we, also, tried training the CRNN with a different multilingual dataset alphabet that contains characters from four different Latin languages, namely English, French, German and Italian, in addition with numbers and symbols. This alphabet is as follows: "0123456789aáàâäɑbßcdeéèêɜœfghiìîjklmnoóòôöpqrstuúùûüvwxyzȝAÁÀÂÄÃ^BCÇDEÉ ÈÊËŒFGHIÌÎJKLMNOÒÔÖPQRSŠŚTUÙÜVWXYŸZΦ.,!?:;~_▁●▪··•§|@#&+- */^=%$€£²°<> 《》 “”()[]‘`'”".

3.2.1. Training with synthetic data

Before creating our datasets to train the CRNN for the desired alphabets, we had to ensure that Mei's code was, actually, achieving Shi's *et al*. described results. For this reason, we recreated Shi's *et al.* training with the synthetic dataset by Jaderberg *et al.* [30], which is the one used in the training of CRNN by Shi *et al.* to achieve their reported high accuracy. Some sample images from this dataset are shown in Figure 12. Figure 13 shows the accuracy we got during the training, which is 84.3%. We get this accuracy after the third epoch of training,

Figure 12. Sample images from the synthetic dataset.

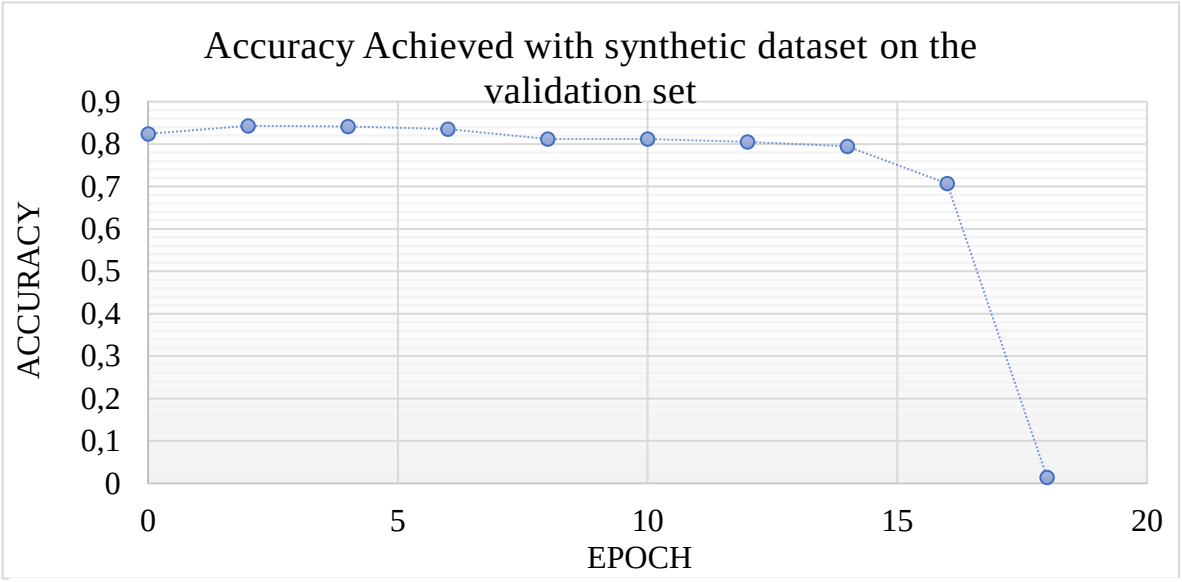


Figure 13. Accuracy achieved when recreating Shi's et al. training [20] with the synthetic dataset. The exact accuracy numbers are reported in Appendix A.

while the model seems to be overfitting in the next training epochs. This accuracy is very close to Shi's *et al.* results [20], which is 86.7%, hence we concluded that Mei's solution can be applied to the problem we focus on in this Thesis.

As a next step, we wanted to assess the size of the dataset we have to create, in order to train the NN and achieve an acceptable accuracy. We believe that an acceptable accuracy should be at least 0.5, so that the majority of the depicted words are recognized. Thereupon, we experimented on training the CRNN with images from Jaderberg's *et al.* synthetic dataset [30] on an alphabet containing lowercase English letters and digits. Specifically, we started with 500 samples in the training dataset and successively doubled the dataset size. The samples in the training and validation sets were distributed on the basis of 80/20 percent rule. From these experiments, the results of which can be found in Table 8 in Appendix B, we concluded that more than 8000 distinct images are needed in the training set to attain an accuracy of 0.5, as shown in Figure 14 As a consequence, when gathering the images for our trainings of the CRNN we aimed for at least 8000 samples in the dataset, if possible.

3.2.2. Character encoding

The encoding used in the code for reading the characters is utf-8. This fact indicates that, when Python is instructed to read a character from a string, it will read one byte from the string. However, in our alphabets we come across some letters and symbols, such as the Greek characters, that are encoded in more than one bytes with the utf-8 encoding.

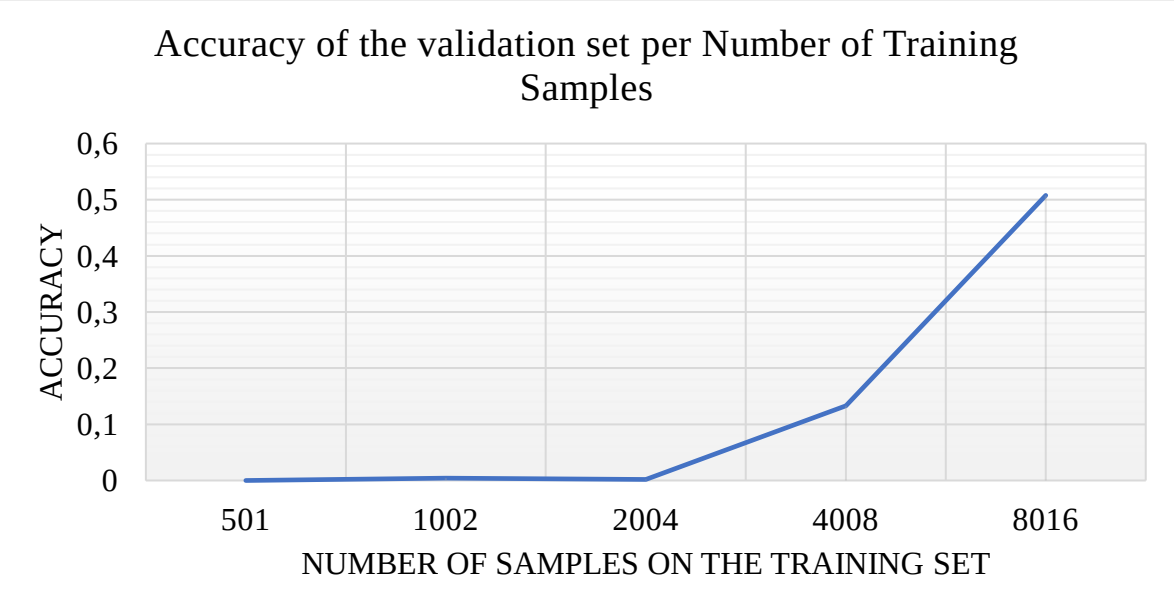


Figure 14. Graph of valid accuracy per number of samples used in training. To achieve an acceptable accuracy of 0.5 we need at least 8000 images in the training dataset. The data used for this graph are taken from Table 8.

Concretely, as mentioned in the beginning of Section 3.2, both the alphabets we use contain English letters and numbers, which occupy one place (one byte) in the alphabet table, while the rest of the characters occupied two or even three places. To solve this encoding problem, we firstly searched for a different encoding, where all the characters we are interested in occupy the same number of bytes in the table. Some of the encodings we studied were utf-16, utf-32 and ascii. However, we found out that, unfortunately, there is no encoding in which all the letters, numbers and symbols we use occupy the same number of places in the alphabet table.

For this reason, we had to create our own encoding so that all the characters in our alphabet occupied one place in the table. We chose to have all characters occupy one byte, because, as mentioned above, our Python code is encoded in utf-8, which means that Python is reading one byte for each character. In particular, we had to create a different code for each character's encoded byte. First, we observed the encodings of each character and kept intact those that already occupied one byte. Then, for the remaining characters that occupied more than one byte, we removed their first one or two bytes of the encoding – we only kept the least significant byte. If the remaining byte was unique among the other existing encodings, we used that as the encoded byte of the character. Otherwise, we would create a unique encoding of one-byte size for that character. With this one-off process we managed to have a translation map which assigns a single, one byte encoding for each character of the alphabet used.

After that, we wrote functions to automatically perform the specific encoding change for each character of the alphabet. Of course, since this is a made-up encoding, Python cannot properly display the characters when encoded like this, hence we, also, had to write functions to reverse this encoding alteration back to the original utf-8 encoding. This task was significantly easier, considering that every character now occupies only one byte, so we only had to replace each character's encoding with its original utf-8 encoding.

3.2.3. Dataset creation for CRNN

Concerning the case of scene text with Greek and English alphabets, we initially focused on finding a reusable dataset with scene text images from Greek cities. However, after a thorough search online we could not find any Greek scene text dataset, so we decided to create one. First, we had to take pictures that depicted Greek shop signs, with both Greek and English characters. In order to accomplish that, we captured 5 different videos, out of

which we would crop signs to include in our dataset. The videos have been captured with a smartphone camera attached on a car's front window, while moving around a city center. The video has been taken at a 1920x1080 resolution and a 30 frames per second framerate.

After acquiring the videos, we extracted their frames to get the required images. While examining the quality of the frames, we noticed that their content would significantly change every approximately 15 frames, thus we utilized only one frame per 15 (or, equivalently, 2 frames per second). Next, for each frame we have detected the areas containing text and cropped them to produce the dataset images. Last, we had to manually annotate the text depicted in each image. This way, we have generated a dataset with 517 samples, which we, then, split on the basis of 85/15 percent rule to generate the training and validation sets needed. Some sample images of this dataset are depicted in Figure 15.

In contrast, the creation of the dataset for the Latin languages training was less effortful. There are plenty of scene text datasets with our preferred languages online, so we just had to gather the samples for our dataset. We decided to use images from the ICDAR 2019 dataset [18]. Specifically, we used the images that captured text from the four different languages we included in our multilingual alphabet, as shown in Figure 16. The total number of images in our dataset is 46622 and we split them on the basis of the 80/20 percent rule to generate the training and validation sets.

3.2.4. Training with scene text images

With the generated datasets we started training the CRNN model for each alphabet case separately, once for the Greek and English and once for the Latin alphabet. In particular, in

Figure 15. Sample images from the Greek-English scene text dataset.

Figure 16. Sample images from the Latin scene text dataset.

the occasion of the Greek and English alphabet we trained the NN for more than 28 hours, however the results were very disappointing. The accuracy on the validation set never got higher than 0.05 and the test loss was over 100. On the other hand, in the case of the Latin alphabet we trained the CRNN for about seven and a half hours and the validation accuracy we got was 0.483. Although this result was much better than the one on the other alphabet, we, still, wanted to enhance it, too.

In order to achieve that, we decided to perform some augmentation on the data in order to create a better dataset for training, which we will elaborate on in the next Section. After that, we observed the two alphabets so that we could examine the possibility of reducing their sizes and eliminating the characters that did not occur frequently. In the Greek and English alphabet we did not perform any changes, since, as we will discuss in the next Section, the size of the dataset used was not enough for a proper training of the CRNN.

Regarding the Latin alphabet, after finding a better dataset for training, we gradually reduced the size of the dataset removing 55 characters from the alphabet and retrained the CRNN to check if the accuracy was increasing. Those characters are the ones with the lowest frequency of occurrence in the dataset, that is mostly some rare symbols. The new alphabet is now: "0123456789aáàâäbcdeéèêfghiiîjklmnoóòôöpqrstuúùûüvwxyzABCDEFGHIJKLMNOPQRSTUWXYZ.,!?:_@#&+-/%€()'". Indeed, with this alphabet shrinking, the validation accuracy we achieved was higher than the one in the full alphabet by 0.052, as seen in Figure 17, hence we kept the smaller alphabet in our afterward training attempts.

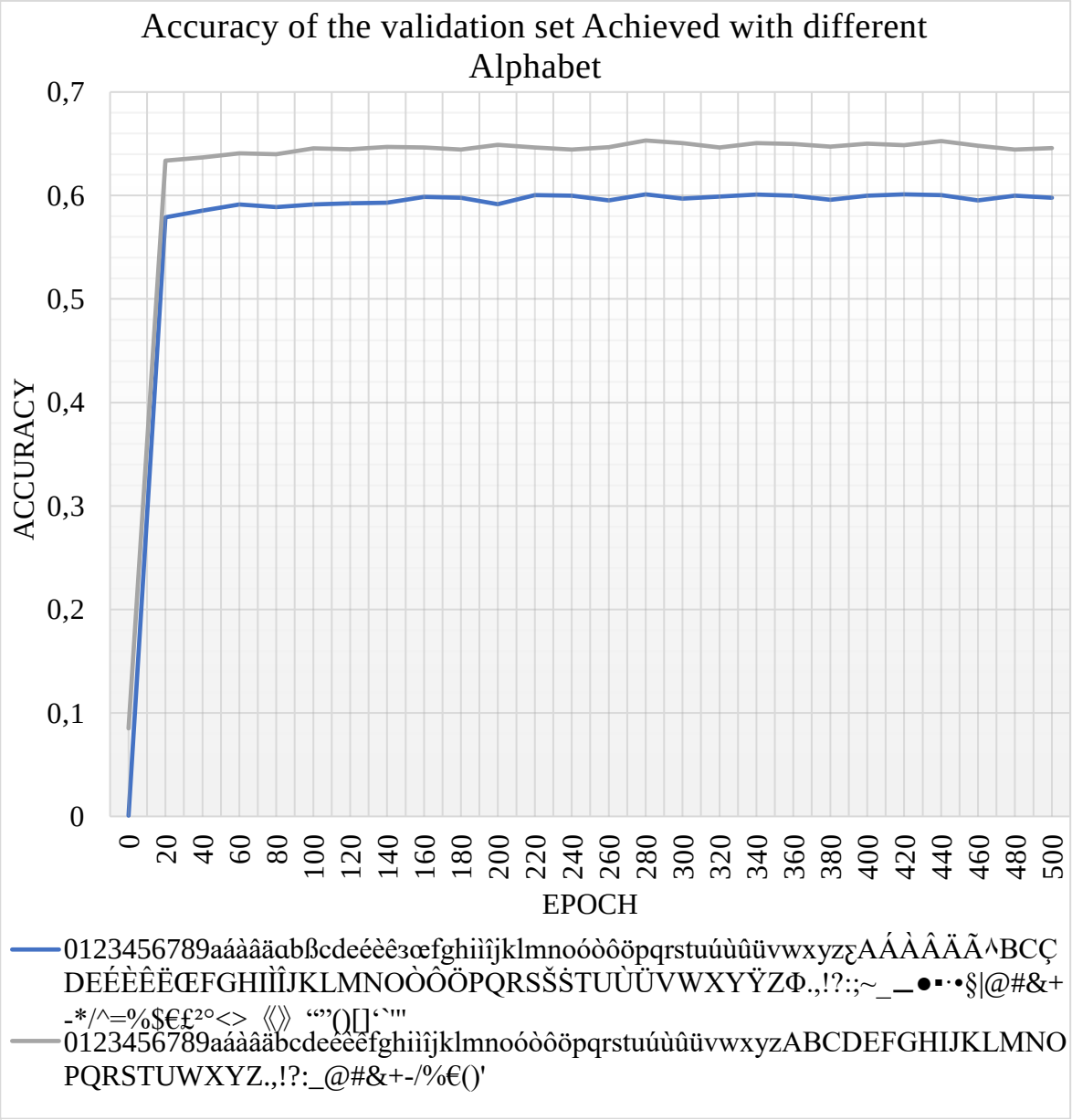


Figure 17. Accuracy achieved by CRNN on the validation dataset per training epoch, when trained with different alphabets.

3.2.5. Dataset augmentation

As reported earlier, for both training alphabet cases, we augmented the datasets so as to expand their size , in order to improve the model's ability to generalize. In the case of the Greek and English alphabet our goal was to increase the number of samples in the dataset, since having 517 images was not enough for a descent training. On that account, we rotated the images twice – once at 15 degrees and another time at minus 15 degrees - and we, also, added HSV noise with dulling 6, hue 11, saturation 11,9% and a luminosity value of 21,6% to the images. These augmentations can be seen in Figure 18.

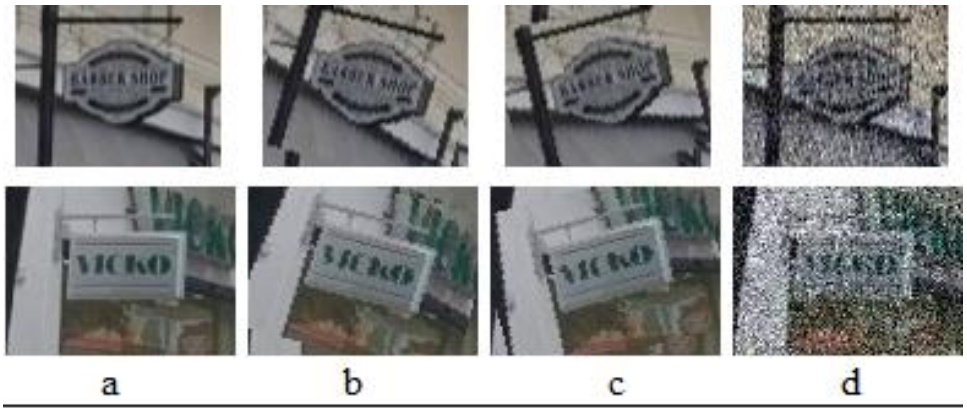


Figure 18. Samples from the augmentations on the Greek-English scene text dataset. a. Original image. b. 15 degrees rotation. c. -15 degrees rotation. d. HSV noise.

It is evident that the images after the augmentation process (Figure 18) have degraded. Specifically, the rotations pixelize them and the noise corrupts them in a great degree, so there can only be limited improvement in the training. This corruption happens because the original images do not have a very good quality, since they are taken from video frames, hence we could not fix their quality. In addition, we had these augmentations on the Greek scene text dataset in order to expand it into having at least 8000 samples, as we detected in Section 3.2.1 that are needed for an acceptable accuracy. However, after a couple of augmentations we did not observe any improvement on the achieved validation accuracy, so we stopped the augmentations at the point we only had 2068 images in the dataset. This amount of images in the dataset, still, seemed very few in comparison with the 8000 ones needed, hence we had to get more annotated images on the dataset. Nonetheless, since we wanted to spend the working time of this Thesis on understanding the training procedure of the NN, it would not be a good investment of time to proceed in manually producing a large enough Greek scene text dataset to achieve our goal.

For the Latin training, we, also, tried both applying rotations and HSV noise to the images. In particular, seeing that in this case we were aiming in rising the validation accuracy, we conducted experimental trainings of the CRNN with different augmentations on the dataset. Apart from having the original images from ICDAR 2019 [18] in our dataset, we have added up to three random rotations in the range [-15, 15] degrees and HSV noise (with dulling 5, hue 59, saturation 68.3% and luminosity value 43%) in different combinations to generate an effective dataset to train the CRNN with. The selection of the augmentations to be applied is made by checking the result they give on the quality of the images and the outcome can be seen in Figure 19. In this Figure we can see that the images are not as corrupted as in the case of the Greek scene text ones, which occurs due to the fact that these original images are

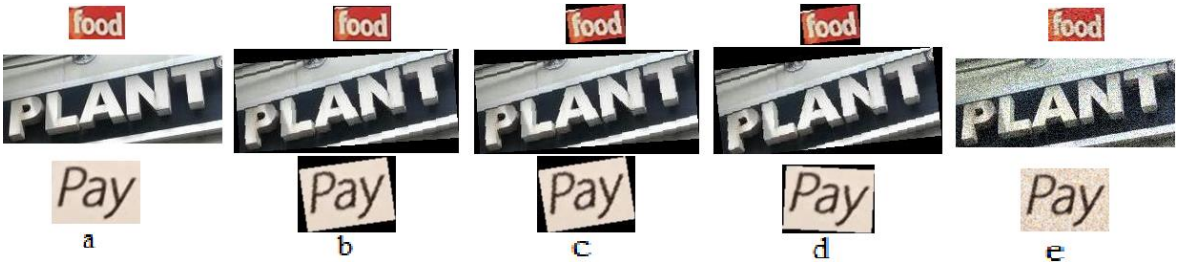


Figure 19. Samples from the augmentations on the Latin scene text dataset. a. Original image. b-d. Random rotation in [-15, 15] degrees. e. HSV noise.

better in quality. The combinations of the augmentations we applied are shown in Figure 20. Also, in this Figure, it is evident that the dataset that contained the original images, two copies of them randomly rotated, and another copy with HSV noise achieves the best accuracy on the validation dataset. Thus, we selected this dataset as the most effective dataset for training the CRNN.

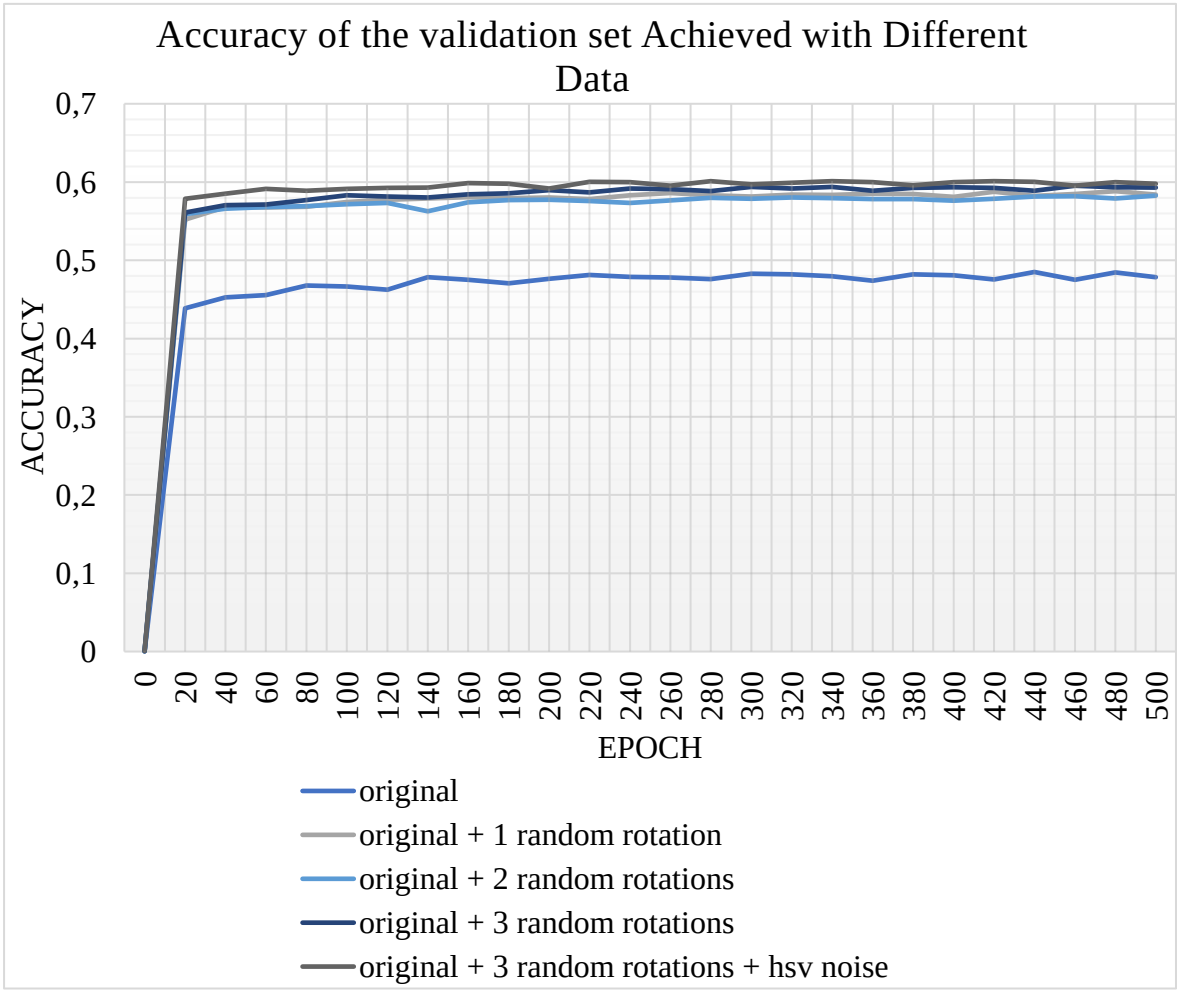


Figure 20. Accuracy achieved by CRNN on the validation dataset per training epoch, when trained with different datasets.

3.2.6. Hyperparameter tuning

The last step after finalizing all the training parameters, was to analyze the effect of the values of hyperparameters. The hyperparameters of the CRNN are the optimizer type, the batch size and the size of the LSTM hidden size. The default values for these hyperparameters are the RMSprop optimizer, 500 samples per batch and 256 layers in the LSTM hidden state. We did not try to experimentally optimize these values for the Greek and English training, since the problem with that training lies in the fact that we had very few training samples in comparison to the ones that were actually needed.

Consequently. we experimented with the hyperparameter tuning on the Latin alphabet training. In particular, we, first, focused on the optimizer. The default optimizer is the Root Mean Square Propagation (RMSprop), which is an optimizer very similar to the gradient descent algorithm with momentum and the user has to define the learning rate to apply. Another option for which the learning rate has to be manually defined is the Adam optimizer, which is a combination of two extensions of the stochastic gradient descent, the adaptive gradient algorithm (Adagrad) and the RMSProp. However, the optimizer we selected for our task is the Adadelta, which is an extension of the Adagrad optimizer that reduces the monotonically decreasing learning rate. When training the CRNN with different optimizers, we found out that the training converges faster whit Adadelta. Hence, for all the trainings following, we use the Adadelta optimizer.

Afterwards, we continued with training the CRNN with different batch sizes. Starting from 500 batches we binary searched to identify the optimal number of batches. As shown in Figure 21, the training performance is not very sensitive on the batch size. Still, we experimentally concluded that 1000 is the best size for the batches. Finally, we experimented with different sizes for the LSTM hidden state. Similarly to batch sizes, we binary searched for the optimal number of layers for the CRNN's LSTM hidden state. That number can be deduced from Figure 22 and is 256. Appendix E summarizes the exact validation accuracies achieved by each different training.

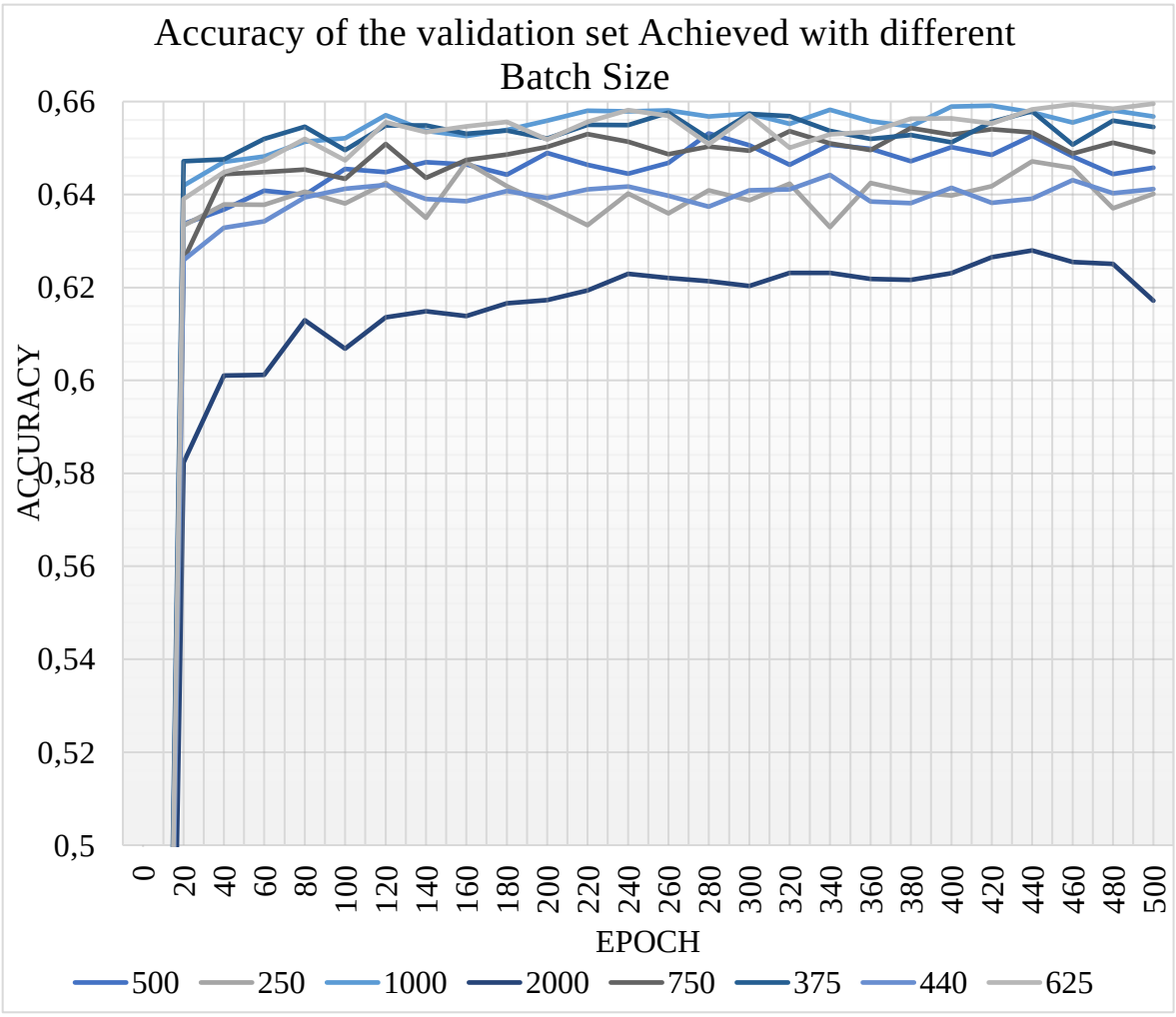


Figure 21. Accuracy achieved by CRNN on the validation dataset per training epoch, when trained with different batch sizes.

3.3. Integration of text detection and recognition

In order to make a complete evaluation of text detection and recognition on a multilingual character set, we had to integrate the above assessed NNs into one system. In particular, the image first goes through the EAST text detector, which will determine the text's bounding boxes and, then, each of the bounding boxes is cropped out of the original image to be fed to the CRNN. Since the CRNN takes as input a rectangular image, text part needs to be cropped out as rectangular. For this reason, we use the rectangular bounding box on every case of text orientation and we crop 10 pixels outside the bounding box to increase the probability of including the whole word in the cropped image.

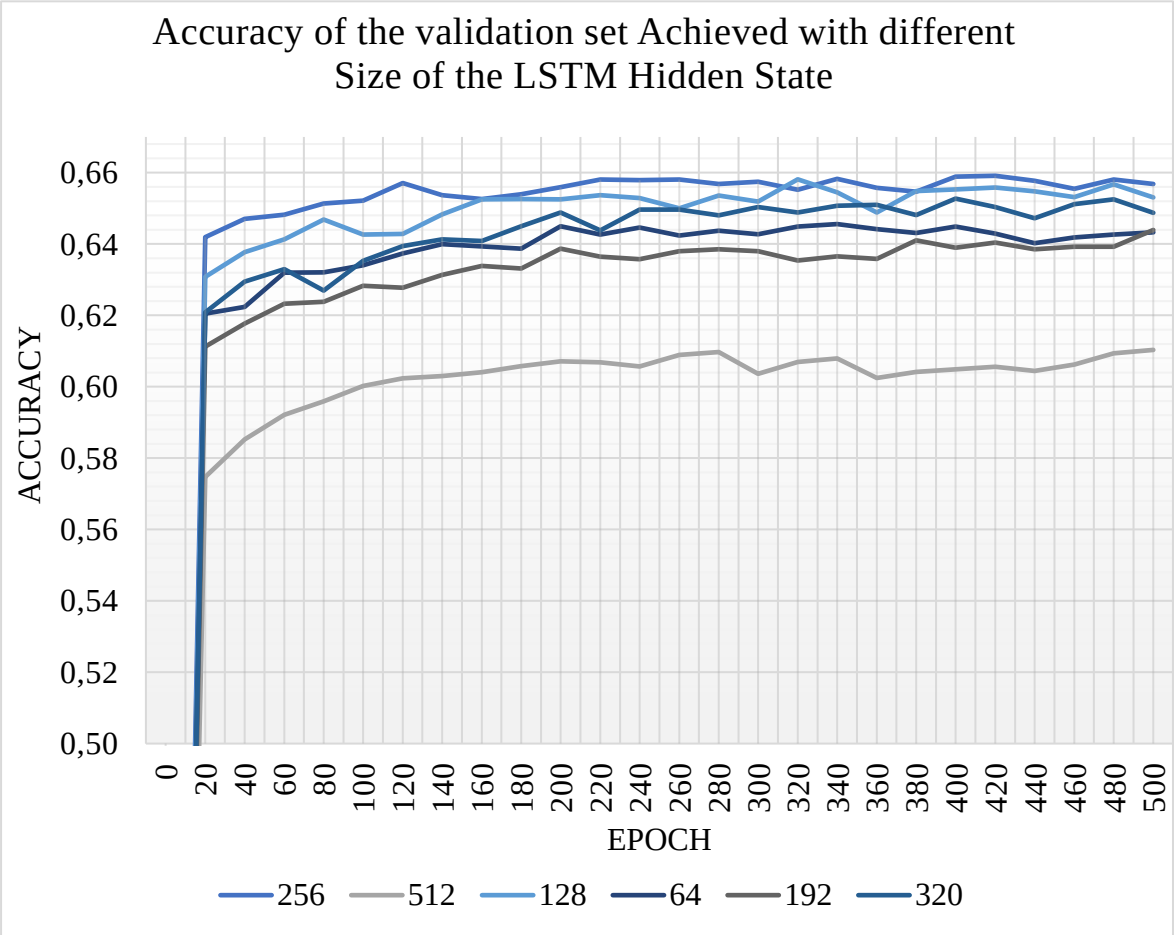


Figure 22. Accuracy achieved by CRNN on the validation dataset per training epoch, when trained with different sizes of the LSTM hidden state.

The cropped image is converted to grayscale, and then it is resized to 32 x 100 pixels so that it fits the dimensions of the CRNN input. In addition, before the image is given to CRNN, it is normalized to [0, 1], as demonstrated in Figure 23. When the results of the CRNN are ready, they are written on the original image along with the predicted bounding boxes by the

Figure 23. The cropped image goes through three stages before it is given as input to the CRNN. a.. Grayscale. b. Resize to 32 x 100 pixels. c. Normalization to [0, 1].

EAST text detector. The final results we get are shown briefly in Figure 4 and in detail later, in Section 4.

3.3.1 Performance optimization

Our final goal was to optimize the performance of the inference code. Initially, we used the 2 NNs sequentially. However, in order to achieve better performance for our system, we decided to use them in parallel, specifically with pipelining. The main thread performs the text detection on a frame, and a second thread is created to perform the text recognition for the text parts of each image. Next, we decided to have each text part being recognized in parallel, so for each bounding box found in a frame, a separate thread is created to have it run through the CRNN. This multithreading on the CRNN did not result in a better performance of our system, due to the fact that we had too many CRNN threads spawned for the same input image, thus we removed it.

In order to annotate each frame with the results we have created one last thread (per frame) which waits until the text recognition thread is finished and, then, it saves the final result in an image file. After performing the text detection on one frame, the main thread moves on to the next frame without waiting for the CRNN threads to finish their work, therefore the text detection and text recognition stages overlap. It is also important to mention that the text recognition stage takes 1.15 times more time to finish its work on the image than the text detection stage. This pipeline is illustrated in Figure 24 for a frame where three bounding boxes were detected.

As a last step, we enabled GPU execution. First, we sent our CRNN model to the server's CUDA-enabled GPU device. We do this by calling the Pytorch's .cuda() method on the model and turning the input images into cudaTensors. Next, we tried moving the text detection part to the GPU. Specifically, we made use of OpenCL-enabled OpenCV to automatically transfer the heavy matrix operations to the accelerator device. Along these lines, OpenCL detects the accelerator device on the server and transforms the executable code in order for it to run on the detected GPU. However, enabling OpenCL did not affect the system's speed, since the text detection part executed at the same speed as in CPU. Therefore, we ended up not using OpenCL. In the next Section we evaluate the performance of the inference.

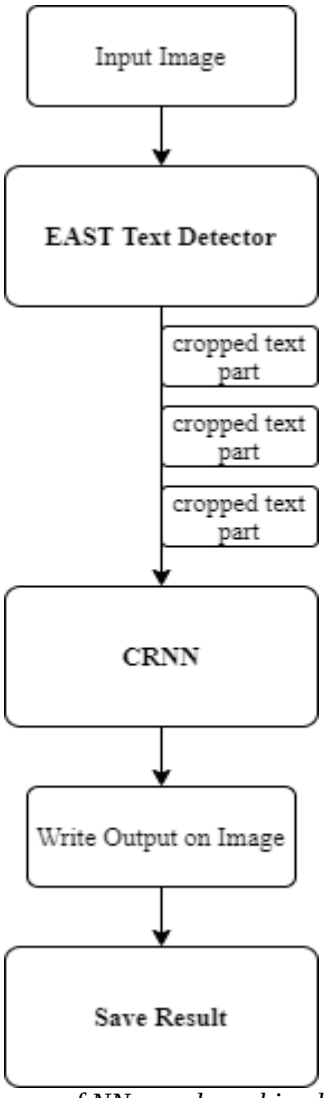


Figure 24. Diagram of NNs used combined in parallel.

4. EVALUATION

4.1. Accuracy

In this Section we will evaluate the scene text detection and recognition system for scene text in English, French, German and Italian. We did not proceed on producing results for the Greek and English alphabet, because, as explained in Section 3.2, the limited size of the dataset we had thwarted the proper training of the CRNN for this case. For the evaluation we used images from the ICDAR 2019 dataset [18], different from the ones we used for training and validation. Figure 25 shows the results for some indicative scenes. We discuss the results in detail below.

Figure 25. Final output images of the system.

4.1.1. Text detection

As far as text detection is concerned, it is apparent in the images of Figure 25 that the bounding boxes cover the text parts of the image in a quite satisfactory degree. In particular, we observe that in most of the images almost all the text is identified. More specifically, the text detector identifies the text parts of the image with 0.814 recall. As expected, in many cases the text detector falsely identifies parts of the image as text. Such cases are depicted in Figure 26, however we do not consider this as a disadvantage, since our goal, which is the detection of the actual text, is achieved. This precision of the text detector is 0.853.

Table 2 summarizes the precision, recall and F-score we accomplish with the text detector. In these measurements we can observe that we achieve an F-score of 0.8315 with the EAST text detector on the ICDAR 2019 multilingual scene text images, which is even better than the reported one by Zhou *et al.* That F-score is 0.8072 and it is measured on the images from

Figure 26. Cases where the text detector falsely identifies parts of the image as text.

the ICDAR 2015 incidental scene text challenge [27], which are different than the ones from the ICDAR 2019 dataset that we use. Hence, we get an improvement of the F-score.

Another metric we evaluated, which is, also, included in Table 2, is how often the true positive bounding boxes cover the whole word depicted. As described in Section 3.1.1, although the diagonal orientation of the text makes the full coverage of the text a challenging process, we are able to have whole words covered by the bounding box at a percentage of 0.619. Particularly in Figure 27 it becomes clear how, in many cases, the bounding box cannot fully cover the detected text. Undoubtedly, this fact affects the next stage of our Thesis, the word recognition, since words that are not fully covered by the bounding box may have some letters cropped in half, which can make it challenging for the CRNN to recognize them.

4.1.2. Text recognition

Concerning the text recognition stage, the final accuracy of correct word prediction we achieved after all the training optimizations elaborated in Section 3.2 is 0.6582. At this point, it is important to clarify that we measure the accuracy as in the number of whole words recognized to the number of all the words to be recognized and not as in the number of letters identified correctly to the number of all the letters to be identified, as mentioned in Section 3.2. Also, we do not take into consideration the falsely identified bounding boxes, since they

	Recall	**Precision**	**F-score**
Text detection	0.814	0.853	0.8315
Total text coverage	0.619	0.528	0.5699

Table 2. Recall, precision and F-score measurements of the EAST text detector on the ICDAR 2019 images. We included both the detection of text and the whole coverage of the text depicted.

Figure 27. Cases where the text detector fails to cover the whole word with the bounding box.

do not contain any text and the CRNN is not designed to recognize this kind of errant input. However, at the granularity of individual characters, the accuracy the final trained model achieves is high, at 0.846.

presents the expected accuracy of random letter recognition, the final accuracy we achieve and the percentage of accuracy improvement over random, for both cases of the Latin alphabet we discuss, full and condensed. In the case of the condensed alphabet, for which we got the final results shown in Figure 25, we were able get an improvement in accuracy of 81.22x This raise is even higher for the full alphabet, 122.24x, since the possibility of randomly predicting a word correctly is significantly lower. This is a strong indication of the effectiveness of the CRNN.

In particular, it is evident in Table 4 that in most cases of wrong word identification the error is in the identification of only one or two letters of the whole word.. As a consequence, the conclusion of our evaluation of the CRNN is that it is a quite reliable NN for multilingual scene text recognition.

In comparison with Shi's *et al.* [20] achieved accuracy on the lower case English letters and numbers, that is reported in the beginning of Section 3, we get quite similar accuracy on our Latin alphabet, although a bit inferior. That is because of various reasons. Firstly, we train the model with about 233 thousand dataset samples, which are significantly fewer than the nine million samples Shi *et al.* use for their training. Secondly, the alphabet which we want the CRNN to make predictions for is almost 1.5 times bigger than the lower-case English alphabet used by Shi *et al.*, increasing the number of parameters in the CRNN by a large margin. Additionally, the type of images used in the training dataset affects the final results substantially. In scene text images it is more difficult to identify words than in synthetically created ones, because of all the obstructions due to lighting differences, the image stabilization and resolution, that can be present in the scene text images.

Alpahabet	Random Accuracy	Achieved Accuracy	Accuracy Improvement
Full	0.0066225	0.8095238	122.2383994x
Condensed	0.0104166	0.8459958	81.21611658x

Table 3. Text recognition random accuracy, achieved accuracy and its raise for both full and condensed alphabet.

Prediction	Ground Truth
Seitensteife	Seitenstreife
Ker	Ker
te	te
FONDAZION	FONDAZION
verkau	verkau
Polic	Polic
Finanzcente	Finanzcente
RA17-1	R417-1
FU	fü
Hinwe	Hinwei
testat	testat
Unio	Unio
destination	destination
WEL	wei
these	thèse
PANTOTHERARI	PHYTOTHERAPI
Emeiberganodeg.i	E-mail:bergamo@agsg.i
EMPT	EMPT
polui	potizi
see	gate
Inaugurazione	Inaugurazione
OU	OU
Pierr	Pierr
ancienn	ancienn
Rosevill	Rosevill
l	l
de	de
Le	Lè

Table 4. CRNN results on images from the ICDAR 2019 dataset. The left column shows CRNN's prediction and the right one shows the ground truth text depicted on the image

4.2. Performance

As explained in Section 3.3.1, after integrating the two NNs into one system, we concentrated on optimizing the code's performance. To do that, we went through three different optimization stages, that are described in Table 6. The first one is when the NNs are combined sequentially and the second when they are combined in a pipeline as shown in Figure 24. Both of these stages include executing the code on the CPU and the optimization is achieved by the pipeline parallelism. The last stage makes use of the server's GPU

accelerator, detailed in Table 1, for the execution of the most time consuming of the two NNs, the CRNN.

Table 5 shows the time (in seconds) that takes the system to infer 100 images, but also the average inference time per image. In particular, in stage one, in which there are no optimizations, the system takes 88.356 seconds to process 100 images, that is 0.884 seconds per image on average. When we added the first optimization, which is the 2-stage pipeline parallelism, the total time dropped to 42.024 seconds which means that the code became about 2 times faster with the use of two threads, aswe would expect it to be. Transferring the CRNN model to the GPU made execution 1.31 times faster than the previous stage; processing a hundred images takes 32.048 seconds at this point. In this final stage, where we have made use of the system's GPU, the processing takes 0.32 seconds per image on average.

It is clear from these timing data that implementing all these optimizations on the inference code significantly reduced the image processing time. Specifically, the code is executing 2.76 times faster after all the optimizations. In addition, this system could be used as part of an application that has its heaviest calculations transferred to a server, since it can produce very fast results, at slightly more than 3 frames per second.

Optimization stage	Description
1	Sequential integration of the NNs
2	Pipeline parallelism
3	Pipeline parallelism and execution of CRNN in GPU

Table 6. Description of the optimization stages on the inference code.

Optimization stage	Time per 100 images	Time per image
1	88.356 sec	0.884 sec
2	42.024 sec	0.42 sec
3	32.048 sec	0.32 sec

Table 5. Time per optimization stage, that the system takes to process images. Optimization stages are described in Table 1.

5. RELATED WORK

Many researchers have approached the text detection and recognition problem, either each of its parts separately, or the problem as a whole. In this Section we discuss closely related work on text detection and text recognition. We also discuss an end-to-end approach for detecting text from images and recognizing the words in it.

5.1. Text detection

There are two primary techniques used to perform text detection on images. The first one is image based and it entails the image being partitioned into multiple segments with similar features. These image segments are then classified as text or non-text with the use of methods coming from the field of machine learning. One of these methods are SVM engines. Shin *et al.* [7] have discerned the text detection problem as textured based, hence they elucidate it as a pattern classification problem. Since the SVMs are a pattern classification mechanism, the authors propose a system that scans the image with a small window, and for each position of the window it classifies the central pixel as text or non-text. This analysis is enabled by using predefined features of the input's grayscale pixels, which are, then, dot multiplied with the support vectors to produce the final result.

The weakness of this approach to text detection is the fact that the initial features have to be manually selected. To avoid this hindrance, Delakis and Garcia have introduced CNNs in their study [8], where not only the features used, but also the localization of the text is determined in a one-step process. Their CNN consists of one layer where the image is decomposed to its RGB components, followed by four layers of filters applied to these components, where they, also, have their features fused together to assemble the attributes evaluated during classification. The last two layers of the CNN perform a sigmoid classification to generate the final result. Both image-based techniques reviewed above have in common the fact that in order to detect text of different sizes, the image needs to be fed to the system multiple times in different scales and the results of each pass are combined to get the final detected bounding boxes of the text.

The second text detection method that is commonly used in literature is frequency based. It is based on the principle that document images can be perceived as two-dimensional signals, to which we can apply signal processing methods. Yeotikar *et al.* [9] apply a DFT in order to extract text from images. There are many applications of the Fourier transform (FT) that

can be used in image processing, such as low pass or median filters to remove noise, correlation to locate different image features and convolution, since the reverse product of the FTs of two matrices gives the result of the matrices' convolution. In their research [9], Yeotikar *et al.* have demonstrated five different implementations of these methods applied with the purpose of locating text in document images. The first two methods are a two-dimensional FT and a two-dimensional DFT. Then they tested the spatial domain filters, which act like convolutions of the image matrix with a small window. Lastly, they experimented on how the FT can be used for image enhancement and edge reinforcement.

Apart from the FT, another method from signal theory that is used for text detection is the Discrete Wavelet Transform (DWT). Liang and Chen [10] have worked with the Haar DWT to perform text extraction from images. More specifically, they, firstly, perform a two-dimensional DWT, which is able to detect edges existing in the image in three different directions, horizontal, vertical and diagonal. After that, they apply the Haar DWT, which enhances the edges in the image and, then, they combine all the types of edges with a logical AND operation so as the text regions to be revealed. Distinguishing of the text from the background, in this case, is performed with dynamic thresholding. As a result, the frequency-based approaches have the text become more distinct from its background, thus it can be easily located in the image.

5.2. Text recognition

Concerning the word recognition on images, we first need to clarify that the words to be recognized are already detected in the image and the text's bounding box is defined. One of the dominant approaches available in literature is bottom-up. This kind of method includes the image going through an OCR engine. According to Kumar *et al.* [11], segmentation is the practice when an image is binarized and then it is given as input to an OCR engine. The binarization process is especially important when the image is a scene text image, since such images often lack quality or they might be distorted. For this reason Kumar *et al.* have proposed two different binarization algorithms [11] [12] for scene text image segmentation.

Their first binarization algorithm is called midline analysis and propagation of segmentation (MAPS) [11] and it consists of five steps. In their research they find that the middle row of the image is affected less by the low quality problem, so, in MAPS, this part of the image is segmented with the use of two different methods, the niblack and the min-max methods. For the rest parts of the image the algorithm propagates the labels found in the middle row.

Kumar's *et al.* second proposed binarization algorithm is the nonlinear enhancement and selection of plane (NESP) [12], which is quite similar to MAPS. The only difference between these two binarization algorithms is the separation of the image in horizontal zones. In contrast with MAPS, in NESP a nonlinear enhancement of the image's RGB components is performed and the RGB component in which the text which is most separable from its background is selected to complete the binarization process of the algorithm.

After the segmentation process, we get the binarized image, which we can, now, give as input to any OCR engine in order to recognize the words in it. An example OCR engine is the Tesseract [31]. Patel *et al.* have studied the way Tesseract achieves text recognition and describeit in [13]. When Tesseract is given a binarized image, it performs a connected components analysis from which character outlines are determined. These character outlines are turned into blobs, which makes it easy to individuate the different text lines and words. A two-phase word recognition follows that gives the final output of the OCR.

Another approach for text recognition is top-down. In this type of approaches, compared to bottom-up, the image does not need to be segmented. Instead, these methods use a set of words coming from a dictionary and identify the word that the text in the image matches best. One way of defining this match of images with the words they depict is through a prior computed high order statistical language model, as explained by Mishra *et al.* in [14]. They have created a higher order conditional random field (CRF) of all the possible combinations of characters and digits that can be found in a lexicon. When it comes to the word recognition, they use this CRF to determine the frequency of the possible words, that are depicted in the image, inside the dictionary. As a consequence, the word with the highest frequency is selected as output of the text recognition system.

5.3. End-to-end approach

Most approaches to text detection and recognition use a two-phase structure, one phase for the text detection and another for the word recognition, however there are some researches [15] [32] that make use of DNNs to generate end-to-end text detection and recognition engines. For instance, Bartz *et al.* [15] have proposed a single DNN that divides the work into subtasks with the use of a localization network. Their network consists of two parts, the localization network and a text recognition stage. The localization network is the one that identifies the text regions in the image and with the application of a grid generator the bounding boxes are calculated and the text regions are extracted to continue on to the

recognition stage. The recognition stage is comprised by a CNN with ResNet architecture and outputs the labels of the detected text in the image. This kind of systems are end-to-end trainable and, usually, they make use of one single multitask DNN.

6. CONCLUSION

In this Thesis we evaluated the performance of two different NNs, the EAST text detector [19] and the CRNN [20], on multilingual scene text detection and recognition respectively. We used a serialized model of the EAST text detector, which performs fairly accurately on detecting text in scene text images, whereas it falls short in locating the edges of the detected text. As a result, although the EAST text detector can identify the text, when it comes to cropping the text parts out of the initial image, only about half of the produced cropped areas will include whole words.

For the text recognition stage, we, first, tried using a Greek and English alphabet, however, due to the lack of dataset, we shifted to a Latin alphabet that contains letters from English, French, Italian and German, along with digits and symbols. We trained the CRNN on images containing scene text of these languages and we, additionally, optimized this training by augmenting our dataset and finding the best values for the NN's hyperparameters. Finally, we achieved a 0.658 accuracy of correct word identification and an even higher accuracy for correct letter recognition, at 0.846.

After preparing each NN to perform at their best on multilingual scene text images, we integrated them into one system. This system takes as input an image and outputs the bounding boxes of the text depicted, as well as the identified words of the text. We had the two NNs run in a pipeline to reduce the execution time and, furthermore, we exploited a GPU to achieve better optimization performance. In this way we accomplished 2.38 times faster code execution and, most importantly, our system can produce results in 0.378 seconds per image.

Following the final evaluation of the NNs, we concluded that the most important factor for the success of a NN is the training it has gone through. In particular, the training dataset plays the most critical role on the quality of the NN's produced results, since the whole learning of the NN depends on it. On the current Thesis we dedicated most of the working time on training the CRNN, hence we understood many aspects of the training process, like the training dataset, the hyperparameters, and the expected output.

As a continuation of this Thesis, the CRNN can, also, be trained on the Greek and English alphabet. We have found that at least eight thousand samples are needed in the dataset, therefore we need to extend and properly augment the Greek scene text dataset of 517 annotated images we generated. In addition, there can be an improvement in text coverage

of the text detector, by finding a better shape for the bounding box. In the performance side, as reported the system executes fast in the server, however there should be a performance testing on a portable device and, possibly, a further optimization in order for it to be part of a locally executed application.

REFERENCES

[1] M. Walker-Ford, "SocialMediaToday," 26 February 2019. [Online]. Available: https://www.socialmediatoday.com/news/12-local-seo-stats-every-business-owner-and-marketer-should-know-in-2019-i/549079/. [Accessed 23 January 2020].

[2] B. Shaw, "SEO Expert Brad," [Online]. Available: https://seoexpertbrad.com/local-seo-stats/. [Accessed 23 January 2020].

[3] M. Rouse, "Computer Glossary, Computer Terms," April 2013. [Online]. Available: https://whatis.techtarget.com/definition/reverse-image-search. [Accessed 23 January 2020].

[4] A. Watson, "Angela's Watson The Cornerstone For Teachers," [Online]. Available: https://thecornerstoneforteachers.com/why-using-a-camera-to-take-notes-is-smart-not-lazy/. [Accessed 24 January 2020].

[5] Wikipedia, "Wikipedia," 4 December 2019. [Online]. Available: https://en.wikipedia.org/wiki/Scene_text. [Accessed 9 January 2020].

[6] F. Rithcer, "Statista," 31 August 2017. [Online]. Available: https://www.statista.com/chart/10913/number-of-photos-taken-worldwide/. [Accessed 23 January 2020].

[7] C. S. Shin, K. I. Kim, M. H. Park and H. J. Kim, "Support vector machine-based text detection in digital video," in *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*, Sydney, NSW, Australia, 2000.

[8] M. Delakis and C. Garcia, "text Detection with Convolutional Neural Networks," in *Proceedings of the Third International Conference on Computer Vision Theory and Applications*, Funchal, Madeira, Portugal, 2008.

[9] V. K. Yeotikar, M. T. Wanjari and M. P. Dhore, "Text Extraction from Document Images Using Fourier," *International Journal of Computer & Mathematical Sciences*, vol. 3, no. 9, pp. 89-95, November 2014.

[10] C. W. Liang and P. Y. Chen, "DWT based text localization," *International Journal of Applied Science and Engineering,* vol. 2, no. 1, pp. 105-116, 1 January 2004.

[11] C. Bartz, H. Yang and C. Meinel, "STN-OCR: A single Neural Network for Text Detection and Text Recognition," *ArXiv,* vol. abs/1707.08831, 27 July 2017.

[12] Missing Link, "Missing Link," [Online]. Available: https://missinglink.ai/guides/computer-vision/neural-networks-image-recognition-methods-best-practices-applications/. [Accessed 11 January 2020].

[13] J. E. Stone, D. Gohara and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering,* vol. 12, no. 3, pp. 66-73, 10 May 2010.

[14] N. Nayef, Y. Patel, M. Busta, P. N. Chowdhury, D. Karatzas, W. Khlif, J. Matas, U. Pal, J.-C. Burie, C.-l. Liu and J.-M. Ogier, "Dataset of ICDAR 2019 Robust Reading Challenge on Multi-lingual scene text detection and recognition," in *2019 International Conference on in Document Analysis and Recognition*, Sydney, Australia, Australia, 2019.

[15] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He and J. Liang, "EAST: An Efficient and Accurate Scene Text Detector," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017.

[16] B. Shi, X. Bai and C. Yao, "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 11, pp. 2298-2304, 29 December 2016.

[17] OpenCV, "OpenCV," [Online]. Available: https://opencv.org/about/. [Accessed 25 January 2020].

[18] Wikipedia, "Wikipedia," 30 December 2019. [Online]. Available: https://en.wikipedia.org/wiki/PyTorch. [Accessed 25 January 2020].

[19] D. Sayantini, "Edureka!," 14 May 2020. [Online]. Available: https://www.edureka.co/blog/keras-vs-tensorflow-vs-

pytorch/?fbclid=IwAR2R_c28I7AhAjfguIxzoskz8y8ZcaDOeeEf8evkLka7sPCwKOSFKqrxCXE. [Accessed 14 May 2020].

[20] A. Rohilla, "Towards Data Science," 11 April 2020. [Online]. Available: https://towardsdatascience.com/comparative-case-study-of-ml-systems-tensorflow-vs-pytorch-a554dce5f585. [Accessed 14 July 2020].

[21] Wikipedia, "Wikipedia," 29 December 2019. [Online]. Available: https://en.wikipedia.org/wiki/OpenCL. [Accessed 25 January 2020].

[22] Wikipedia, "Wikipedia," 23 January 2020. [Online]. Available: https://en.wikipedia.org/wiki/CUDA. [Accessed 25 January 2020].

[23] Create Pro, "Create Pro," 28 November 2017. [Online]. Available: https://create.pro/opencl-vs-cuda/. [Accessed 14 July 2020].

[24] D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. Ghosh, A. Bagdanov, M. Iwamura, J. Matas, L. Neumann, V. R. Chandrasekhar, S. Lu, F. Shafait, S. Uchida and E. Valveny, "ICDAR 2015 Competition on Robust Reading.," in *Proceedings of 13th International Conference on in Document Analysis and Recognition*, 2015.

[25] A. Rosebrock, "PyImageSearch," 20 August 2018. [Online]. Available: https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/. [Accessed 26 September 2019].

[26] A. Bissacco, M. Cummins, Y. Netzer and H. Neven, "PhotoOCR: Reading Text in Uncontrolled Conditions," in *IEEE International Conference on Computer Vision*, Sydney, Australia, 2013.

[27] M. Jaderberg, K. Simoyan, A. Vedaldi and A. Zisserman, "Deep Structured Output Learning for Unconstrained Text Recognition," in *3rd International Conference on Learning Representations*, San Diego, 2015.

[28] M. Jaderberg, K. Simoyan, A. Vedaldi and A. Zisserman, "Reading Text in the Wild with Convolutional Neural Networks," *International Journal of Computer Vision*, pp. 1-20, 7 May 2015.

[29] J. Mei, "Github," 7 September 2018. [Online]. Available: https://github.com/meijieru/crnn.pytorch. [Accessed 11 November 2019].

[30] M. Jaderberg, K. Simonyan, A. Vedaldi and A. Zisserman, "Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition," *International Journal of Computer Vision,* no. 116, pp. 1-20, January 2016.

[31] D. Kumar, M. N. Anil Prasad and A. G. Ramakrishnan, "MAPS: midline analysis and propagation of segmentation," in *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, 2012.

[32] D. Kumar, M. N. Anil Prasad and A. G. Ramakrishnan, "NESP: Nonlinear enhancement and selection of plane for optimal segmentation and recognition of scene word images," in *Proceedings of SPIE - The International Society for Optical Engineering*, San Francisco, CA, USA, 2013.

[33] Smith, Ray; Hewlett-Pachard;, *Tesseract,* 2017.

[34] C. Patel, A. Patel and D. Patel, "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study," *International Journal of Computer Applications*, vol. 55, no. 10, pp. 50-56, 20 October 2012.

[35] A. Mishra, K. Alahari and C. Jawahar, "Scene Text Recognition using Higher Order Language Priors," in *Proceedings of the British Machine Vision Conference*, 2012.

[36] R. Smith, C. Gu, D.-S. Lee, H. Hu, R. Unnikrishnan, J. Ibarz, S. Arnoud and S. Lin, "End-to-End Interpretation of the French Street Name Signs Dataset," *Computer Vision - ECCV 2016 Workshops,* vol. 9913, pp. 411-426, 2016.

[37] C. C. Chatterjee, "Towards Data Science," 31 July 2019. [Online]. Available: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add. [Accessed 2020 January 2020].

[38] NVIDIA, "Artificial Intelligence Computing Leadership from NVIDIA," [Online]. Available: https://www.nvidia.com/en-gb/data-center/tesla-k80/?fbclid=IwAR3uG0KvAVdfA4XLIwrNvDNedRN6HeMmRstF92Ca9cK7MhEVlwv_1-LyXxo. [Accessed 10 May 2020].

[39] J. A. Thompson and K. Schlachter, "An Introduction to the OpenCL Programming Model," 2012.

[40] AMD, *AMD OpenCL User Guide*, 2015.

APPENDICES

APPENDIX A

Results of training the CRNN with the synthetic dataset [30]

Epoch	Valid Accuracy
0	0.82376
2	0.84318
4	0.84136
6	0.83514
8	0.81192
10	0.8116
12	0.80534
14	0.7943
16	0.7068
18	0.01376

Table 7. Accuracy achieved per epoch of training when training the CRNN with the whole synthetic dataset.

APPENDIX B

Results of training the CRNN with a small part of the synthetic dataset [30]

Training Samples:	501	1002	2004	4008	8016
Epoch	**Valid Accuracy**				
0	0	0	0	0	0
20	0	0	0	0	0.07072
40	0	0	0	0.00347	0.42654
60	0	0	0	0.06482	0.47807
80	0	0	0	0.09375	0.47807
100	0	0	0	0.09375	0.47478
120	0	0	0	0.10764	0.47752
140	0	0	0	0.09144	0.48849
160	0	0	0	0.10648	0.49123
180	0	0	0	0.10417	0.49342
200	0	0	0	0.12616	0.50384
220	0	0	0	0.11458	0.49561
240	0	0	0	0.11111	0.50055
260	0	0	0	0.10301	0.49781
280	0	0	0	0.11921	0.49397
300	0	0	0	0.11343	0.49397
320	0	0	0	0.13079	0.48958
340	0	0	0	0.12037	0.50439
360	0	0	0	0.11111	0.48958
380	0	0	0	0.11111	0.49232
400	0	0	0	0.11806	0.49616
420	0	0	0	0.12037	0.49342
440	0	0.00391	0	0.13079	0.4841
460	0	0	0	0.1331	0.49671
480	0	0	0.00195	0.11921	0.48849
500	0	0	0	0.11574	0.50768
Max Accuracy:	**0**	**0.00391**	**0.00195**	**0.1331**	**0.50768**

Table 8. Accuracy achieved per epoch of training when training the CRNN with different number of samples from the synthetic dataset [30]. The alphabet used in this case is "0123456789abcdefghijklmopqrstuvwxyz".

APPENDIX C

Results of experimental training on reducing the size of the Latin alphabet

alphabet:	0123456789aáàâäɑbßcdeéèêɜœfghiıîjklmnoóòôöpqrstuúùûüvwxyzȝAÁÀÂÄÃ^BCÇDEÉÈÊËŒFGHIÌÎJKLMNOÒÔÖPQRSŠṠTUÙÜVWXYŸZΦ.,!?:;~_▁●▪·•§|@#&+-*/^=%$€£²°<> 《》 “”()[]‘`'"	0123456789aáàâäbcdeéèêfghiıîjklmnoóòôöpqrstuúùûüvwxyzABCDEFGHIJKLMNOPQRSTUWXYZ.,!?:_@#&+-/%€()'
data:	**original + 1 random rotation**	
batch size:	**500**	
size of lstm hidden state:	**256**	
epoch	**valid accuracy**	
0	0.000791209	0.085066667
20	0.578813187	0.633688889
40	0.585318681	0.636755556
60	0.591450549	0.640822222
80	0.588813187	0.639955556
100	0.591406593	0.645511111
120	0.592461538	0.644777778
140	0.593010989	0.646955556
160	0.598659341	0.646466667
180	0.597912088	0.644288889
200	0.591736264	0.648955556
220	0.60032967	0.646422222
240	0.59989011	0.644444444
260	0.595406593	0.6468
280	0.601120879	0.653133333
300	0.597054945	0.6506
320	0.598945055	0.6464
340	0.600989011	0.650755556
360	0.599912088	0.649822222
380	0.595978022	0.647133333
400	0.599758242	0.650177778
420	0.601120879	0.648533333
440	0.60043956	0.652666667
460	0.595384615	0.648155556
480	0.599692308	0644377778
500	0.597912088	0.645755556

Table 9. Accuracy achieved per epoch of training when training the CRNN with different alphabets.

APPENDIX D

Results of data augmentation experimental training on the Latin alphabet

alphabet:	0123456789aáàâäɑbßcdeéèêɜœfghiıîjklmnoóòôöpqrstuúùûüvwxyzƹAÁÀÂÄÃ^BCÇDEÉÈÊËŒFGHIÌÎJKLMNOÒÔÖPQRSŠŚTUÙÜVWXYŸZΦ.,!?:;~_⚊●▪·•§|@#&+-*/^=%$€£²°<> 《》 “”()[]‘`'"				
data:	**original**	**original + 1 random rotation**	**original + 2 random rotations**	**original + 3 random rotations**	**original + 3 random rotations + hsv noise**
batch size:	**500**				
size of lstm hidden state:	**256**				
epoch	**valid accuracy**				
0	0.00043243	0.00043243	0.00043636	0.00043836	0.00079121
20	0.43859459	0.55189189	0.55916364	0.5608761	0.57881319
40	0.45275676	0.56810811	0.56589091	0.57041096	0.58531868
60	0.45545946	0.56767568	0.56810909	0.57112329	0.59145055
80	0.46767568	0.56848649	0.56923636	0.57704110	0.58881319
100	0.46637838	0.57432432	0.57170909	0.58298630	0.59140659
120	0.46243243	0.57735135	0.57334545	0.58164384	0.59246154
140	0.47854054	0.57913514	0.56269091	0.58024658	0.59301099
160	0.47508108	0.58054054	0.57418182	0.58443836	0.59865934
180	0.47048649	0.57929730	0.57694545	0.58550685	0.59791209
200	0.47621622	0.58016216	0.57734545	0.58964384	0.59173626
220	0.48113514	0.57810811	0.57589091	0.58682192	0.60032967
240	0.47859459	0.58308108	0.57338182	0.59178082	0.59989011
260	0.47794595	0.58600000	0.57650909	0.59098630	0.59540659
280	0.47578378	0.58308108	0.58000000	0.58849315	0.60112088
300	0.48308108	0.58151351	0.57850909	0.59372603	0.59705495
320	0.48194595	0.58394595	0.58018182	0.59183562	0.59894505
340	0.47945946	0.58356757	0.57956364	0.59386301	0.60098901
360	0.47400000	0.58518919	0.57829091	0.58884932	0.59991209
380	0.48200000	0.58459459	0.57810909	0.59235616	0.59597802
400	0.48091892	0.58118919	0.57632727	0.59350685	0.59975824
420	0.47567568	0.58762162	0.57861818	0.59249315	0.60112088
440	0.48502703	0.58232432	0.58167273	0.58895890	0.60043956
460	0.47524324	0.58470270	0.58181818	0.59528767	0.59538462
480	0.48437838	0.58837838	0.57887273	0.59339726	0.59969231
500	0.47821622	0.58400000	0.58265455	0.59309589	0.59791209

Table 10. Accuracy achieved per epoch of trainng when training the CRNN with different augmentations on the dataset.

APPENDIX E

Results of hyperparameter tuning experimental training on the Latin alphabet

alphabet:	0123456789aáàâäbcdeéèêfghiíîjklmnoóòôöpqrstuúùûüvwxyzABCDEFGHIJKLMNOPQRSTUWXYZ.,!?:_@#&+-/%€()'							
data:	**original + 1 random rotation**							
batch size	**500**	**250**	**1000**	**2000**	**750**	**375**	**440**	**625**
size of lstm hidden state:	**256**							
epoch	**valid accuracy**							
0	0.0851	0.1115	0.0858	0.0839	0.0858	0.0886	0.0854	0.0858
20	0.6337	0.6333	0.6419	0.5823	0.6259	0.6471	0.6259	0.6390
40	0.6368	0.6378	0.6470	0.6011	0.6444	0.6475	0.6328	0.6448
60	0.6408	0.6378	0.6482	0.6012	0.6448	0.6519	0.6342	0.6473
80	0.6400	0.6406	0.6514	0.6130	0.6454	0.6546	0.6394	0.6519
100	0.6455	0.6380	0.6521	0.6068	0.6434	0.6496	0.6413	0.6474
120	0.6448	0.6424	0.6571	0.6136	0.6508	0.6548	0.6420	0.6555
140	0.6470	0.6350	0.6537	0.6148	0.6436	0.6548	0.6390	0.6534
160	0.6465	0.6469	0.6526	0.6139	0.6474	0.6530	0.6385	0.6547
180	0.6443	0.6418	0.6539	0.6166	0.6486	0.6538	0.6408	0.6556
200	0.6490	0.6377	0.6559	0.6173	0.6503	0.6520	0.6392	0.6518
220	0.6464	0.6334	0.6580	0.6193	0.6530	0.6550	0.6411	0.6556
240	0.6444	0.6402	0.6579	0.6230	0.6513	0.6549	0.6417	0.6582
260	0.6468	0.6359	0.6581	0.6221	0.6486	0.6576	0.6398	0.6569
280	0.6531	0.6409	0.6568	0.6213	0.6503	0.6521	0.6374	0.6508
300	0.6506	0.6388	0.6574	0.6203	0.6494	0.6573	0.6409	0.6571
320	0.6464	0.6423	0.6552	0.6231	0.6536	0.6569	0.6411	0.6500
340	0.6508	0.6330	0.6582	0.6231	0.6510	0.6537	0.6442	0.6530
360	0.6498	0.6424	0.6558	0.6218	0.6496	0.6519	0.6385	0.6535
380	0.6471	0.6405	0.6546	0.6216	0.6543	0.6528	0.6381	0.6563
400	0.6502	0.6398	0.6589	0.6231	0.6529	0.6512	0.6415	0.6564
420	0.6485	0.6418	0.6591	0.6265	0.6540	0.6556	0.6382	0.6552
440	0.6527	0.6471	0.6577	0.6280	0.6533	0.6579	0.6391	0.6583
460	0.6482	0.6457	0.6555	0.6255	0.6488	0.6507	0.6431	0.6594
480	0.6444	0.6370	0.6581	0.6251	0.6511	0.6559	0.6403	0.6584
500	0.6458	0.6402	0.6568	0.6172	0.6490	0.6545	0.6412	0.6595

Table 11. Accuracy achieved per epoch of training when training the CRNN with different batch sizes.

alphabet:	0123456789aáàâäbcdeéèêfghiìîjklmnoóòôöpqrstuúùûüvwxyzABCDEFGHIJKLMNOPQRSTUWXYZ.,!?:_@#&+-/%€()'					
data:	**original + 3 random rotations + hsv noise**					
batch size	**1000**					
size of lstm hidden state:	**256**	**512**	**128**	**64**	**192**	**320**
epoch	**valid accuracy**					
0	0.0857778	0.0857778	0.0857778	0.0857778	0.0857778	0.0857778
20	0.6419111	0.5746889	0.6308222	0.6205111	0.6112444	0.6208000
40	0.6470000	0.5852667	0.6376889	0.6223333	0.6177333	0.6294889
60	0.6481556	0.5920889	0.6413111	0.6320000	0.6232444	0.6329111
80	0.6513778	0.5958667	0.6468667	0.6320444	0.6238222	0.6269778
100	0.6521111	0.6002222	0.6426222	0.6340222	0.6282444	0.6352889
120	0.6570667	0.6023556	0.6427778	0.6373111	0.6277333	0.6394444
140	0.6536889	0.6029556	0.6482889	0.6399778	0.6312889	0.6412889
160	0.6525556	0.6040889	0.6525333	0.6392889	0.6338444	0.6408444
180	0.6539111	0.6057556	0.6526222	0.6387111	0.6331111	0.6450000
200	0.6558889	0.6070889	0.6524889	0.6450000	0.6387111	0.6488000
220	0.6580222	0.6068000	0.6536889	0.6426222	0.6364667	0.6437778
240	0.6579111	0.6057111	0.6528889	0.6446222	0.6356889	0.6496222
260	0.6580889	0.6088667	0.6500222	0.6423556	0.6379556	0.6496444
280	0.6567778	0.6096889	0.6536222	0.6436889	0.6385333	0.6480000
300	0.6574000	0.6036444	0.6518667	0.6427333	0.6380000	0.6503111
320	0.6552222	0.6069778	0.6581111	0.6448889	0.6353778	0.6488444
340	0.6582222	0.6079111	0.6545111	0.6455778	0.6365556	0.6507333
360	0.6557778	0.6024444	0.6488000	0.6441778	0.6358222	0.6510222
380	0.6546444	0.6041556	0.6548444	0.6430444	0.6410444	0.6480889
400	0.6589111	0.6048889	0.6552667	0.6448444	0.6389556	0.6527111
420	0.6591111	0.6056222	0.6558667	0.6429333	0.6404222	0.6503333
440	0.6576889	0.6044444	0.6547111	0.6402222	0.6384889	0.6472444
460	0.6554667	0.6062000	0.6531333	0.6418444	0.6392000	0.6511556
480	0.6580667	0.6093333	0.6567111	0.6426667	0.6392000	0.6525111
500	0.6568000	0.6103111	0.6530667	0.6432222	0.6439111	0.6487333

Table 12. Accuracy achieved per epoch of training when training the CRNN with different sizes for the LSTM hidden state.