UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA DE RECONOCIMIENTO ÓPTICO DE TEXTOS MANUSCRITOS EN ESPAÑOL

A Handwriting Recognition System for Spanish

Realizado por
JOAQUÍN ANTONIO TERRASA MOYA

Tutorizado por
JUAN MIGUEL ORTIZ DE LAZCANO LOBATO

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Septiembre de 2020

Resumen

Recientemente, los avances en computación, en miniaturización de componentes electrónicos, así como la creciente accesibilidad a dispositivos móviles, ha llevado a un aumento exponencial en el número de usuarios de computadores. Este cambio ha facilitado el acceso a una mayor parte de la población a tecnologías que hacen más cómodo el día a día. Desde mejorar la comunicación entre individuos hasta realizar recomendaciones sobre el consumo alimentario propio, los dispositivos móviles están permitiendo cada vez más una vida más sencilla, aunque aún hay margen de mejora.

Uno de los hábitos que no han sido totalmente sustituidos por la tecnología es escribir a mano. Aunque sí existen sistemas refinados de Reconocimiento Óptico de Caracteres, su principal uso es sobre textos impresos, ya que los métodos clásicos no son tan robustos para reconocer el constante cambio de estilo de la escritura a mano.

Gracias a los últimos avances en inteligencia artificial, principalmente en los sistemas de redes neuronales, existen tecnologías actuales que proporcionan una precisión mucho mayor en el Reconocimiento de Texto Escrito a Mano. Aun así, estos avances no han sido extensamente usados en aplicaciones actuales, como las creadas para toma de notas o escritura de documentos extensos; en estos casos, la facilidad y portabilidad de los documentos son factores decisivos.

En el trabajo presente, se propone un sistema de reconocimiento de texto escrito a mano que tenga en cuenta el estilo y formato del documento. A diferencia de estudios anteriores, el reconocimiento del color y estilo del texto tiene un papel central en el sistema, de forma que se puedan mantener palabras subrayadas o secciones del documento. Por último, y con el fin de dar soporte a textos escritos en castellano moderno, se crea un conjunto de datos novel, que también se libera para futuros avances.

Palabras clave: reconocimiento de texto manuscrito, procesamiento de textos, procesamiento de color, redes neuronales, aprendizaje profundo

Abstract

In recent years, the advances in computing power, miniaturization of electronic components, and accessibility to mobile computers have led to an exponential increase in the number of computer users. This change has increased the accessibility to modern technologies to a greater part of the population. From enabling faster communication among us to advising about our food consumption, mobile devices are being used to increasingly ease our daily life, although there is still room for improvement.

One of the tasks that has not been replaced with technology is handwriting. Even though Optical Character Recognition systems have been greatly improved, they mainly focus on the recognition of printed texts, and therefore the algorithms used are not robust enough for the constantly changing styles of handwriting.

Thanks to the latest advances in artificial intelligence, namely in the field of neural networks, current research is expected to greatly improve Handwriting Text Recognition accuracy. However, the application of such techniques has just started to be applied in modern forms of handwriting, like taking quick notes and writing down extensive documents, where the ease of use and portability are key issues.

In this work, a handwriting text recognition pipeline, which also takes text style and layout into account, is proposed. Unlike previous research, color and style recognition plays a key role in the workflow, so that highlighted words or document sections can be kept mostly unchanged. Besides, with the aim to support modern handwritten Spanish, a novel dataset is created, which is also made available for future projects.

Keywords: handwriting text recognition, text processing, color processing, neural networks, deep learning

Table of Contents

List of Abbreviations

HTR	Handwriting Text Recognition
OCR	Optical Character Recognition
ICR	Intelligent Character Recognition
AI	Artificial Intelligence
NN	Neural Network
DL	Deep Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CTC	Connectionist Temporal Classification

Chapter 1. **Introduction**

In this chapter, the main concepts of the system and how it works, and the motivation behind each of them, will be detailed. In addition, to improve the reader's understanding of the reach of the project, a brief background of the topic is given. Finally, a description of the State-of-The-Art research on the topic is given.

1.1. Context

The desire to reproduce written content to multiple forms has been a priority since the time of the printing press, back in the 16th century. Thanks to the advances of technology, nowadays we can keep the same content in multiple formats, like handwritten text, printed text, or digital text. As of today, the goal is now different: Can text information, presented in one format, be recognized, and extracted so that it can be moved to another format, directly?

When it comes to the conversion from printed machine text to digital format, Optical Character Recognition (OCR) is the main approach. By using a set of image processing techniques, as well as expert typographic knowledge, software applications can pre-process and extract information from a photography or a scan of printed text, and then process this information to find out what may be written in the paper, newspaper or document. This way of extracting information from text forms has become quite robust (Rice, Jenkins, and Nartker 1995) to different text fonts and use cases – uneven lighting, small font size, or unique text layout. Modern OCR applications range from car license plate recognition to searchable textbook databases (Google 2011), or commercial document text search tools.

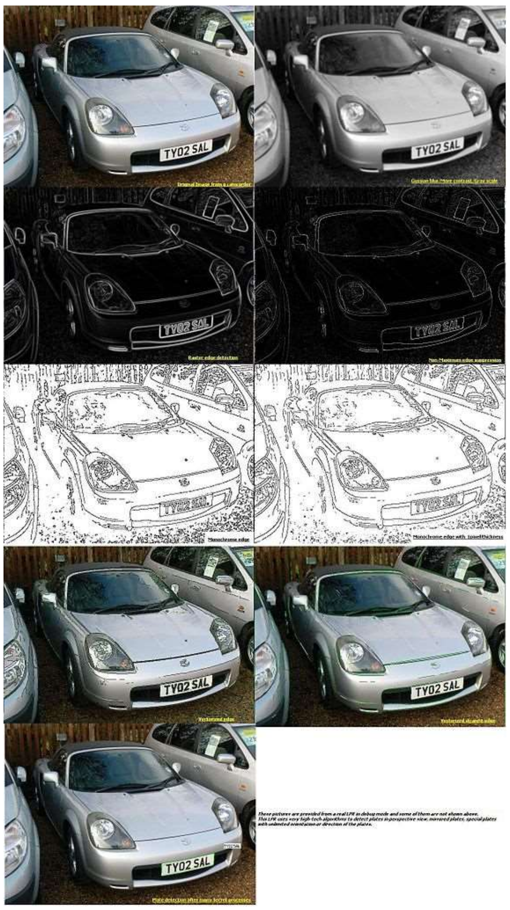
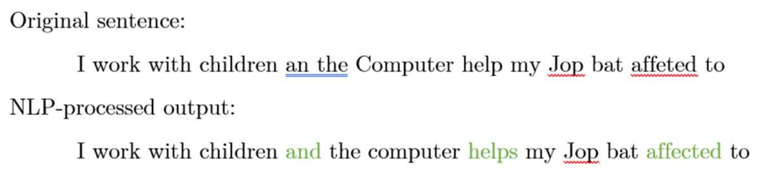

These pictures are provided from a real LPR in debug mode and some of them are not shown above.
This LPR uses very high-tech algorithms to detect plates in perspective view, mirrored plates, special plates with unlimited orientation or direction of the plates.

Figure 1.1: OCR system for plate recognition.

Thanks to the latest advances in artificial intelligence (AI), a new approach, evolved from OCR, is being applied to allow conversions from handwritten text to digital text. Intelligent Character Recognition (ICR) leverages classical OCR techniques and applies AI to extract information from uneven writing styles (Filestack 2018), a common feature of handwritten or custom text fonts.

A common approach to improve ICR results by means of AI is to use neural networks (NN): systems that recall biological neurons and can learn from experience (Lauzon 2012) – allowing data to drive its learning. By combining multiple NN architectures, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), this approach can learn about the visual properties of text, as well as the lexical, syntactical, and semantical attributes of the language.

To enhance even more the results, this approach is usually combined with natural language processing (NLP) techniques, which seize the extracted properties and attributes to create a more realistic result.



Original sentence:

I work with children an the Computer help my Jop bat affeted to

NLP-processed output:

I work with children and the computer helps my Jop bat affected to

Figure 1.2: Sample from a language-processing module

There are, however, two core subproblems in ICR: online recognition, which is based on data recorded by a digital notebook, like a tablet; and offline recognition, which works directly on pictures of handwritten text on paper. It is also important to note that the main datasets used in these subproblems are made up of English and French corpus (namely, the IAM and RIMES (Grosicki and El-Abed 2011; Marti and Bunke 2003) datasets). Therefore, there is not only the issue of handling different languages, like Spanish, Portuguese, or Czech, but the restriction of not using common symbols, like *$*, *@* or *%* is there as well.

Besides, these datasets only use grayscale, pre-processed images (or digital records for online recognition) that are unable to represent color styles typically used in handwritten text. Even though top-notch tools like Google Lens (Google 2020) still extract text from natural scene images, they cannot exploit its color properties.

The scope of this project is to address some of the problems detailed before. On one side, it seeks to expand ICR advances to Spanish corpus, by creating a curated dataset and developing an ICR system capable of processing it. On the other side, it addresses the opportunity for leveraging text style and color, in such a way that the ICR system can reproduce the styling and layout of the original text.

1.2. State of the Art

To further realize how far the previous developments in the topic have gone, and how this project can fit in the timeline, it is essential to survey the most remarkable works. Nowadays, most NN-based ICR systems take the ideas from the concepts of Seq2Seq (Sutskever, Vinyals, and Le 2014) and CTC loss (Graves et al. 2006).

Briefly, the Sequence-to-Sequence (Seq2Seq) pattern proposes an encoder-decoder architecture to predict the network result by relying on an internal state; this way, the network model can produce an output of a different shape from the input, and the network architecture and learned weights can be reused for other purposes (what is known as *transfer learning* (Pan and Yang 2010)).

In a similar way, Connectionist Temporal Classification (CTC) loss looks at simplifying how a prediction can be matched to a sequence. Given an audio clip, we seek to predict its transcript. However, we do not know how words in the transcript align to the audio shape. Moreover, as the speech rate varies from person to person, maybe *z* occupies 3 timeslots in an audio and 1.2 timeslots in another one.

Thus, the CTC algorithm makes a guess for each timestep from the input, in such a way that if the same character is predicted twice or more times in a row, it is later merged as a single one. If the word contains 2 equal characters in a row (e.g. *hello*, *ll*), a special character is used to separate these, so that they are not merged.

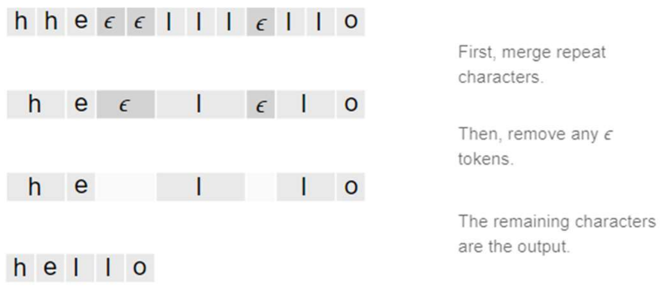

h h e ϵ ϵ l l l ϵ l l o
h e ϵ l ϵ l o
h e l l o
h e l l o
First, merge repeat characters.
Then, remove any ϵ tokens.
The remaining characters are the output.

Figure 1.3: CTC algorithm, visually explained (Source)

By taking advantage of these concepts and multidimensional LSTM (Hochreiter and Schmidhuber 1997) layers, great improvements in the field of offline and online HTR at word level are obtained in the works of (Graves and Schmidhuber 2009) and (Graves 2013). Thanks to the recent advances in GPU training, these architectures have been also applied to line level (Voigtlaender, Doetsch, and Ney 2016) and full-page level (Bluche, Louradour, and Messina 2017).

However, recent works (Puigcerver 2017) suggest that using CNNs for feature extraction along with a unidimensional LSTM, instead of multidimensional LSTMs, greatly reduces the computing power requirements. Furthermore, a set of layer optimizations for RNNs and CNNs on dimensionality reduction (Chen et al. 2019; Laurent et al. 2016; Pham et al. 2014) and computing power reduction (Jacob et al.

2018; Narang et al. 2018) have allowed for huge improvements in training and inference speeds (Xu et al. 2018).

These systems, which target to offline HTR recognition, are mostly based on standardized datasets, namely RIMES and IAM, which are composed of scripts from the 20th century written by more than 500 writers. Even though there are works that introduce Spanish or Catalan texts (Romero et al. 2013), these contain mainly scripts written by 2 or 3 writers using 16th-century forms and writing style of the given languages.

To this date, however, there is no other equivalent dataset targeting the remaining Latin symbols, such as *ñ*, *ý,* or *ÿ*, so these cannot be guessed by the aforementioned systems. There have been works trying to extend the reach of these systems to new symbols (Al-Ma'adeed, Elliman, and Higgins 2002; Alonso, Moysset, and Messina 2019; Hull 1994).

Besides this, a different track for improving OCR and ICR precision is to make the system flexible to scene changes. Typically, these changes influence text rotation, font and background color, and type font. This topic has been extensively researched for OCR systems (Saidane and Garcia 2007; Shi et al. 2016; Shi, Bai, and Yao 2017) but no work has been found for ICR systems. However, some works do not focus neither on OCR nor on ICR, and provide a generic framework for unconstrained document inputs (Li et al. 2008; Montreuil et al. 2009; Quirós 2018).

A final point to highlight is that most recognition systems also implement language post-processing features. Namely, (Bluche et al. 2017; Chung and Delteil 2019) take advantage of the contextual information from a sentence to correct guessing errors for each word (Kukich 1992). However, none of them exploit the information from the document layout, such as font size, emphasis, or indentation within each paragraph.

In this project, the latest advances in NN-based, efficient ICR systems are combined with features from unconstrained document recognition systems. Finally, its output is enhanced by applying language-processing methods (Parr and Quong 1995) extensively used in modern compiler designs.

1.3. Scope

The goal of this project is to extend the latest advances on ICR systems based on Deep Neural Networks (DNN). Namely, it seeks to complete three objectives:

1. Create a Spanish dataset for offline handwriting text recognition and deliver a trained ICR system on this dataset.
2. Build up an AI system to extract color from words.
3. Apply language processing and language compiling techniques to reproduce handwritten layout into a digital layout, through a markup language.

Chapter 2. **Project Planning**

2.1. Task Descriptions

In order to fulfill the project goals, a set of modules must be created to meet the requirements. Therefore, the same procedure will be carried out for each module: research about state-of-the-art schemes for the given topic, design a system based on the results of this research, and then implement it. Hence, the following tasks will be conducted:

1. Research, design, and implementation of the handwritten text recognition module.
2. Research, design, and implementation of the handwritten Spanish text database.
3. Research, design, and implementation of the text color recognition module.
4. Research, design, and implementation of the style and layout language processor module.

   Note that the way this module is implemented in the current work is by recognizing a defined handwritten markup language.

Finally, to provide a minimum viable product or application prototype, a set of extra modules are created to bind the first four modules.

5. Research, design, and implementation of a text segmentation module.
6. Research, design, and implementation of an utility module.

A set of validation tests is created for each task, as a way to set a task milestone, except for the utility module. In addition, a validation test is conducted when the system is finished.

2.2. Scheduling

This project fits into the last stage of a bachelor's degree. To this end, it is equivalent to 12 ECTS, which matches to 296 hours. Therefore, a thorough review of the stated stages and its nested milestones have been done to fit them into the schedule. In the following sections, a timetable that matches the schedule settings is detailed, and the methodology that will allow the researcher to follow such timetable is explained thereafter.

2.2.1. Timetable

STAGE	HOURS
Research about state-of-the-art schemes for HTR on word level.	20
Research about state-of-the-art schemes for handwritten text color recognition (HTCR) on word level.	20
Research about state-of-the-art schemes for language processing.	10
Research about state-of-the-art schemes for creating handwritten text databases.	20
Design of validation and final tests for the HTR, HTCR, and language processing modules.	20

Create the main system. Apply the HTR scheme and extend it for full-page recognition.	20
Validation tests on the main system.	11
Creation of the database of handwritten text.	20
Database integration with the main system.	20
Validation tests on the main system.	15
Extend the main system with the HTCR module.	20
Validation tests on the main system.	15
Design of the language processing layouts.	20
Extend the main system with the language processing module.	20
Validation tests on the main system.	10
Final tests on the main system and results.	20
Design and development of the main system's demo.	15
	296

Table 2.1: Expected time planning for the project

2.3. Methodology

Given the aforementioned aims and time constraints, as well as the human resources constraints, an agile methodology is used to reach the goals within such a deadline. Namely, a version of the Feature-Driven Development (Ambler 2005) process is used:

- The project is structured by modules, each one addressing a required functionality.
- The evolution of the project is measured on behalf of the progression of each module. Therefore, at the beginning of the project, the design and implementation of each functionality is planned. At the end of each module, a milestone is set to track the progress.
- Absolute time is divided into 1-week blocks. To keep track of the changes made, and if help is needed right away, a report is delivered when shifting from one block to the next.
- The schedule is open to modifications in case of having obstacles.

Moreover, to monitor changes done throughout the project in each of the modules, particularly in the design and development, a git (Torvalds 2005) local repository is used to prevent from losing any progress. Likewise, to keep a record of the use of external source code, Anaconda environments (Anaconda 2012) are used.

2.4. Obstacles and Risks

In this section, the possible obstacles and risks that might appear throughout the development of the project are described. Additionally, for each obstacle, an alternative is detailed in the case the addressed problem arises.

2.4.1. Obstacles

1. Learning curve of Machine-Learning methodologies

Even though the project researcher has previous training in the latest machine learning techniques, there are new concepts, specific to language processing and model optimization, which may slow the research pace.

To overcome this, the potential techniques that are new to the researcher are thoroughly reviewed during the *Research* stage of each task.

2. Data generation inconsistency

The need to create a dataset from scratch, and to extend (data-augment) it, puts pressure on producing a consistent, uniformly distributed dataset. However, the result can differ from the reference datasets (IAM, RIMES) in multiple factors: character properties, word resolution, image histogram balance, etc.

To tackle this issue, the resulting samples from the dataset will be carefully selected according to the reference datasets, in such a way that the factors abovementioned do not pose a threat to the system performance.

3. Error-prone machine learning packages

Although the software packages used for the project are commonly used by the machine learning community, they are not flawless. Here, we have to take into account that this software has been created in the latest 5-10 years; hence, it is still experimental for some stages of the scheme and there is not a consistent interface.

As a way to solve this issue, two different software tools will be used to design the model architectures and train them. These tools share a common API for most of their features.

4. Poor accuracy from using novel model architectures

As some modules have no counterpart in previous research, there is a chance of not getting the desired output from the proposed, experimental schemes. To tackle this issue, classical techniques that have been long used for similar problems are also taken into account, and multiple configurations for each scheme are tested.

2.4.2. Risks

1. Poor accuracy from using the novel dataset

One of the main drawbacks of creating a novel handwritten text dataset for offline recognition is the need to match the specifications of IAM or RIMES datasets, which are huge, and provide a high amount of text and writers.

Since the novel dataset contains a low amount of text and writers compared to these datasets, there is a chance that the trained neural model yields poor accuracy from this fact.

2. Unstable machine-learning architecture design

Due to the fact that implementing neural network schemes right away can be tedious, and to the inexperience of the project researcher, some key attributes might be wrongly implemented.

3. Failure to get an MVP

Due to the sophistication of the proposed project, as well as the need for extra modules to provide a minimum viable product (MVP), the complexity of the project might be higher than expected. Thus, there is a chance that project re-planning is needed to fit the deadline.

2.5. Resources Used

In this section, the software and hardware resources that have been used to attain the project goals are detailed.

2.5.1. Hardware resources

- Laptop PC: Intel i7-8565U CPU with Intel GPU 630, 8GB RAM
- Google Collaboratory (Google Colab Team 2019) instance: Intel Xeon CPU, 12GB RAM, Nvidia Tesla P4 GPU.

  Google Collaboratory provides an easy-to-use IDE to develop machine learning algorithms. However, user sessions have a limited time of 90 minutes, and all the uploaded or generated data is deleted when the time limit is reached.

2.5.2. Software resources

2.5.2.1. Python 3

Python 3 (Python Team 2018) is a programming language that is widely used in the data science and machine learning fields. In short, it focuses in a simple syntax and a wide support of technologies, being currently used for server programming, data processing pipelines and web development among others. In addition, the following python modules, known as *packages,* have been used in this project:

1. $numpy$ provides a fast, numerical computation interface to Python. It is used all across the system, from image processing to data management.
2. $pandas$ provides a SQL-like interface to Python, by loading structured datasets in tables and allowing to run queries on these datasets.
3. $OpenCV$ is a framework, also available in other languages like $C++$, to simplify the tasks of image processing and computer vision.
4. $Numba$ (Anaconda 2018) allows to compile Python by using a *Just-In-Time* (JIT) compiler. This allows the system to obtain the advantages of compiled code, such as lower time complexity, while retaining the advantages of Python, such a simpler syntax.
5. $Jupyter\ Notebook$ (Project Jupyter 2017) provides an interactive development environment to allow faster development times.

2.5.2.2. Machine Learning Toolkit

The following technologies have been used to design and implement the machine learning modules:

1. $scikit-learn$ is a Python interface to the core machine learning algorithms, such as *K-Means* and *Support Vector Machines.*
2. $Tensorflow\ 2.0$ (Google Brain Team 2015) is the second revision of $Tensorflow$, a toolkit for developing neural networks. It provides a basic interface, $Keras$, which allows the researcher to build the neural network model by layers. In addition, it has support for a wide range of devices, such as laptops or embedded devices, which is useful to test the reach of the project.

2.5.2.3. Language Processing Toolkit

To be able to handle and process the complex text structure outputted by the Text Recognition and Color Recognition modules, the following technologies have been used:

- *pyspellchecker* is a Python interface that provides a spell checker. This is used to ensure that the predicted text from the Text Recognition module is correct in form and syntax, and to correct it if necessary.

3. *ANTLR4* (Parr and Quong 1995) is a tool to create *LL(*)* parsers. It is used to ease the creation and modification of the markup language processor.

2.6. Deviations from the original planning

Multiple deviations have occurred during the development of the project. Firstly, due to the limited knowledge of the researcher in Deep Learning techniques applied to Natural Language Processing, the research and design stage of the text recognition module has been combined with its corresponding implementation stage, to ensure that a right model architecture is delivered.

Secondly, on early versions of the novel dataset, the accuracy of the text recognition model when using the dataset to train it was poor. This may have been due to either the varying properties of the text images gathered or the contrast between the novel dataset and IAM and RIMES dataset. To improve these results, 2 extra weeks have been devoted to this stage, and the timetable has been reorganized so as to fit the new time constraints.

Thirdly, the initial models designed for text segmentation were faulty and difficult to implement. Namely, a deep learning architecture was proposed for the line and word segmentation blocks, which have been lately replaced by statistical models. This decision has been made attending to two additional metrics: lack of training data and robustness to unprocessed data. The findings are detailed in the corresponding sections.

Chapter 3. **Theoretical Foundations**

In this chapter, the core of the technologies and methods used in this project are described. Firstly, an overview of the classical approaches to handwriting text recognition, mainly from the field of computer vision, is given. Secondly, the neural network model, which greatly improves the results of the classical approaches, is described thoroughly.

3.1. Introduction to Handwriting Text Recognition

Handwriting Text Recognition can be defined as the task to process a text source, usually in the form of an image, and transcribe its content to a digital form. It is a classical problem in the field of computer vision, and there are many ways to approach it. A common solution is to use an *Optical Character Recognition* algorithm, which recognizes the text content at character level, and forms the sentences upwards. It does so by carefully studying the features of each character, and their most common deviations to adapt to any text.

However, this approach gets inconsistent results when dealing with unknown inputs. Think, for example, if the machine is dealing with a text source written with a slightly different alphabet, or with a cursive style. One of the reasons of this problem is that OCR algorithms do not implement, by themselves, a language engine that can handle contextual information at paragraph, line, or word level.

A recent approach to this problem is to use algorithms, based on Artificial Intelligence, that both leverage the advantages of computer vision and contextual information.

3.1.1. Concepts based in Computer Vision

To better understand the advanced algorithms that this project uses, it is recommended to learn in advance about some of the concepts from the field of computer vision that are used in this project:

3.1.1.1. Vertical and Horizontal projection

Given a 2D binary image, where a value of 0 represents information, how do we get the position of the bits holding information? A simple way to get it is to use the projections of the image.

The vertical projection can be defined as the sequence of the sums of pixels grouped by its vertical coordinate. For example, for the following matrix

$$\begin{matrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$$

with dimensions $n \times n, n = 3$. The resulting sequence would be $[2, 0, 3]$. Similarly, the horizontal projection for this matrix would be $[2,1,2]$.

In addition, we define the concepts of *valley*, being an element of the projection whose value is n, and *peak*, being an element of the projection whose value is lesser than $k \in [0 \dots n-1]$. In Figure 3.1, a text image and its associated vertical projection is shown.

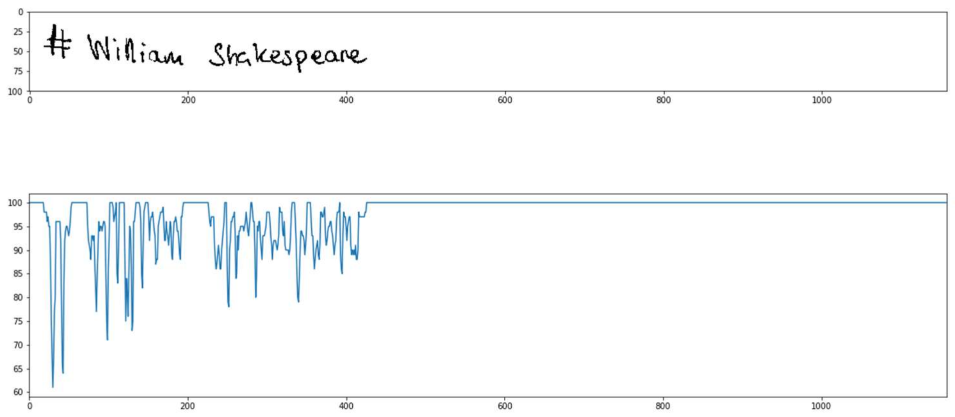


Figure 3.1: Text image and its associated vertical projection

3.2. Foundations of Advanced Artificial Intelligence

One of the key abstractions that has accelerated the improvement of HTR systems in the last one or two decades is the concept of neural networks (Rojas 1996). In simple terms, a neural network leverages the organic structure of neurons to improve feature learning. In Figure 3.2, a *multilayer perceptron*, one of the core neural network architectures, is shown.

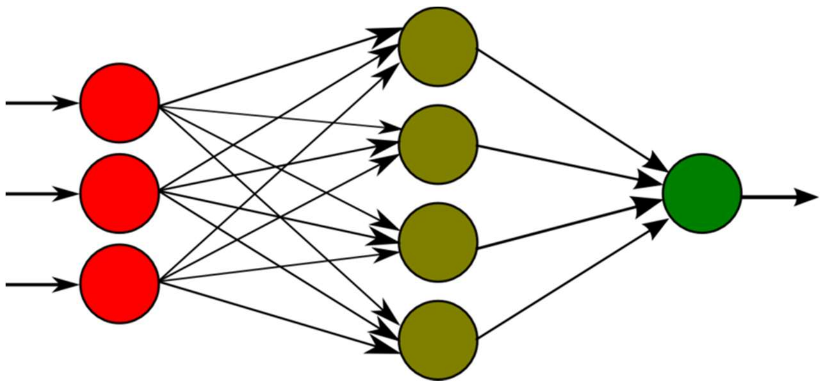


Figure 3.2: A Multilayer Perceptron (MLP)

To better understand the core ideas of the field neural networks, it is recommended to review the next widely used neural network models.

3.2.1. Feed-forward Neural Network model

One of the fundamental neural network models is the feed-forward neural network model. One of the key examples is the multilayer perceptron model shown in Figure 3.2

Mainly, each abstract neuron, which is represented by a circle, is grouped in a vertical set or layer, just like real neurons; in a similar manner, these layers are connected onwards, so that there can be no cycles. Therefore, the leftmost layer (red neurons) is referred to as *input layer*, whereas the rightmost layer (green neuron) is referred to as *output layer*. All the other intermediate layers are called *hidden layers*. Besides, each layer has a level of depth, being 0 for the *input layer* and n for the *output layer*. Depth can be associated with an expression $depth(A)$ for *input layer* A.

Finally, all the neurons from a given layer are *fully connected* to all neurons in the next layer – for any given layers A and B such that $depth(B) = depth(A) + 1$, there is a link between each pair of neurons $(n_A, n_B) \mid n_A \in A, n_B \in B$. These links have an associated *weight* so that the network can learn which connections are more important for each neuron. Hence, the purpose of training a neural network, given a specific kind of data, is to learn which weights suit best each neuron so that the network can get a precise output representation from the inputted data.

3.2.1.1. Learning

As explained before, each neural connection has an associated *weight* which tells the neuron how important that connection is with respect to the other connections the neuron has. The set of all connection weights among neurons of the layer n and the layer $n+1$ are represented in a *weight matrix*. Usually, to ensure that the network is not pre-conditioned to learn the solution to a given problem, this matrix is initialized with random values.

For example, imagine that a neural network is used to solve the following problem: split a set of 2D points in two groups: one must be over the line $f(x) = x$ and other below it. We already know the groups for the following points:

POINT	GROUP	VALUE
$(\mathbf{1},\mathbf{0})$	Below	0
$(\mathbf{0},\mathbf{1})$	Above	1
$(\mathbf{5},\mathbf{4})$	Below	0
$(\mathbf{5},\mathbf{10})$	Above	1

Table 3.1: Sample dataset of 2D points

Each neuron has a *transfer function* that is used to combine all its inputs and weights into a single output. This output is later passed onto the neurons of next layer, which determine, on behalf of its associated weights, if the output is meaningful or not.

A trivial transfer function can be

$$f(h) = \begin{cases} 1, & if\ h \geq \theta \\ 0, & if\ h < \theta \end{cases},\ h = \sum x_i \cdot w_i\ , \theta \in\ R$$

Being h the sum of the product between each input and its associated weight, and θ a threshold, which for this example can be $\theta = 0$.

With this in mind, note that the weights are updated each time a sample from the Table 3.1 of points comes in. To focus on the behavior of this step, the single perceptron, as shown in Figure 3.3, is more intuitive.

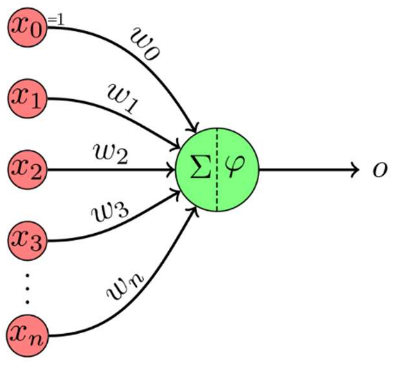


Figure 3.3: A singleton perceptron

To ease the task of updating the weights, we also need to define a learning rate η, a dynamic value, usually going from 1 to 0, that tells the significance of the next weight update. The reason behind this is to help the network to learn faster at the start and learn at a lower rate later, so that its results do not change drastically.

We can define the variation of the weight w_j at timestep k as

$$\Delta w_j(k) = \eta(k) \cdot \big(z(k) - y(k)\big) \cdot x_j(k)$$

Being $z(k)$ the expected group value and $y(k)$ the predicted group value. All the weights related to the connection from each input to the singleton perceptron can be grouped in a weight vector V.

Finally, in order to know when to stop updating V, the network can keep track of the last m updates of V, and if when comparing the last $t = 2$ updates, the change from one to another is not above a threshold λ:

$$\forall\, w_i \in V_n, \qquad \forall w_j \in V_{n-1}, \qquad w_i - w_j \leq \lambda$$

Where V_n is the last update and V_{n-1} is the last but one update. Then, it is said that the network is stable and the learning finishes.

3.2.1.2. Backpropagation algorithm

The learning stage on the singleton perceptron is simple, but to do it in the multilayer perceptron, an additional step is required. That is because the learning algorithm does not take into account the final output for intermediate neurons. In order for the learning stage to take into account this, the backpropagation algorithm is used.

The backpropagation algorithm introduces the concept of *gradient descent*, which uses the update of the other neurons to which a particular neuron X is connected, to influence the update of this neuron X. In this algorithm, the weight update function is

$$\Delta w_{ji}^{a} = \eta \cdot \sigma_{j}^{a} \cdot S_{i}^{a-1}, \qquad \Delta w_{ji}^{1} = \eta \cdot \sigma_{j}^{1} \cdot x_i$$

Being a the depth of the layer for the weight between the neuron j and the neuron i, S_i^a the output of the transfer function in i-th neuron in the a-th layer, and σ_j^a the error update of the j-th neuron in the a-th layer. Note that in the multilayer perceptron there is a weight vector V for each neuron. To keep it simple, all the weight vectors from a layer can be grouped in a weight matrix W.

We can define the error update in the output layer with depth N as

$$\sigma_i^N = (S_i^N)' \cdot (z_i(k) - y_i(k))$$

Which is the product of the first derivative of the output of the transfer function with the difference between the expected output and the real output for the i-th neuron at timestep k. For the other layers, the error update is

$$\sigma_j^n = \left(\sigma_j^n\right)' \cdot \Sigma_{i=1}^{M} w_{ij}^{n+1} \cdot \sigma_i^{n+1}$$

3.2.2. Deep Learning

However, the exclusive use of one or more blocks of the *multilayer perceptron model* does not allow for state-of-the-art models. Instead, by combining it and other types of neural models in big schemes, much more complex problems, such as face recognition, can be addressed and solved. This is what it is called *Deep Learning*, referring to the total depth of the network in terms of layers, reaching depths in the order of hundreds (He et al. 2016; Huang et al. 2017).

In order to be able to process and learn data representations of all kinds (from images, audio, text, etc.), specialized neural network models have been designed. Two of them are the *Convolutional Neural Network* model and the *Recurrent Neural Network* model.

3.2.2.1. Convolutional Neural Network model

One of the problems of the feed-forward model is that they are liable to overfitting data. Moreover, it struggles when dealing with large data inputs. For example, when using an image of dimensions 150×150, flattening it makes it a vector of size 22500, and just for the input layer, a feed-forward neural network with n neurons would need $22500 \cdot n$ connections.

To solve this problem, the convolutional neural network (CNN) model (Zhou 2019a), also known as the space invariant neural network model, seizes the hierarchical structure of data to learn about smaller and simpler patterns, which are later summed up to learn complex patterns. In addition, they use a convolution function to update the weight matrix.

Take for example the following 4×4 image as an input, which represents a T:

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{matrix}$$

If we use a convolutional layer to learn the features of this image, instead of connecting each neuron to a vector of length 16, we will take advantage of the spatial uniqueness of images. Thus, we can apply a 2D convolution of size 2×2 to obtain a smaller representation

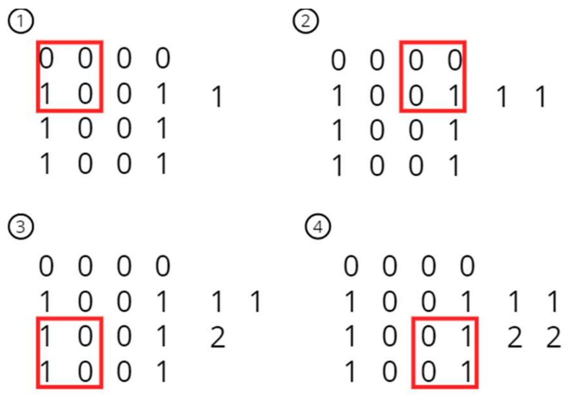


Figure 3.4: A convolution step in a CNN layer

Doing this step in sequence allows the convolutional model to learn about the features of an image without spending too much time nor space on computing the weights, as it happens in the feed-forward model. Thus, they are widely used in the computer vision field.

In this project, there are used 3 variations of the common convolutional model: the depth-wise separable convolutional model (Chollet 2017), the gated convolutional model (Lin et al. 2019), and the octave convolutional model (Chen et al. 2019). Each variation deals with certain disadvantages of the original convolutional model, such as the training and inference performance, the size of the weight matrix or the overall information processed by a convolution operation.

3.2.2.2. Recurrent Neural Network model

The Recurrent Neural Network (RNN) model (Zhou 2019b) thrives to model time-based or sequence-based data. Namely, they are designed to learn the structure of sequential data by keeping a hidden state between every pair of steps $(n, n+1)$ in the sequence and combining these hidden states later on in the output.

One of the main reasons to use the RNN model instead of the feed-forward model or the CNN model is that these two deal with fixed-size inputs and outputs, whereas the RNN model can work with a variable-length input and output. This renders useful for dealing with tasks such as language translation, where the input length may differ from the output length (for example, *buenos días* in Spanish and *bonjour* in French).

The RNN model works by adding internal connections among the neurons in a layer; however, these connections are done sequentially, so that no cycle can be done. Take for example the task of language translation as an example. If the model takes *hola* as an input, we could look at the RNN layer and it would be like

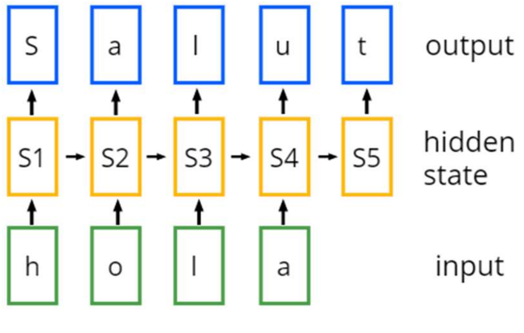


Figure 3.5: Representation of a RNN layer for a sample input

By keeping the internal state for each input character and updating the nth internal state with the $(n-1)th$ internal state, it can generate an output with a different length than the input length.

In addition, the following variation of the RNN model are used in the project as well:

- The Long-Short Term Memory (LSTM) model is better than vanilla RNN model when learning long sequences. This is due to the fact that the RNN model struggles with keeping information about the initial cells in the internal state when reaching the last cells of the layer, also known as the *gradient vanishing problem.*

Chapter 4. **Design and Implementation of the proposed system**

In this chapter, an overview of how each module of the system has been designed and developed is given. Firstly, the proposed system is described, emphasizing which module covers which functionality, and the critical points to look out. Secondly, the design and implementation of each module is explained.

Note that, as each functionality must cover a set of tests, these tests are also used to improve the design and implementation of each module. These tests are described in the following chapter, along with an analysis of the test results.

4.1. Description of the proposed system

In order to provide the reader with a broader view of the proposed system, a brief description of each module is provided, along with a full picture of how these modules are organized.

Namely, the following modules are used, each one addressing a core functionality in the system:

1. Image Input: a simple image processing module to convert an image to RGB, Grayscale or Binary mode.
2. Text Segmentation: it solves the issue of having to process big text images containing paragraphs. To this end, it breaks paragraphs into line and word images, which are then inputted to the next modules.

3. Data Processing and Data Augmentation: an utility module to enable easy-to-use data processing and data augmentation methods. In addition, it also contains the data generator for the Spanish dataset.
4. Text Recognition: it addresses text recognition. Specifically, it contains the neural model that recognizes the inputted text source. Besides, it includes a spell checker to seize contextual information from lines. It produces UTF-8 text as output for each line.
5. Color Recognition: it addresses color recognition by providing a neural model to recognize highlighted words and a color classification algorithm to transform RGB colors into color names (i.e. (255,255,255) to *white*). It produces a color tuple of (font color, background color) as output for each word.
6. Language Processing: it provides a means for producing custom outputs by combining text and color information. Particularly, it provides a specification for the *MiniDownColor* language, which processes text and color information inputted from the previous modules to create a consistent HTML5 output.

In Figure 4.1, a flowchart of the system is shown.

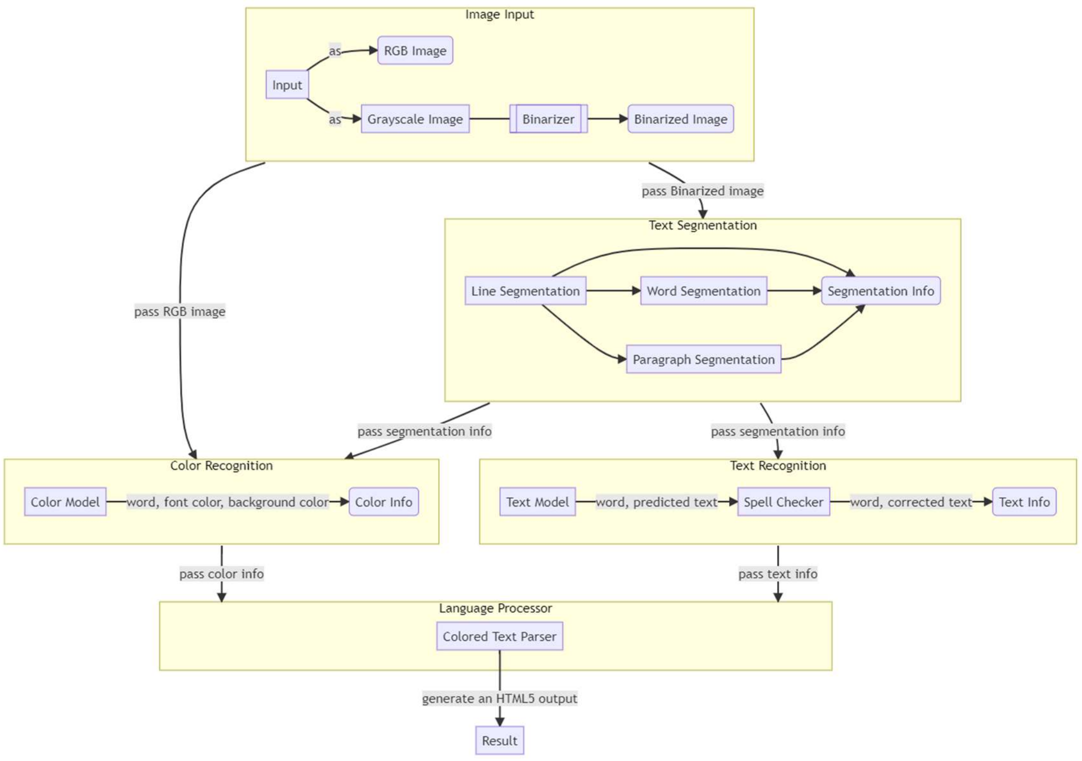
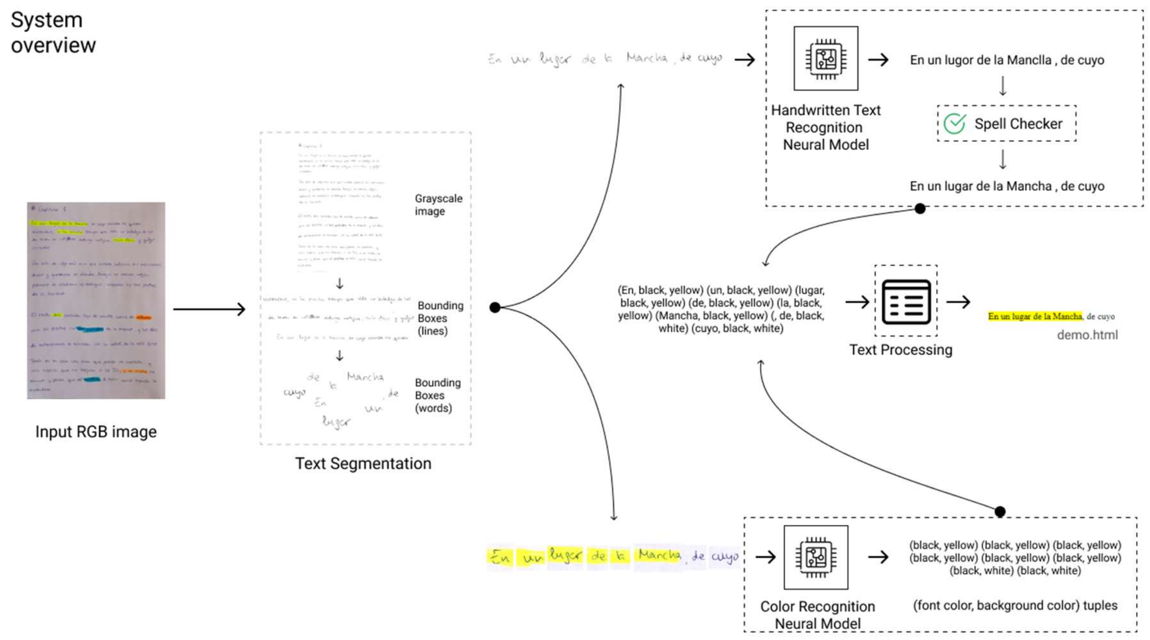


Figure 4.1: Flowchart of the proposed system

In Figure 4.2, a flowchart detailing the behavior of the system is shown.



System overview
Input RGB image
Grayscale image
Bounding Boxes (lines)
Bounding Boxes (words)
Text Segmentation
En un lugar de la Mancha , de cuyo
Handwritten Text Recognition Neural Model
En un lugor de la Mancl1a , de cuyo
Spell Checker
En un lugar de la Mancha , de cuyo
(En, black, yellow) (un, black, yellow) (lugar, black, yellow) (de, black, yellow) (la, black, yellow) (Mancha, black, yellow) (, de, black, white) (cuyo, black, white)
Text Processing
En un lugar de la Mancha, de cuyo
demo.html
Color Recognition Neural Model
(black, yellow) (black, yellow) (black, yellow) (black, yellow) (black, yellow) (black, yellow) (black, white) (black, white)
(font color, background color) tuples

Figure 4.2: Behavior of the proposed system

4.2. Design and Implementation of the system modules

To ensure that the functionality tests are meaningful, the *loss of information* metric is used. It is defined as the *L2* norm applied to image data, and it is stated in percentage values. Thus, a 100% loss of information given by a text image T with respect to the expected text image T' means that T is completely white, while a 0% loss of information means that $T = T'$.

4.2.1. Image Input Module

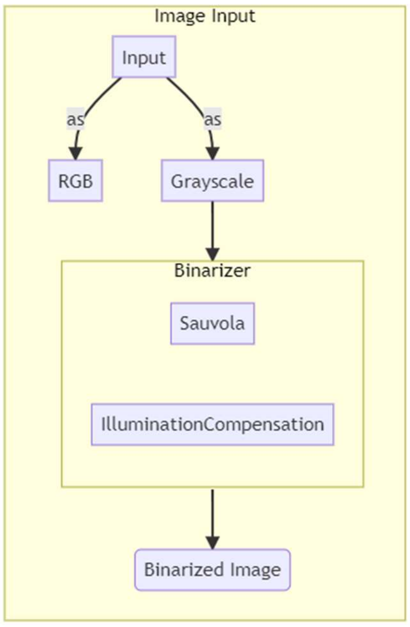


Figure 4.3: Flowchart of the Image Input Module

This module uses two main classes:

- **ImageInput** class, which defines the methods to convert a colored image to RGB mode and grayscale mode.
- **Binarizer** class, which defines the methods to binarize an image.

The **ImageInput** class uses built-in methods from the *OpenCV* package to convert the colored image, which is usually in RGB mode, to grayscale and RGB mode. However, as some text images might contain highlighted words, these can appear darker or lighter depending on the highlight color. Therefore, instead of averaging the RGB channels when converting to grayscale, the three channels are separately and then averaged.

The **Binarizer** class is somewhat more complex. As all HTR models use binarized or grayscale images for training or prediction, it is of vital importance to have a robust algorithm that can process raw input images.

Moreover, a binarization technique is applied to each RGB channel. Due to the chance of having to process large images (2 megapixels and more), multiple adaptive threshold algorithms have been tested. For this step, qualitative testing is chosen as the validation testing method, as images also contain highlighted words, which are difficult to process for some algorithms.

In Figure 4.4, a comparison among the used algorithms (Chen, Chen, and Chang 2012; Niblack 1985; Sauvola and Pietikäinen 2000) is shown. Note that all algorithms are designed specifically for processing grayscale text images.

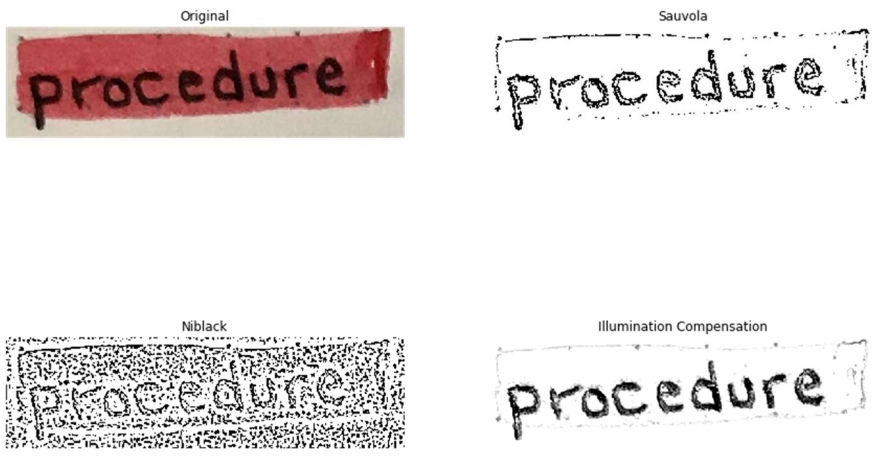


Figure 4.4: Binarization algorithms

The chosen algorithm is *Illumination Compensation*(Chen et al. 2012), as it proves to be more robust than the other ones, at the cost of higher time-based computational complexity. Moreover, Niblack and Sauvola thresholds are greatly influenced by the window size used, which inherently depends on the input image size. Lastly, Niblack and Sauvola cannot deal with highlighted words as elaborately as *Illumination Compensation* does.

In Figure 4.5, a comparison among these algorithms for a paragraph image is shown. Note that the Niblack algorithm is discarded as it binarizes images but generates too much noise. Besides, multiple configurations for the window size WZ parameter are used for the Sauvola algorithm.

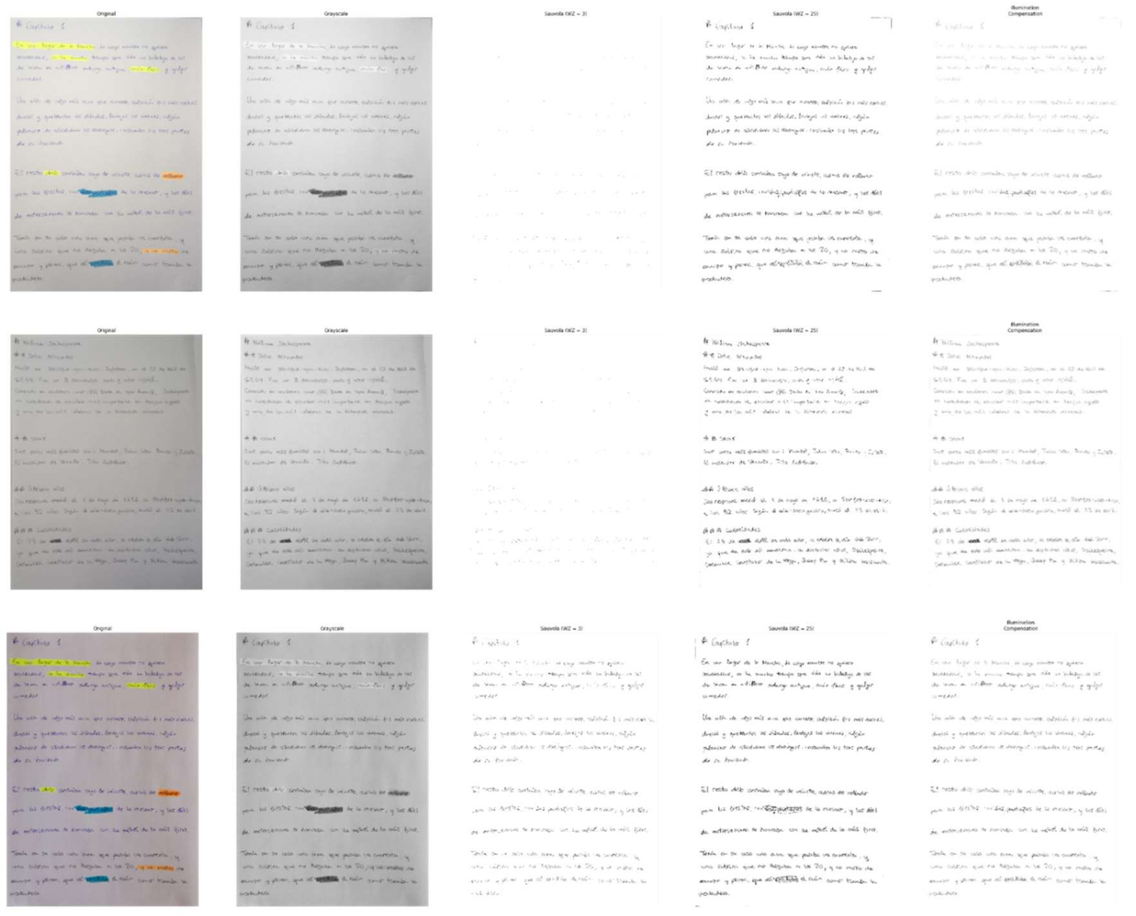


Figure 4.5: Comparison of binarization algorithms

Observe that while the Sauvola algorithm produces a correct output if window size value WZ is between $20 - 40$ (in Figure 4.5, $WZ = 25$), it does not completely remove the highlight frame in highlighted words, while *Illumination Compensation* does it. Be that as it may, as shown in rows 1 and 3, the result may vary significantly if the input image does not have brightness uniformly distributed. Finally, another reason to choose *Illumination Compensation* over Sauvola is that parameters have not to be tweaked for each input.

The *Illumination Compensation* algorithm is implemented using the default parameters as stated in (Chen et al. 2012).

4.2.2. Text Segmentation Module

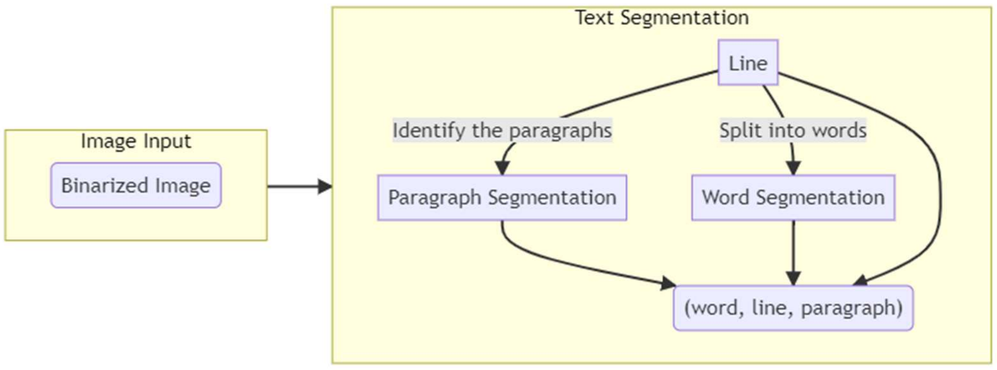


Figure 4.6: Flowchart of the Text Segmentation Module

A text segmentation module should enable the system to split a handwritten document into smaller blocks, like word or line blocks. This is useful for multiple reasons: first, it gives contextual information about each block, which is later used for error correction and sorting; second, it eases the text recognition, as the guessing algorithms have to work with fewer blocks and this can reduce potential faults.

Due to the need of the style recognition algorithm to work at word level, the text segmentation scheme must output word blocks. This could pose a setback on the error correction stage within the text recognition scheme, as the system loses contextual information from surrounding words. However, this can be solved by keeping the error correction stage outside of the text recognition scheme.

Multiple approaches have been tested for this module before deciding the final design. At first, a deep learning architecture, inspired by the findings from (Chung and Delteil 2019) have been chosen. However, its implementation and setup was cumbersome and did not meet the time constraints of this project. Therefore, a classic approach has been used to solve these pitfalls. This approach is described in the following sections.

4.2.2.1. Paragraph detection block

The goal of this block is to group neighboring lines that might form a paragraph; this knowledge is used later on in the system to provide contextual information to the spell checker.

To keep it simple and efficient, this block takes the output of the *Line segmentation block* and tries to group the lines by paragraphs on behalf of its Y coordinate. Therefore, this block uses a clustering algorithm that does not previously know the number of clusters, such as a hierarchical clustering algorithm or a kernel density estimation algorithm.

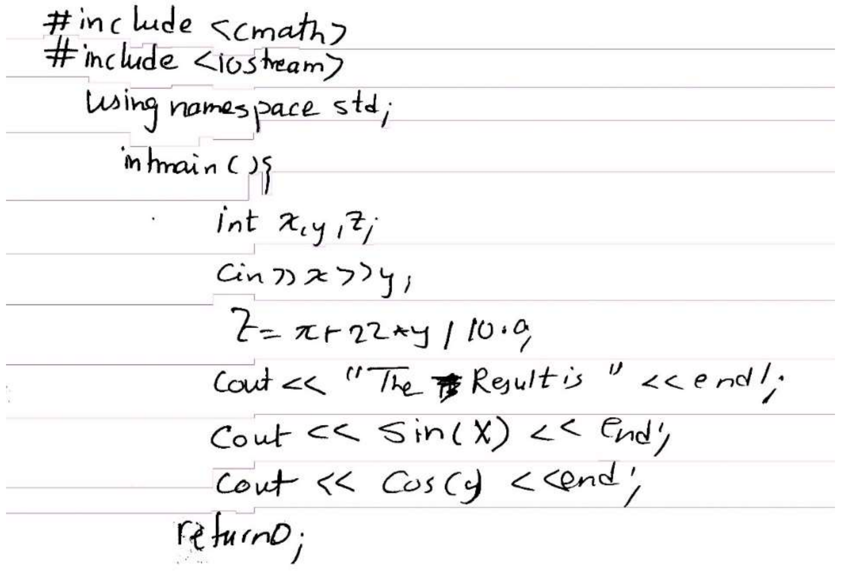
The Kernel Density Estimation algorithm has been selected as the clustering algorithm for this block. The OPTICS algorithm has also been tested, but it has a higher computational complexity. The kernel hyperparameter is set to *gaussian* and the *bandwidth* hyperparameter is set to 0.75.

4.2.2.2. Line segmentation block

Following the guidelines of creating segmentation algorithms with excellent processing speed and good accuracy, the line segmentation algorithm described in (Arivazhagan, Srinivasan, and Srihari 2007) is used. This algorithm is based on bivariate Gaussian densities to group characters and words to each line.

In detail, the possible lines are detected by using a piece-wise projection profile of the document. Then, each character or word is modeled as a connected component, so that the task is to assign each connected component to the line above or below it. In order to do this, the bivariate Gaussian density for each line is calculated, hence the connected component belongs to a line according to the probability of it under each Gaussian.

In Figure 4.7, a sample image from (Samir 2017) is shown, which is obtained after applying the line segmentation algorithm. In a similar manner to the image binarization block, qualitative testing is chosen as the validation testing method, as line segmentation model accuracy cannot be properly measured using bounding boxes.



```
#include <cmath>
#include <iostream>
  Using namespace std;
    intmain(){
          int x,y,z;
          Cin>>x>>y;
          z=x+22*y/10.0;
          Cout<< "The Resultis " <<endl;
          Cout << Sin(X) << End;
          Cout << Cos(y) <<end;
      return0;
```

Figure 4.7: Results from the line segmentation algorithm

4.2.2.3. Word segmentation block

The main problem exposed here is splitting up a group of characters separated from each other by a generous white space. Taking advantage of the fact that the input image is binarized, the vertical projection of the image is used to perform this task.

Namely, the vertical projection highlights the areas of the image where a continuous set of white pixels are grouped, which are modeled as white spaces. The target size for a white pixel group to be considered white space is computed from the mean of the group's sizes. Note, however, that header lines are also processed; these lines contain a big group at the end, which would unbalance the mean. Therefore, the last group is discarded.

In Figure 4.8, an image and its associated vertical projection is shown. Qualitative testing is chosen as the validation testing method for this block.

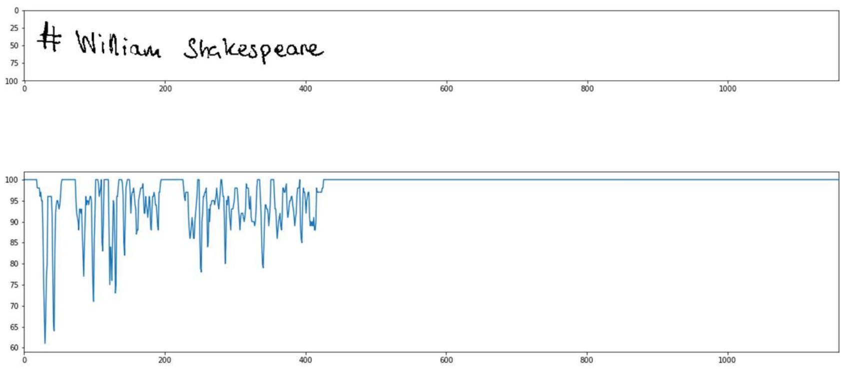


Figure 4.8: Vertical Projection for a text line

The implementation of the word detection block is straightforward, as the processing time is less than a second with the base Python performance. Firstly, the vertical projection of the line image is computed, by adding pixel values (which are either 0 or 1). Whitespace blocks are contiguous set of vertical lines where the sum of pixels equals the image height.

Then, the whitespace blocks are filtered out if their width is smaller than the average width. In the case that the processed image is a header line (which has a big trailing whitespace block at the end), the last block is discarded. Finally, words are split by using the filtered whitespace blocks as cut points. A minimum width of 11 lines is required for a whitespace block to be considered.

4.2.3. Data Processing and Data Augmentation Module

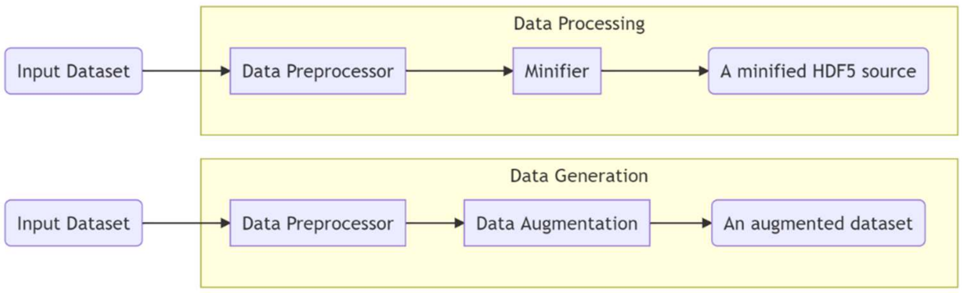


Figure 4.9: Flowchart of the Data Processing and Data Augmentation Module

This module is an utility module, which means that it has no required functionality and its methods are used across other modules, mostly in the color recognition and text recognition modules. On one side, the Data Processing block contains methods to deal with raw datasets and obtain a processed and ready-to-use dataset; on the other side, the Data Augmentation block contains methods to increase the size of a given dataset or to create new datasets.

This module was originally centered around the creation of the Spanish handwritten text dataset and has evolved since then in a more complex and purposeful module.

4.2.3.1. Spanish handwritten dataset

Most popular datasets related to the HTR problem can be classified into two classes:

- Modern text corpus: datasets tend to have more than 100 writers, text area blocks are evenly distributed throughout the document layout or structured document layouts, like form or letter layouts, are used.
- Historical text corpus: datasets have 2 or 3 writers at most, text area blocks are unevenly distributed throughout the document layout, character set height and width are not uniformly distributed (some letters have a much larger ratio than others, blurring line separation).

This project focuses on modern text corpora, and specifically, in modern Spanish text corpora. To this date, there are no modern Spanish text corpora – yet we can find some historical text corpora, such as (Fernández-Mota et al. 2014; Serrano, Castro, and Juan 2010). Moreover, as document layout is described by a markup language, there is an additional need for recognizing special ASCII characters, such as *@, #,* or *$.*

In Figure 4.10, the logarithmic character distribution for IAM and RIMES 2011 datasets is shown.

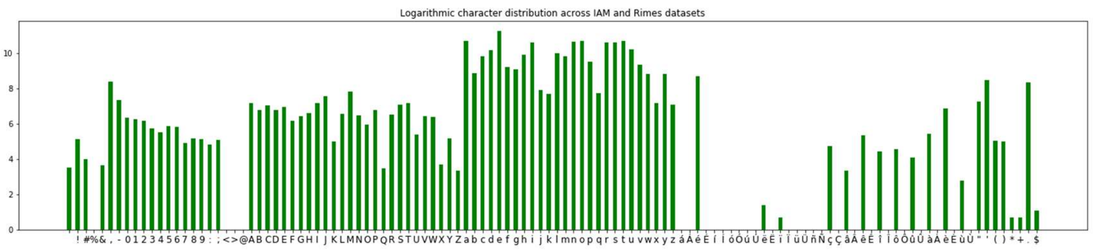


Figure 4.10: Logarithmic character distribution for IAM and RIMES 2011 datasets

Upon creating a new dataset, several conditions have been considered. Character distribution and word size ratios are one of the key factors that influence text recognition at later stages. This takes a major influence when resizing input images to fit the network constraints, thus some works (Chung and Delteil 2019) extend the resizing capabilities for words or lines to fit a certain zoom ratio.

Other approaches include using data generators, which fundamentally apply image processing techniques, like zooming, shearing or rotation to a given image in order to generate n variations of the same image. This simple approach can level up the number of samples for a small dataset. A sophisticated alternative is posed when using synthetic data generators via RNNs (Graves 2013) or GANs (Alonso et al. 2019). In Figure 4.11, a synthetic data sample, with yellow color font and blue background color, generated via RNN can be seen.

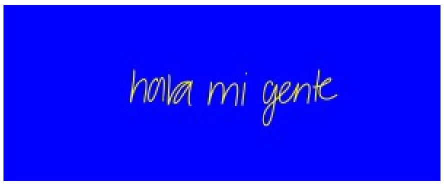


Figure 4.11: Synthetic data sample generated with RNNs

The final dataset is described hereunder, which has been created after weighting all of the explained options and gathering the experiment results.

The Spanish handwritten text dataset or *Spanish* dataset, in short, comprises handwritten notes from 15 writers. Besides, a subset of 2 writers also added full-page documents matching the document layout in IAM; this data is later used for testing the paragraph and word detection models. In Table 4.1, the mean word width and height for this dataset is compared to that of IAM and RIMES 2011.

Dataset	Average height	Average width
IAM	70	155.75
RIMES	72.18	186.14
Spanish	106.75	223.58

Table 4.1: Average image height and width in the datasets

Moreover, data is augmented by adding multiple grid paper backgrounds, as shown in Figure 4.12 . This is however discarded in later experiments, as it makes the model learning and color recognition unstable, but it is noted as feasible future work.

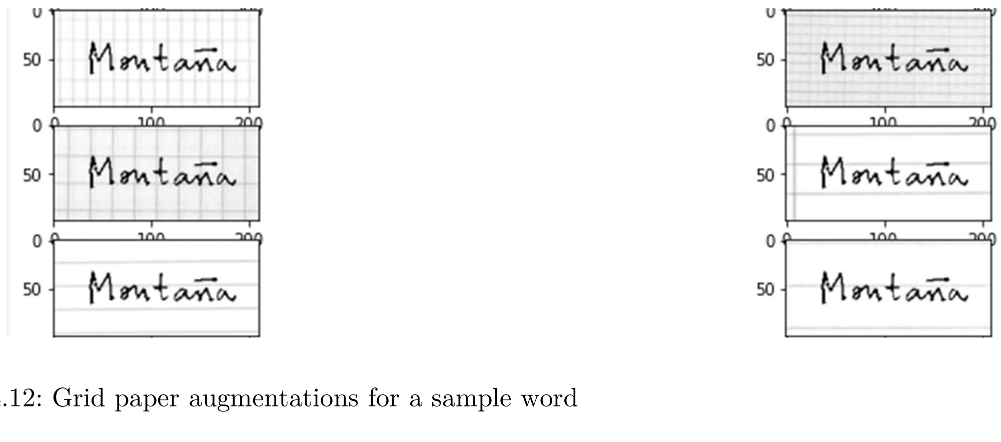


Figure 4.12: Grid paper augmentations for a sample word

Data is manually labeled using (Skalski 2012), and it is saved in two formats: as RGB images to be used in the color recognition block, and as grayscale images to be used in the text recognition block.

In Figure 4.13, the logarithmic character distribution for IAM and RIMES 2011 is shown. In Figure 4.14, this logarithmic character distribution also takes into account the Spanish dataset, where an improvement in Spanish and French characters can be observed, as well as in special characters.

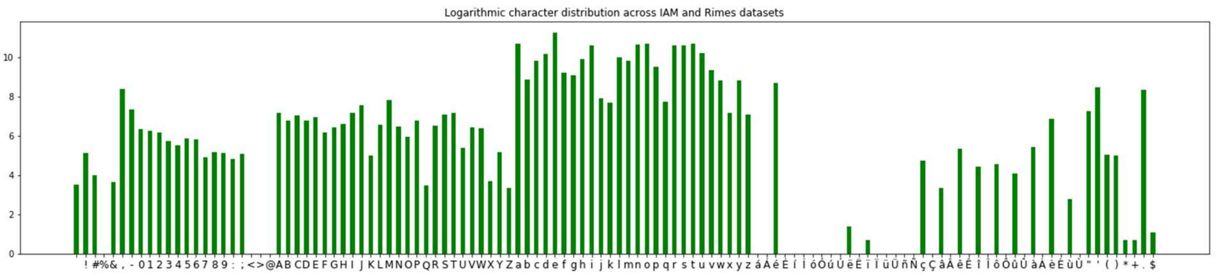


Figure 4.13: Logarithmic character distribution for IAM and RIMES 2011 datasets

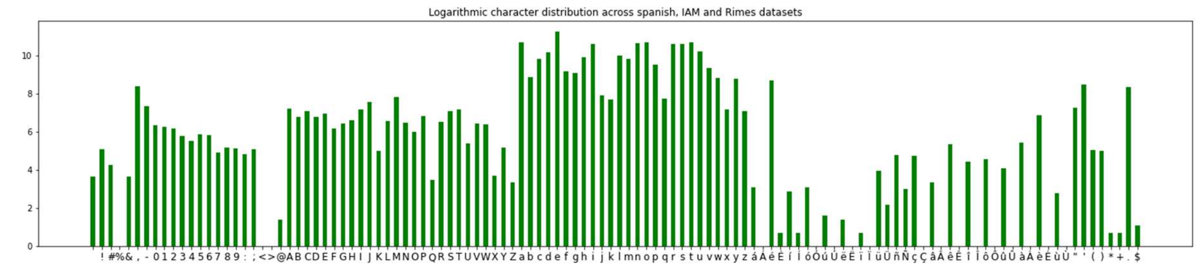


Figure 4.14: Logarithmic character distribution for IAM, RIMES 2011 and Spanish datasets

4.2.3.2. Data preparation

Input data is comprised of the IAM, RIMES 2011 and *Spanish* word datasets. Predefined indexes for selecting the training, validation and test subsets are used for IAM and RIMES 2011; these indexes are made available along with each dataset. In addition, the following augmentation techniques are applied to increase the training set size:

- Width shifting by a 0.2 range
- Height shifting by a 0.2 range
- Zooming by a 0.2 range
- Shearing by a 0.2 range

Finally, all images are normalized. In Figure 4.15, a data augmentation sample is shown.

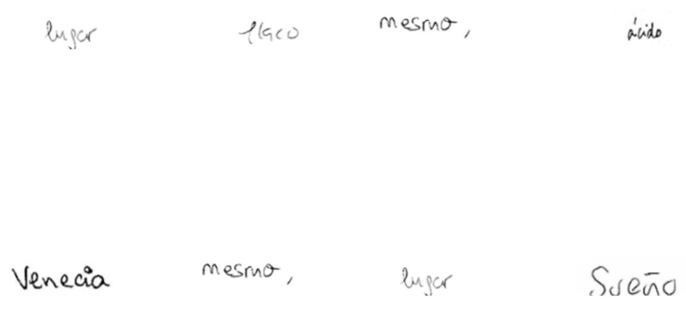


Figure 4.15: Data augmentation samples

In Figure 4.16, a comparison among an original word input and the set of augmented samples from it is shown.

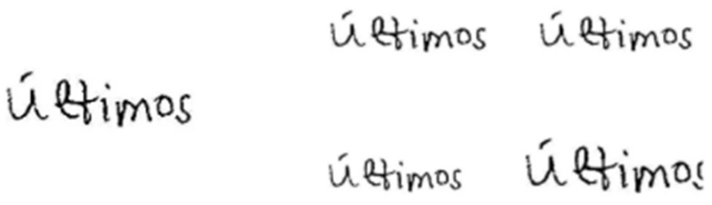


Figure 4.16: Comparison between the original input source and the augmented data samples

In order to unload data quickly in the Google Collab platform, input data is zipped into HDF5 file format, which reduces the total dataset size on disk by 53% and reduces the image load time by 92.5%. Note that this step is performed before data augmentation.

4.2.4. Color Recognition Module

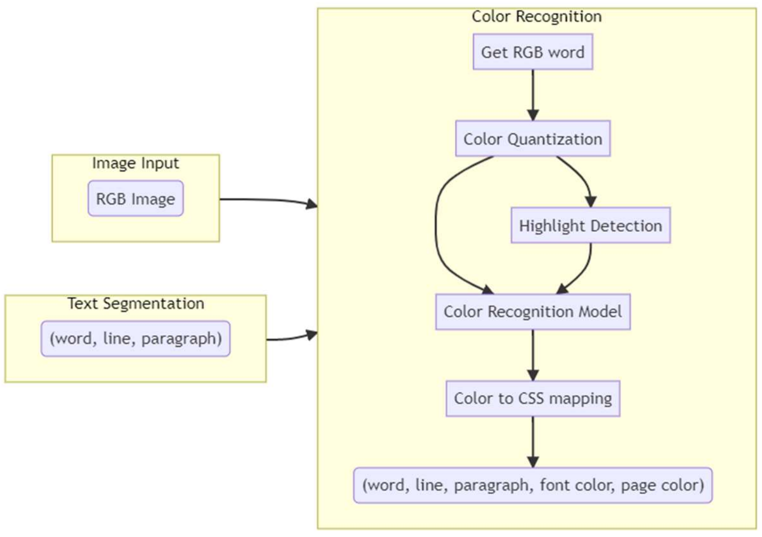


Figure 4.17: Flowchart of the Color Recognition Module

This module expects an unconstrained word image as input and produces a color tuple *(font-color, background-color)* as output.

To demonstrate the possibilities of recognizing handwritten, styled text, a subset of common text properties is selected for this work. Namely, font and background color, bold text, and emphasized text are text style properties detected by this system. However, bold text and emphasized text are modeled as supplementary characters added to such text, just like it is done in markup languages. Therefore, these properties are considered in the *Language Processing* module.

In order to recognize the font and background color of a word, image processing techniques are used. The way it is done in this project is by setting some assumptions. Firstly, it is assumed that the default font color is black, and the default background color is white. Thus, the background color only changes if there is a highlighted word. Secondly, all colors are grouped into a smaller set of colors, so as to use the CSS3 named colors specification (Çelik, Lilley, and Baron 2011).

To provide a test dataset for this module, the handwritten Spanish dataset samples are also saved in RGB format, and at least half of the dataset are highlighted words.

Besides, a simple NN-based model, composed of 2 convolutional layers and 2 feed-forward layers, is used to detect if a word is highlighted or not. Its scheme is detailed in Figure 4.18. Categorical cross-entropy is used to compute the network loss. Input images are resized to 150×150 to speed up the network training and inference.

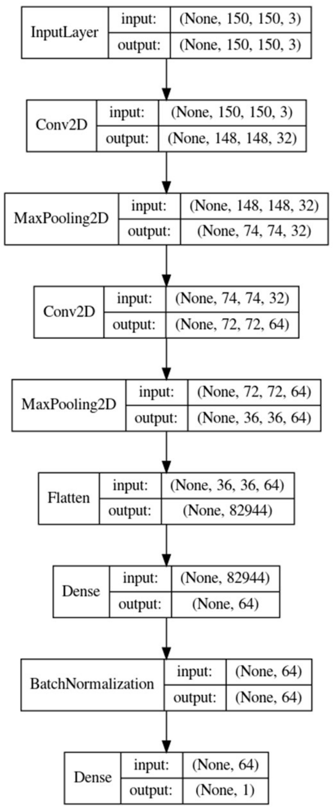

InputLayer
input: (None, 150, 150, 3)
output: (None, 150, 150, 3)
Conv2D
input: (None, 150, 150, 3)
output: (None, 148, 148, 32)
MaxPooling2D
input: (None, 148, 148, 32)
output: (None, 74, 74, 32)
Conv2D
input: (None, 74, 74, 32)
output: (None, 72, 72, 64)
MaxPooling2D
input: (None, 72, 72, 64)
output: (None, 36, 36, 64)
Flatten
input: (None, 36, 36, 64)
output: (None, 82944)
Dense
input: (None, 82944)
output: (None, 64)
BatchNormalization
input: (None, 64)
output: (None, 64)
Dense
input: (None, 64)
output: (None, 1)

Figure 4.18: Highlighted word recognition model architecture

The proposed network architecture is implemented with Keras. A subset of the original dataset which is class-balanced is used, with 277 samples on each class. Images are pre-processed with a custom function and are later normalized. In Figure 4.19 and Figure 4.20, two samples of the dataset are shown.

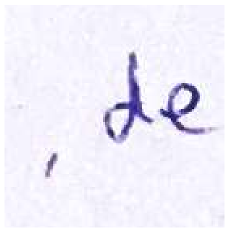


Figure 4.19: Dataset sample with colored font

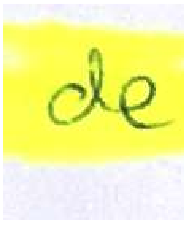


Figure 4.20: Dataset sample with colored background

The custom pre-processing function performs the following steps:

1. Brightness value is incremented by 20, gamma value is incremented by 0.05, and contrast is adjusted by 150%. This ensures that the word background is white and that the highlighting features are not turned into shades of white.
2. A color quantization algorithm, implemented with K-means, reduces the color dimensions, and produces a faster and more stable inference. $K = \{3,4\}$ values have been tested and $K = 4$ seems to deliver more consistent results, as the background stays white even for highlighted words.

An alternative step is to choose the darkest RGB channel, as background stays white and the darkest channel is the one holding the highlighted color.

In Figure 4.21 and Figure 4.23, two original samples are shown; in Figure 4.22 and Figure 4.24, respectively, the same samples after running the preprocessing step are shown. Qualitative improvements can be observed, as color representations for highlighted words are more consistent when quantization is applied.

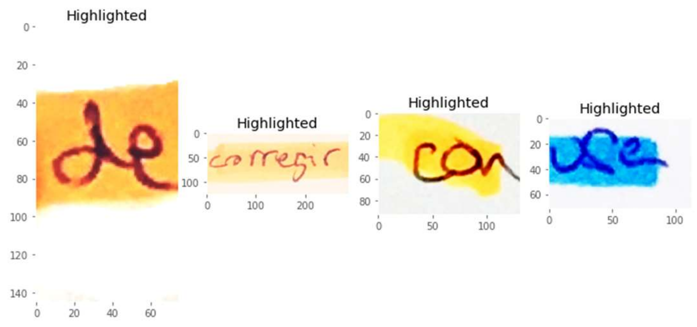


Figure 4.21: Sample from highlighted words dataset

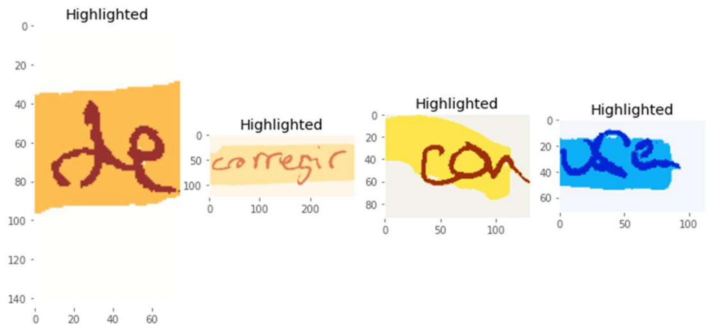


Figure 4.22: Sample from highlighted words dataset after preprocessing

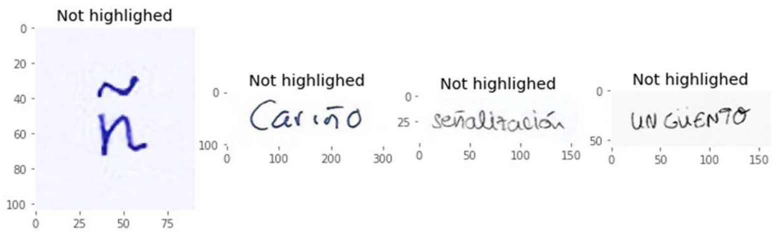


Figure 4.23: Sample from highlighted words dataset

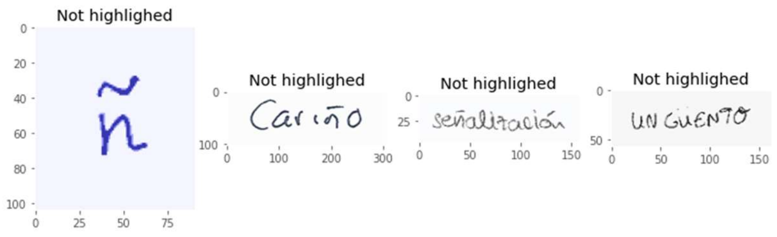


Figure 4.24: Sample from highlighted words dataset after preprocessing

In Figure 4.25, a comparison between K values is shown.

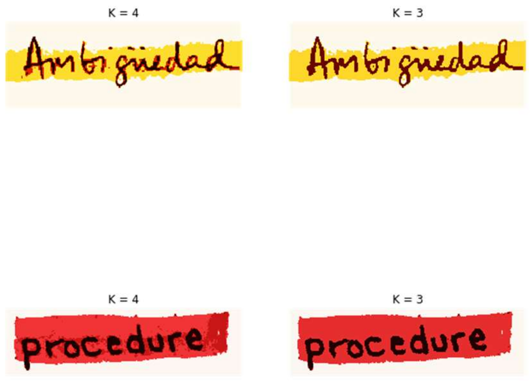


Figure 4.25: Comparison between different K value for color quantization

The recognition model uses categorical cross-entropy as loss function and the Adam optimizer with the parameters from (Kingma and Ba 2015). The model obtains a testing accuracy of 95%, so a further improvement and minification of the model is done.

The following improvements are shown in Figure 4.26: convolutional layers are replaced with depth-wise spatial convolutional layers, which reduce network complexity at the cost of higher training time; and feed-forward layers are shrunk. This network now obtains a testing accuracy of 99.28% with 90% less parameters.

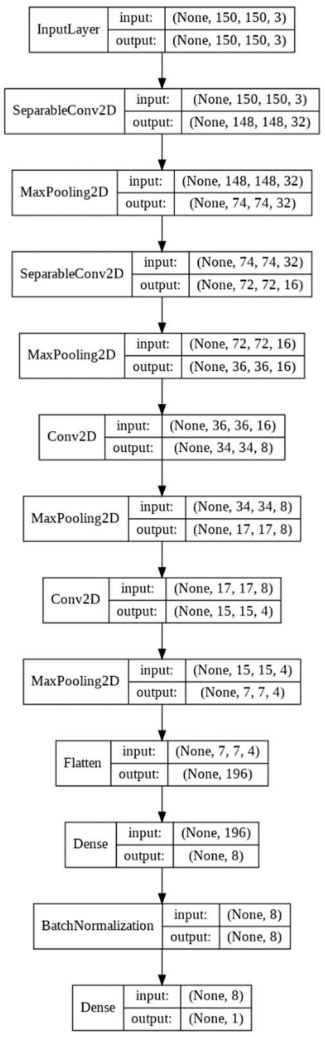
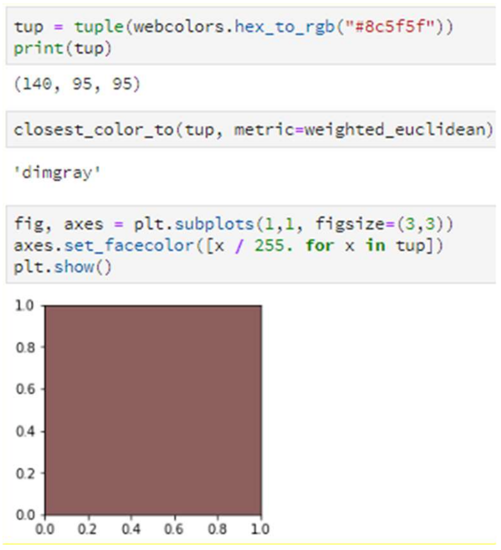


Figure 4.26: Final highlighting recognition network

4.2.4.1. Color classification block

As a means of making easier the recognition and classification of colors, the results from the XKCD color survey (Munroe 2010) are used to group them. The *Modified Median Cut* algorithm as presented by (Bloomberg 2008) is used to recognize the main 3 colors: font color, background color, and highlight color (if any). Finally, a K-Means algorithm is used as a pre-processing step for reducing the color space of each sample to 3 colors.

At first, a simpler approach using Euclidean distance in the RGB and HSV color spaces is used to group colors. However, some shades of the color classes are closer to white, black, or gray, as the image acquisition system and brightness conditions when taking the photograph greatly influence the result at this stage. In Figure 4.27, a red patch is recognized as gray if this method is used.



```
tup = tuple(webcolors.hex_to_rgb("#8c5f5f"))
print(tup)
```

```
(140, 95, 95)
```

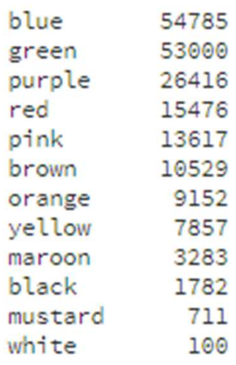
```
closest_color_to(tup, metric=weighted_euclidean)
```

```
'dimgray'
```

```
fig, axes = plt.subplots(1,1, figsize=(3,3))
axes.set_facecolor([x / 255. for x in tup])
plt.show()
```

Figure 4.27: A shade of red is identified as gray

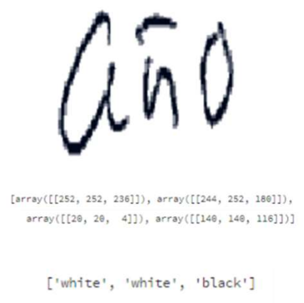
A more reliable approach is to use the results from the XKCD color survey (Munroe 2010), which contains over 5 million RGB color labels tagged by more than 200,000 people; each label is mapped to one of the 12 color classes. In Figure 4.28, the label distribution per color class is shown. A K-Nearest-Neighbors clustering model with $K = 50$ is used to predict a class for new color tuples.



```
blue       54785
green      53000
purple     26416
red        15476
pink       13617
brown      10529
orange      9152
yellow      7857
maroon      3283
black       1782
mustard      711
white        100
```

Figure 4.28: Label distribution per color class

For extracting the 3 main colors from a word image, a Modified Median Cut Quantization algorithm is implemented. Note that the previous image quantization at the pre-processing stage helps to speed up this stage as well. This is implemented using *ColorThief* (Dhakar 2017), and a *Numba* (Anaconda 2018) backend is used to obtain speed ratios close to a C implementation. In Figure 4.29, the extracted color palette for a word sample is shown.



```
[array([[252, 252, 236]]), array([[244, 252, 180]]),
   array([[20, 20,  4]]), array([[140, 140, 116]])]
```

```
['white', 'white', 'black']
```

Figure 4.29: Color palette for a word image

4.2.5. Text Recognition Module

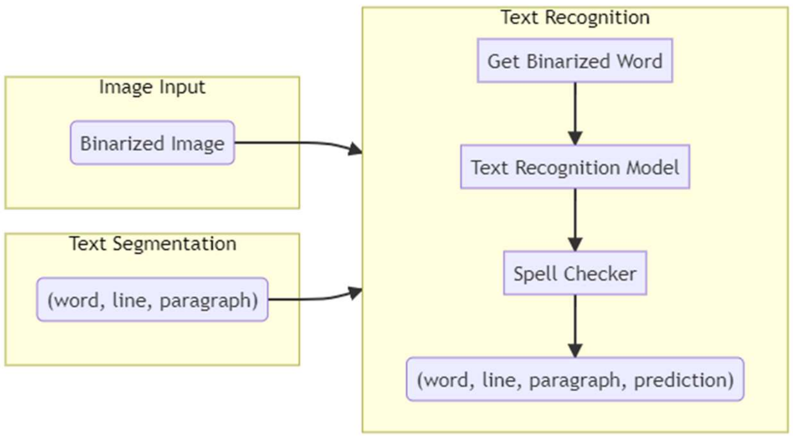


Figure 4.30: Flowchart of the Text Recognition Module

The grayscale word *getter* block is quite simple, as it already receives all the required information from the other modules. This block is designed in plain Python. Likewise, the spell checker block extends the *pyspellchecker* Python interface, which already provides ready-to-use configurations for processing sentences instead of words. Moreover, the Spanish, French and English dictionaries from (LibreOffice n.d.) have been added to the default dictionaries.

As for the text recognition model, most HTR models are based on the seq2seq problem and use the CTC function for computing both the loss and prediction. The latest models are using either modifications of the CNN base layer to speed up feature dimensionality reduction or new approaches for the text prediction. Usually, these systems are evaluated with the following metrics:

- Validation loss, which is a common metric to all deep learning problems.
- Character Error Rate (CER), or the Levenshtein distance between a predicted label and a true label.

- Word Error Rate (WER), which is the weighted difference between two set of words, described by the formula

  $$WER = \frac{S + D + I}{S + D + C}$$

  Being:

  - S the number of substitutions
  - D the number of deletions
  - I the number of insertions
  - C the number of correct words

Following these guidelines, the proposed architecture for the text recognition model is inspired by (Bluche and Messina 2017; Chung and Delteil 2019; Puigcerver 2017; Scheidl, Fiel Wien, and Scheidl Robert Sablatnig 2011) and performs multiple optimizations to allow faster training and inference, while offering comparable performance. Namely, the following modifications are introduced:

- Recognition is reduced to word level, while keeping contextual line information outside of the network for character error correction.
- A spelling corrector block inspired by (Norvig 2007) is used to seize contextual information from lines. This design is based on a language detection module that can identify if the text is in Spanish, French or English, and a spelling correction module that, given a sentence, checks if it has any incorrect spelling by looking it up in a dictionary, and returns the most probably correction based on the sentence context.

Moreover, the following configuration fields are tested with multiple values to find the most fitting setup:

- Apply 8-bit quantization or not, to reduce the time complexity in the inference stage.
- Sample down the input image size between a 25% and 50% factor, to reduce the time complexity in the training stage.
- Use three types of convolutional layers: a common 2D convolutional layer, a gated 2D convolutional layer (Lin et al. 2019) and an octave 2D convolutional layer (Chen et al. 2019).
- Other common hyperparameters such as batch size, kernel size, learning rate or number of filters in the convolutional layers.

In Figure 4.31, the proposed architecture that uses gated convolutional layers is shown.

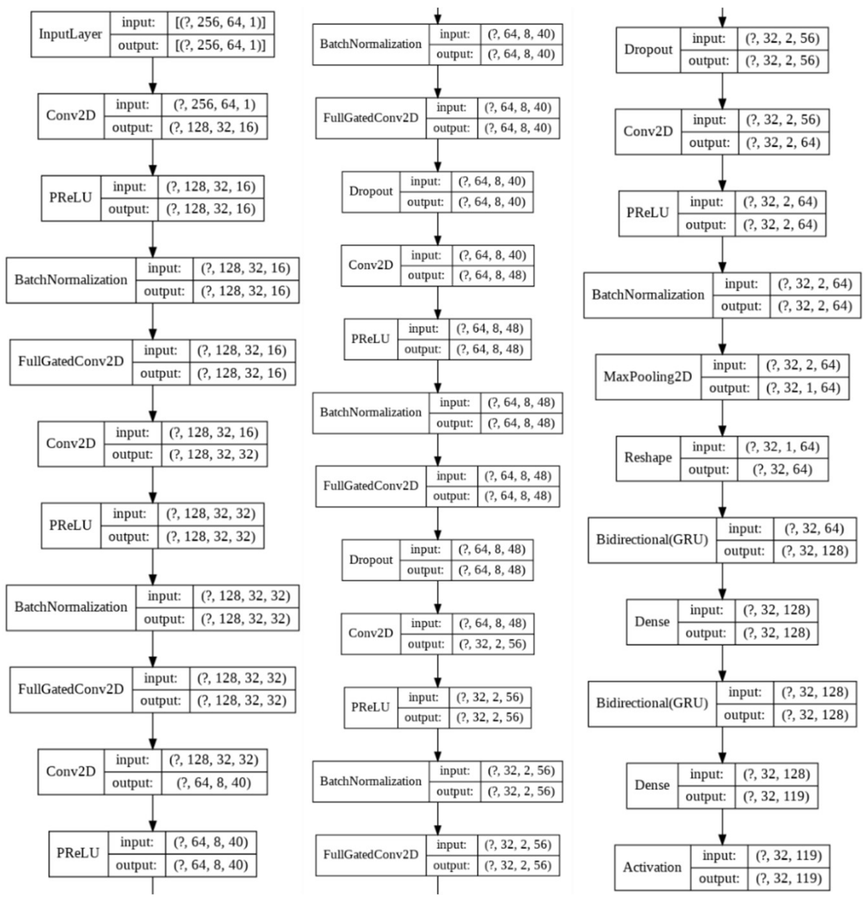
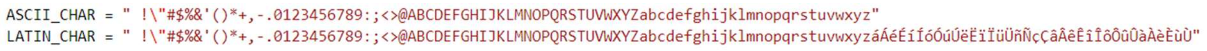


Figure 4.31: Proposed scheme for handwriting text recognition model

The proposed architecture is implemented with TensorFlow 2.0. As explained previously, this one uses the CTC loss function and CTC prediction function. The RMSProp function with a learning rate of 5^{-4} is used as the network optimizer.

As for the network internal setup, a default character set comprising default ASCII characters plus Spanish and French special vowels is used. In Figure 4.32, a comparison between the ASCII and the extended Latin character set used for this network is shown; consider that longer character sets pose a limitation to the network, as the character probability is smaller. The maximum allowed output word length is dependent on the input size, as the network with smaller inputs produces a more consistent result with smaller word lengths. This is due to the fact that information is more compressed on smaller image sizes, and thus is harder for the network to obtain all the character details. The default output word length is 34 and the default input size is 64×256.



```
ASCII_CHAR = " !\"#$%&'()*+,-.0123456789:;<>@ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
LATIN_CHAR = " !\"#$%&'()*+,-.0123456789:;<>@ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzáÁéÉíÍóÓúÚëËïÏüÜñÑçÇâÂêÊîÎôÔûÛàÀèÈùÙ"
```

Figure 4.32: ASCII and Extended Latin character sets

Labels are pre-processed as well, mapping each string to an integer vector of the same length of the output word length, where each integer corresponds to the character position in the character set. The epoch limit is set to 20 and the batch size is set to 128.

The spell checker is implemented by extending the *pyspellchecker* Python interface.

4.2.6. Language Processor Module

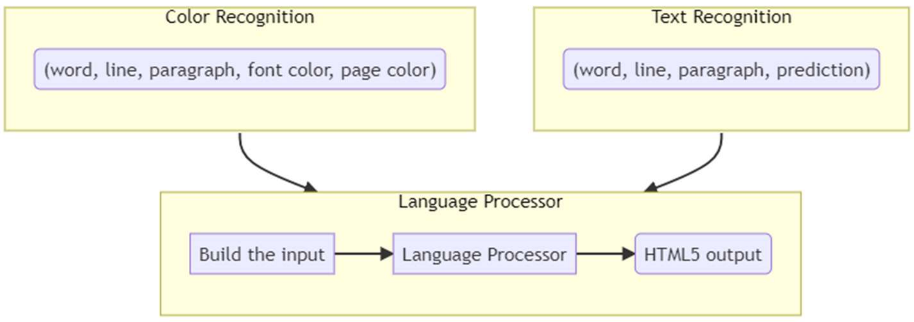


Figure 4.33: Flowchart of the Language Processor module

As far as the consulted sources go, there is no previous paper on the use of compiler tools to address the switchover of document layout to digital formats. For this reason and to simplify the process at this stage, only standard compiler tools are used.

In short, compiler tools usually translate from a given programming language to either native or intermediate code, or another programming language – in the latter case, this is called transpiler. Generally, a compiler performs the following stages: pre-processing, lexical analysis, syntax analysis, semantic analysis, and conversion to the target code. Additional stages such as code optimization are optional.

Popular compiler tools are (Adve and Lattner 2003; Johnson 1975; Parr and Quong 1995). However, as ANTLR4 is the only one that provides a language-agnostic platform, along with a shorter learning curve and simpler interoperability with our Python environment, it is the tool used in this project.

The expected input for this module is a set of tuples

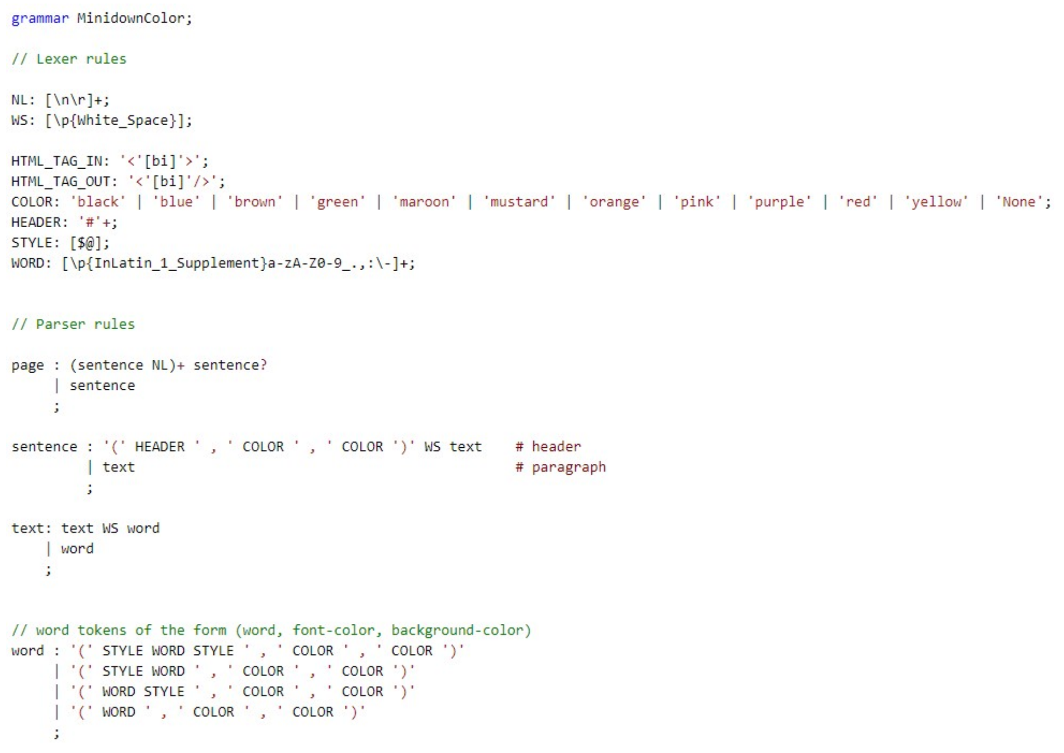
$$(\text{word}, \text{line}, \text{paragraph}, \text{font color}, \text{background color})$$

which is processed to generate HTML5 code. At a later stage, the target code can be exported to other formats using Pandoc (MacFarlane 2006), such as *.pdf* or *.docx*. By default, $font\ color$ is set to black and $background\ color$ is set to white.

For the sake of keeping design minimal, a generic language, *MinidownColor,* has been created. It provides paragraph and header features from *Markdown* and adds the option to use color on text.

In Figure 4.34, the lexer and parser specifications for *Minidown* are provided. Note the following:

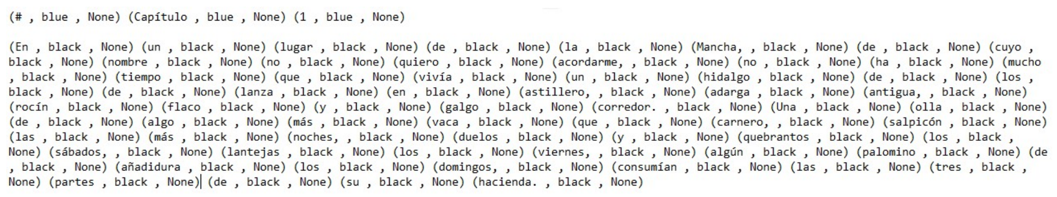
- $\backslash p\{cat\}$ refers to Unicode property escapes, as referred in (MDN Contributors 2020); namely, $\backslash p\{White\_Space\}$ which links to Unicode whitespace characters, and $\backslash p\{InLatin1\_Supplement\}$ which links to the ISO/IEC 8859-1 character set.
- The *COLOR* lexer rule accepts 11 main colors and *None*, which is equivalent to white.
- The *word* parser rule accepts a *STYLE* token around text; this allows us to use *$*, *@* characters to give bold and emphasis styles, respectively. For example, $hello$ is compiled as **hello.**



```
grammar MinidownColor;

// Lexer rules

NL: [\n\r]+;
WS: [\p{White_Space}];

HTML_TAG_IN: '<'[bi]'>';
HTML_TAG_OUT: '<'[bi]'/>';
COLOR: 'black' | 'blue' | 'brown' | 'green' | 'maroon' | 'mustard' | 'orange' | 'pink' | 'purple' | 'red' | 'yellow' | 'None';
HEADER: '#'+;
STYLE: [$@];
WORD: [\p{InLatin_1_Supplement}a-zA-Z0-9_.,:\-]+;


// Parser rules

page : (sentence NL)+ sentence?
     | sentence
     ;

sentence : '(' HEADER ' , ' COLOR ' , ' COLOR ')' WS text    # header
         | text                                             # paragraph
         ;

text: text WS word
    | word
    ;


// word tokens of the form (word, font-color, background-color)
word : '(' STYLE WORD STYLE ' , ' COLOR ' , ' COLOR ')'
     | '(' STYLE WORD ' , ' COLOR ' , ' COLOR ')'
     | '(' WORD STYLE ' , ' COLOR ' , ' COLOR ')'
     | '(' WORD ' , ' COLOR ' , ' COLOR ')'
     ;
```

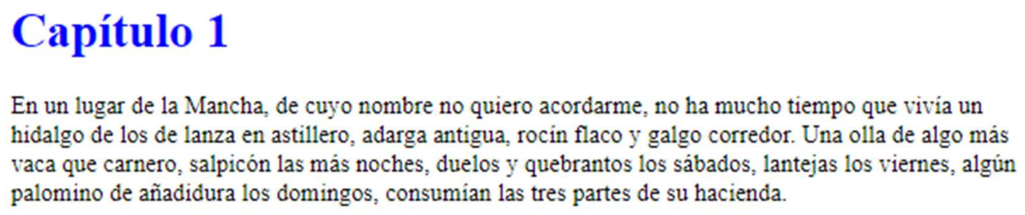
Figure 4.34: Lexer and Grammar specification for MinidownColor

In Figure 4.35, the expected input for the current module is shown. In Figure 4.36, the desired output for the current module is shown.



```
(# , blue , None) (Capítulo , blue , None) (1 , blue , None)

(En , black , None) (un , black , None) (lugar , black , None) (de , black , None) (la , black , None) (Mancha, , black , None) (de , black , None) (cuyo ,
black , None) (nombre , black , None) (no , black , None) (quiero , black , None) (acordarme, , black , None) (no , black , None) (ha , black , None) (mucho
, black , None) (tiempo , black , None) (que , black , None) (vivía , black , None) (un , black , None) (hidalgo , black , None) (de , black , None) (los ,
black , None) (de , black , None) (lanza , black , None) (en , black , None) (astillero, , black , None) (adarga , black , None) (antigua, , black , None)
(rocín , black , None) (flaco , black , None) (y , black , None) (galgo , black , None) (corredor. , black , None) (Una , black , None) (olla , black , None)
(de , black , None) (algo , black , None) (más , black , None) (vaca , black , None) (que , black , None) (carnero, , black , None) (salpicón , black , None)
(las , black , None) (más , black , None) (noches, , black , None) (duelos , black , None) (y , black , None) (quebrantos , black , None) (los , black ,
None) (sábados, , black , None) (lantejas , black , None) (los , black , None) (viernes, , black , None) (algún , black , None) (palomino , black , None) (de
, black , None) (añadidura , black , None) (los , black , None) (domingos, , black , None) (consumían , black , None) (las , black , None) (tres , black ,
None) (partes , black , None) (de , black , None) (su , black , None) (hacienda. , black , None)
```

Figure 4.35: Expected input for the language processing module



Capítulo 1

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.

Figure 4.36: Desired output for the language processing module

In addition, a pre-processor is required to merge the outputs from the Color Recognition and the Text Recognition modules. Note that background and highlighting color are the same at this stage, as the default page color is white.

The development of this module is quite straightforward, once given the lexer and parser specification. In order to keep the specification language-agnostic, ANTLR provides two types of interfaces to traverse the AST tree: the Listener, which simply is activated when a the program enters and exits a tree node; and the Visitor, which extends the Listener by allowing to define a custom tree visiting method. These interfaces can be implemented in multiple programming languages, such as C, Java or Python.

For the sake of simplicity, only the Listener interface is implemented in Python. This way, tags can be directly matched to their equivalent tree nodes. For example, the $< h1 >$ HTML5 tag is written when entering the *Header* node, and the $</h1 >$ tag is written when exiting it. Additionally, in order to keep the font and background color, the default HTML5 colors are used (black for the font color and white for the background color). If a word uses a different color, then it is wrapped inside a $< span >$ tag which uses the $style$ attribute to assign the color.

Even though this is not the most efficient method to build up large documents, it shows the possibilities of keeping color word by word when recognizing text from images. Moreover, it also exposes the possibility of using CSS3 stylesheets for HTML5 output format to define custom color names, header or paragraph styles.

Chapter 5. **Results**

5.1. Tests descriptions and results

5.1.1. Input Image module tests

The following tests are used to ensure the functionality is fulfilled:

- T1.1: Given a RGB image, return a pair of images, one in RGB mode and another in grayscale mode.
- T1.2: Given a grayscale image, return a binarized image, using the Sauvola algorithm. The resulting loss of information must be lower than 30%.
- T1.3: Given a grayscale image, return a binarized image, using the Illumination Compensation algorithm. The resulting loss of information must be lower than 15%.

Moreover, the following tests are used to ensure that a comparable performance is obtained:

- T1.4: Given a grayscale image, return a binarized image, using either the Sauvola or Illumination Compensation algorithm implemented in Numba. The resulting time complexity must be 50% less than the time complexity when the algorithm is implemented in native Python.

Test number	*Has passed*	*Observations*
1	Yes	-
2	Yes	The loss of information is in between $15 \dots 25\%$
3	Yes	The loss of information is in between $3 \dots 10\%$.
4	Yes	The performance gain ratio is always greater than 100

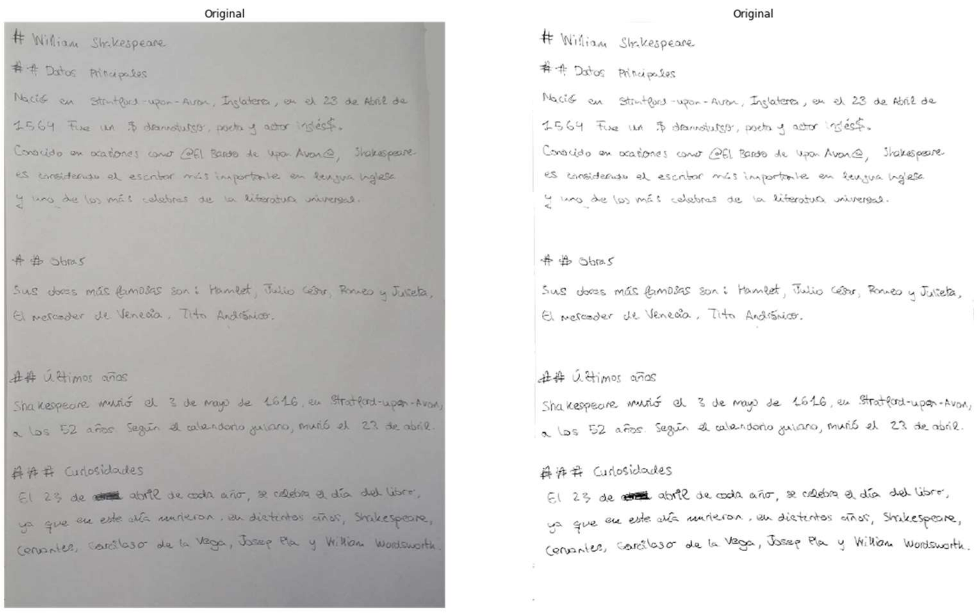
Table 5.1: Test results for the Image Input Module

In addition, in order to optimize the *Illumination Compensation* algorithm, a custom Python solution is used to speed up processing: the *numba* Python package is used to compile down Python code to C code. In Table 5.2, a comparison between default and *Numba* implementations is shown. All images are assumed to be already grayscale.

Image size	*Default performance*	*Numba performance*	*Performance gain ratio*
164×588	2.45s ± 331ms	16.8ms ± 720µs	$x145.83$
209×2104	6.4s ± 227ms	57.9ms ± 1.31ms	$x110.54$
1542×1158	30.6s ± 778ms	303ms ± 31.1ms	$x101$

Table 5.2: Processing speeds improvements in the text segmentation module

Finally, the implementation is tested against a set of RGB sample images to check if some preprocessing steps are needed. Surprisingly, it shows a robust behavior even under low light conditions. In Figure 5.1, a sample input image is processed.


Original
Original

William Shakespeare
Datos Principales
Nació en Stratford-upon-Avon, Inglaterra, en el 23 de Abril de
1564 Fue un $ dramaturgo, poeta y actor inglés$.
Conocido en ocasiones como @El Bardo de Upon Avon@, Shakespeare
es considerado el escritor más importante en lengua inglesa
y uno de los más celebres de la literatura universal.

Obras
Sus obras más famosas son: Hamlet, Julio César, Romeo y Julieta,
El mercader de Venecia, Tito Andrónico.

Últimos años
Shakespeare murió el 3 de mayo de 1616, en Stratford-upon-Avon,
a los 52 años. Según el calendario juliano, murió el 23 de abril.

Curiosidades
El 23 de abril de cada año, se celebra el día del libro,
ya que en este día murieron, en distintos años, Shakespeare,
Cervantes, Garcilaso de la Vega, Josep Pla y William Wordsworth.

William Shakespeare
Datos Principales
Nació en Stratford-upon-Avon, Inglaterra, en el 23 de Abril de
1564 Fue un $ dramaturgo, poeta y actor inglés$.
Conocido en ocasiones como @El Bardo de Upon Avon@, Shakespeare
es considerado el escritor más importante en lengua inglesa
y uno de los más celebres de la literatura universal.
Obras
Sus obras más famosas son: Hamlet, Julio César, Romeo y Julieta,
El mercader de Venecia, Tito Andrónico.
Últimos años
Shakespeare murió el 3 de mayo de 1616, en Stratford-upon-Avon,
a los 52 años. Según el calendario juliano, murió el 23 de abril.
Curiosidades
El 23 de abril de cada año, se celebra el día del libro,
ya que en este día murieron, en distintos años, Shakespeare,
Cervantes, Garcilaso de la Vega, Josep Pla y William Wordsworth.

Figure 5.1: Image binarization without pre-processing

5.1.2. Text Segmentation module tests

The following tests are used to ensure that the functionality is fulfilled:

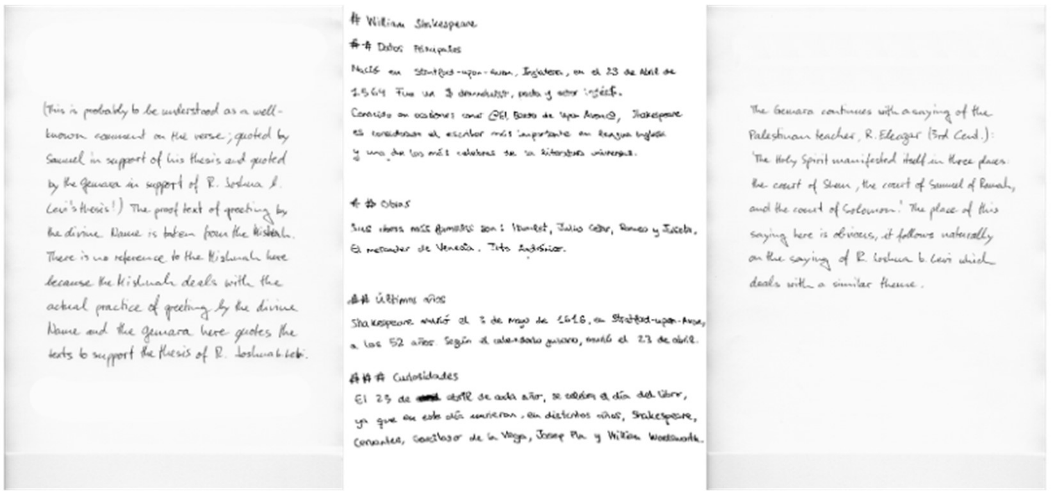
- T2.1: Given a binarized image, return a list of lines; its length must be equal to the expected number of lines, and the loss of information among each predicted line to its expected line must be lower than 25%.
- T2.2: Given a list of lines, return a list of lists of words; each list of words must match each inputted line, the number of words for each line must match the expected number of words, and the loss of information among each predicted word to its expected word must be lower than 25%.
- T2.3: Given a list of lines, return a singleton list of paragraphs; the number of paragraphs must be 1 and the number of expected lines must be equal to the the number of real lines.

- T2.4: Given an image with text content, return a nested list containing a list of paragraphs, each a list of lines and each a list of words. Each word must match the expected line and the loss of information for each predicted line with respect to its expected paragraph must not be greater than a 25%.

Test number	*Has passed*	*Observations*
1	Yes	-
2	Yes	-
3	Yes	-
4	Yes	*[1]

Table 5.3: Test results for the Text Segmentation module

*[1]: The algorithm has a hard time dealing with paragraphs whose distance among them is small, and with lines whose distance among them is small. Sometimes, this is worsened by the zooming of the original image. In Figure 5.2, a sample of input text images with paragraphs which are correctly recognized is shown. In Figure 5.3, a sample of input text images with paragraphs that are wrongly recognized is shown. Notice the separation among paragraphs and the separation among lines in each sample.



(This is probably to be understood as a well-known comment on the verse; quoted by Samuel in support of his thesis and quoted by the Gemara in support of R. Joshua b. Levi's thesis!) The proof text of greeting by the divine Name is taken from the Mishnah. There is no reference to the Mishnah here because the Mishnah deals with the actual practice of greeting by the divine Name and the Gemara here quotes the texts to support the thesis of R. Joshua b. Levi.

William Shakespeare
Datos Principales
Nació en Stratford-upon-Avon, Inglaterra, en el 23 de Abril de 1564. Fue un dramaturgo, poeta y actor inglés.
Conocido en ocasiones como @El Bardo de Avon@, Shakespeare es considerado el escritor más importante en lengua inglesa y uno de los más célebres de la literatura universal.

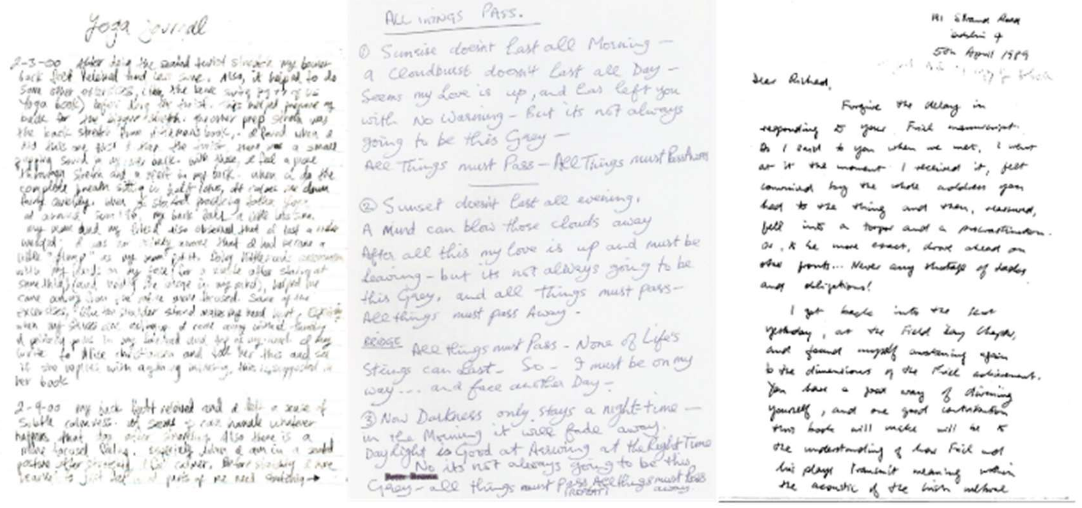
Obras
Sus obras más famosas son: Hamlet, Julio César, Romeo y Julieta, El mercader de Venecia, Tito Andrónico.

Últimos años
Shakespeare murió el 3 de mayo de 1616, en Stratford-upon-Avon, a los 52 años. Según el calendario juliano, murió el 23 de abril.

Curiosidades
El 23 de [illegible] abril de cada año, se celebra el día del libro, ya que en este día murieron, en distintos años, Shakespeare, Cervantes, Garcilaso de la Vega, Josep Pla y William Wordsworth.

The Gemara continues with a saying of the Palestinian teacher, R. Eleazar (3rd Cent.): 'The Holy Spirit manifested itself in three places: the court of Shem, the court of Samuel of Ramah, and the court of Solomon.' The place of this saying here is obvious, it follows naturally on the saying of R. Joshua b. Levi which deals with a similar theme.

Figure 5.2: Paragraphs rightly segmented by paragraphs and lines



Yoga Journal

2-3-00 After doing the seated twist stretch my lower back felt relaxed and less sore. Also, it helped to do some other exercises, (like the knee swing pg [illegible] of the Yoga book) before doing the twist. This helped prepare my back for the bigger stretch. Another prep stretch was the back stretch from [illegible] book. [illegible] her book

2-4-00 My back felt relaxed and I felt a sense of subtle calmness. [illegible]

ALL THINGS PASS.

① Sunrise doesn't last all Morning –
a Cloudburst doesn't last all Day –
Seems my Love is up, and has left you
with No Warning – But its not always
going to be this Grey –
All Things must Pass – All Things must Pass away

② Sunset doesn't last all evening,
A Mind can blow those clouds away
After all this my Love is up and must be
leaving – but its not always going to be
this Grey, and all things must pass –
All things must pass Away –

BRIDGE All things must Pass – None of Life's
Strings can Last – So – I must be on my
way... and face another Day –

③ Now Darkness only stays a night-time –
in the Morning it will fade away.
Daylight is Good at Arriving at the Right Time
No its not always going to be this
Grey – all things must Pass All things must pass away
(REPEAT)

141 Strand Road
Dublin 4
5th April 1989

Dear Richard,

Forgive the delay in responding to your Friel manuscript. As I said to you when we met, I went at it the moment I received it, felt convinced by the whole ambition you had to [illegible] and then, [illegible], fell into a torpor and a procrastination. Or, to be more exact, drove ahead on other fronts... Never any shortage of duties and obligations!

I got back into the text yesterday, at the Field Day Chapter, and found myself awakening again to the dimensions of the Friel achievement. You have a great way of [illegible] yourself, and one good contribution this book will make will be to the understanding of how Friel and his plays transmit meaning within the acoustic of the Irish [illegible]

Figure 5.3: Paragraphs wrongly segmented by paragraphs and lines

In addition, in order to reduce the time complexity of the line segmentation algorithm, it has been implemented using the *numba* package. In Table 5.4, a comparison between default and *Numba* implementations is shown. All images are assumed to be already binary.

Image size	*Default performance*	*Numba performance*	*Performance gain ratio*
738×640	3.9s ± 296ms	187ms ± 12.5ms	$x20.86$
1279×640	6.23s ± 403ms	279ms ± 23.7ms	$x22.33$
1542×1158	14.3s ± 450ms	366ms ± 3.65ms	$x39.07$

Table 5.4: Processing speeds improvements in the text segmentation module

5.1.3. Data Processing and Data Augmentation module tests

The tests implemented for this module are qualitative and thus cannot be tracked by a test specification.

5.1.4. Color Recognition module tests

The following tests are used to ensure that the functionality is fulfilled:

- T4.1: Given a RGB image and a pair of $(word, line, paragraph)$, the resulting RGB word must match the expected RGB word.
- T4.2: Given a RGB word image, the loss of information between the resulting quantized word and the expected word must not be greater than 30%.
- T4.3: Given a highlighted quantized word image, the highlight detection algorithm must detect it is highlighted.
- T4.4: Given a non-highlighted quantized word image, the Highlight Detection algorithm must detect it is not highlighted.
- T4.5: Given a quantized word image, the Color Recognition model must get a pair of font and page colors that are in the same color scale as the expected pair of font and page colors.

- T4.6: Given a pair of font and page colors, the resulting CSS name mapping must be in the same color scale as the expected pair of font and page colors.

Test number	*Has passed*	*Observations*
1	Yes	-
2	Yes	A $K = 3$ retrieves optimal results
3	Yes	-
4	Yes	-
5	Yes	-
6	Yes	-

Table 5.5: Test results for the Color Recognition module

5.1.5. Text Recognition module tests

The following tests are used to ensure that the functionality is fulfilled. Here, the error is defined as the Levenshtein distance between two words:

- T5.1: Given a binarized image and a pair of $(word,\ line, paragraph)$ information, obtain a grayscale word image that must match the expected word image.
- T5.2: Given a grayscale word image, the error between the result of the Text Recognition model, with respect to the expected result, must not be greater than 3, on average.
- T5.3: Given a result of the Text Recognition model and a pair of $(word,\ line, paragraph)$ information, the error between the resulting spell-checked word text and the expected word text must not be greater than 2, on average.

In addition, it is also required to compare the proposed model against some of State-of-the-Art models exposed before. This comparison must also detail the hyperparameter configuration and the datasets used to train each of them. Note that a self-comparison of the model among the three main datasets (Spanish, French, English) is required.

Test number	*Has Passed*	*Observations*
1	Yes	-
2	Yes	-
3	Yes	-

Table 5.6: Test results for the Text Recognition module

In addition, Table 5.7 shows the comparison of the final proposed model with State-of-the-Art models, as well as multiple configurations of the proposed models.

Configuration	*CER*	*WER*	*Speed-up*	*Datasets*
Puigcerver	5.8	18.4	-	English
Bluche and Messina	3.2	10.5	-	English
Baseline	8.98	25.82	1x	English + Spanish
50% input size	10.04	27.2	1.7x	English + Spanish
Gated Convolution	7.9	23.77	0.85x	English + Spanish
Octave Convolution	7.5	21.60	1.15x	English + Spanish
8-bit quantization	9.05	28.04	2.2x	English + Spanish

Table 5.7: Comparison among multiple HTR models

5.1.6. Language Processor module tests

The following tests are used to ensure that the functionality is fulfilled:

- T6.1: Given an empty input, return a syntax error
- T6.2: Given a malformed input, return a syntax error.
- T6.3: Given a valid word, retrieve an HTML output with a paragraph tag.
- T6.4: Given a valid sentence surrounded by $, retrieve an HTML output with a strong paragraph tag.
- T6.5: Given a valid sentence surrounded by @, retrieve an HTML output with an emphasized tag.
- T6.6: Given a valid sentence with a leading #, retrieve an HTML output with a header of level 1 tag.
- T6.7: Given a valid sentence whose first string is at least one or more '#' characters, retrieve an HTML output with a header of level X tag, being X the number of # characters.
- T6.8: Given a valid sentence with a color different from black, retrieve an HTL output with a tag having a style with a color different from black.
- T6.9: Given a valid set of sentences, retrieve an HTML output where each sentence is displayed with its appropriate tag, and they are separated by newlines.
- T6.10: Given a valid set of sentences whose first sentence has a leading '#' string, retrieve an HTML output with a header tag followed by other HTML tags. This set of sentences must contain colors different from black, and these colors must be properly reflected in the HTML output as values within a tag with the style parameter.

Test number	*Has Passed*	*Observations*
1	Yes	–
2	Yes	–
3	Yes	–
4	Yes	–
5	Yes	–
6	Yes	–
7	Yes	–
8	Yes	–
9	Yes	–
10	Yes	–

Table 5.8: Test results for the Language Processor module

5.2. System demo

In this section, the different modules are coupled so that we can get a final, single system to process data end-to-end.

The demo expects an RGB page image as input, and it outputs an HTML5 file resulting from recognizing text and color from the input. The following steps are conducted by the demo web application:

1. A file input field is used to input document pages. Only one page is processed at a time.
2. The image is binarized using *Illumination Compensation* with the default parameters. An RGB copy of the input image is kept for color processing.
3. The paragraph detection block processes the grayscale image, which once cropped is fed into the word detection block. The resulting list of word bounding boxes is then used to obtain the lines.
4. A list of cropped words from the RGB image using the word bounding boxes is created. The color recognition module processes each word and produces a *(font, background)* color tuple list.
5. A list of cropped words from the grayscale image using the word bounding boxes is created. The text recognition module processes each word and produces a list of predicted words.
6. Using the line contextual information, the error correction block processes the predicted word list.
7. The color tuple list and word list are merged into a whitespace-separated string, which is then fed into the language processor module. A HTML5 file is created as the final output.

In (Terrasa 2020), a scaffolding of the described demo can be seen.

Chapter 6. **Conclusions and future work**

In this section, the results derived from the project are discussed. Additionally, the results are contrasted with the project goals in order to evaluate if such goals have been completed to some degree. The personal insight from the project researcher is left consecutively, and a final note on future work is written.

6.1. Conclusions and future lines

Once each system module is completed, module validation tests are gathered and compared. Overall, the validation tests have been fulfilled, and the proposed system achieves high processing times at training and testing stages while obtaining comparable accuracy to state-of-the-art systems.

The need for an integrated solution and the proper refinement of each module as a way to improve the baseline results, in terms of precision and processing time, has been a key motif in the development stage, and hence the tight iterative improvement schedule in the recognition network.

Despite this, the core advances made in this project prove the feasibility and potential of the system, which is reflected in the demo. For instance, style, lexical, and semantical information from words can be extracted by the system by means of simple yet effective algorithms. This is important in two senses: on one side, this allows to edit or extend the current modules right away, as all of them are based on a common interface; on the other side, it also enables the reuse of each independent module or its complete substitution.

6.2. Goal fulfillment

6.2.1. Offline HTR for Spanish texts

This goal has been partially fulfilled. On one side, the associated text dataset has been successfully created and labeled, and the augmentation feature is also done. On the other side, the required dataset size for the model to properly recognize the characters unique to the Spanish language is small. However, this is not within reach of this project.

6.2.2. Color extraction from words

This goal has been fulfilled, as style information can be extracted from text images, such as font, highlighting or background color.

6.2.3. Apply language processing to reproduce layout

This goal has been fulfilled, as document layout can be reproduced, by following a markup specification.

6.3. Personal insight

From my point of view, I think that this project has been an in-depth opportunity to try out what I have learnt and practiced in the bachelors. Not only I have been able to research and learn about computing algorithms and deep learning, but also about software engineering, information architecture and time management. Personally, I have not appreciated to which extent this project has helped me consolidate computer science skills until I have listed them.

On one side, I feel very proud about the work done here, and I value the expertise gained in computer vision, data processing and natural language processing. On the other, I cannot wait to finish the project to continue tinkering with the blocks needed to create a more perfect system.

6.4. Skill set

The knowledge and expertise that have been gathered for the entire length of this project is notorious. Moreover, most of the practices and methodologies learnt throughout the bachelor's degree have been put to the test.

- Image Processing and Computer Vision
- Data Structures
- Oriented-Object Programming
- Neural Networks
- Machine Learning
- Scientific Computation and Data Management
- Compilers and Language Processing
- Computational Complexity and Optimization
- Algebra and Calculus
- Software Engineering
- Analysis and Design of Algorithms

Finally, there is also a set of skills needed that are not directly matched with the bachelor's domain, such as scheduling, task sharing and attainment, commonly known as *soft skills*.

6.5. Future work

In this section, the advances made in this project are reviewed from a different viewpoint. What can be done to enhance the quality of the research presented here? A set of improvements are described for future projects to inspire them.

6.5.1. Improve the Handwritten Text Recognition module

When pointing out the handwritten text recognition module, we must refer to the whole recognition process, this is, to text segmentation and text recognition.

As of the text segmentation, processing speed enhances can be conducted for paragraph-level and line-level segmentation, and accuracy enhances for the word-level segmentation. As of the text recognition, some of the latest advances described in the *State of the Art* section, such as octave convolutions or transformers, can be implemented in future works, as they improve both accuracy and processing speed. Remember that the spell-checking functionality can be improved as well, either as a separate module or integrated in the network design.

6.5.2. Extend the Modern Handwritten Spanish dataset

The novel handwritten Spanish dataset is a good start for setting a standard dataset for the offline handwritten text recognition task on a Spanish corpus. However, it greatly differs from its inspirations. Unlike IAM or RIMES, it is not entirely built up from full-page forms, neither it contains forms written by, at least, more than 100 writers.

An interesting approach for extending the current state of the dataset would be to apply synthetic sampling such as (Alonso et al. 2019; Graves 2013) – these samplers do not merely add noise to the data, but can also learn the text style and width to produce samples by combining multiple writer's unique properties.

6.5.3. Enhance Style Recognition

The work presented shows the possibilities and weaknesses of current systems to handle text styles. Aside from Spanish-related characters and highlighted words, there are tons of text styles left, such as Slavic-related characters or strikethrough, underlined text styles. A good inspiration source is the UTF-8 encoding, which gathers multiple widely used encodings, such as Latin1 or Latin/Hebrew.

6.5.4. Enhance Layout Recognition

Document layout is approached in this work by using markup languages. However, as shown in the State of the Art section, there are different proposals, such as using template layouts or processing grid layouts.

6.5.5. Add more Domain-Specific Languages

Additionally, there is also the chance to extend the current markup language, which is in fact a quite small subset of the common Markdown language (Gruber and Swartz 2004). Basic elements like unordered lists, tables or horizontal rules are not supported in this version.

Otherwise, there are as well other markup languages, which use either different layout blocks (see YAML (Evans 2001)) or different symbols (see Asciidoctor (Rackham

2002)). In any way, a set of new symbols may be needed, so the current model should be trained to learn them.

6.6. Legal information about the project

The work presented in this project is released under Creative Commons license (CC-BY-ND).

References

Adve, Vikram, and Chris Lattner. 2003. "The LLVM Compiler Infrastructure Project." Retrieved June 11, 2020 (https://www.llvm.org/).

Al-Ma'adeed, Somaya, Dave Elliman, and Colin A. Higgins. 2002. "A Database for Arabic Handwritten Text Recognition Research." Pp. 485–89 in *Proceedings - International Workshop on Frontiers in Handwriting Recognition, IWFHR*.

Alonso, Eloi, Bastien Moysset, and Ronaldo Messina. 2019. "Adversarial Generation of Handwritten Text Images Conditioned on Sequences." Pp. 481–86 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. IEEE Computer Society.

Ambler, Scott W. 2005. "Feature Driven Development (FDD) and Agile Modeling." *Agile Modeling*.

Anaconda. 2012. "Anaconda Python Environments." Retrieved June 9, 2020 (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

Anaconda. 2018. "Numba: A High Performance Python Compiler." Retrieved June 9, 2020 (https://numba.pydata.org/).

Arivazhagan, Manivannan, Harish Srinivasan, and Sargur Srihari. 2007. "A Statistical Approach to Line Segmentation in Handwritten Documents." P. 65000T in *Document Recognition and Retrieval XIV*. Vol. 6500.

Beazley, David M. 2007. "Simplified Wrapper and Interface Generator."

Bloomberg, Dan S. 2008. *Color Quantization Using Modified Median Cut*.

Bluche, Theodore, Jeroome Louradour, and Ronaldo Messina. 2017. "Scan, Attend and

Read: End-To-End Handwritten Paragraph Recognition with MDLSTM Attention." Pp. 1050–55 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. Vol. 1.

Bluche, Theodore, and Ronaldo Messina. 2017. "Gated Convolutional Recurrent Neural Networks for Multilingual Handwriting Recognition." Pp. 646–51 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. Vol. 1.

Çelik, Tantek, Chris Lilley, and L. David Baron. 2011. "CSS Color Module Level 3."

Chen, Kuo Nan, Chin Hao Chen, and Chin Chen Chang. 2012. "Efficient Illumination Compensation Techniques for Text Images." *Digital Signal Processing: A Review Journal* 22(5):726–33.

Chen, Yunpeng, Haoqi Fan, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Yan Shuicheng, and Jiashi Feng. 2019. "Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution." Pp. 3434–43 in *Proceedings of the IEEE International Conference on Computer Vision*. Vols. 2019-Octob.

Chollet, François. 2017. "Xception: Deep Learning with Depthwise Separable Convolutions." Pp. 1800–1807 in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vols. 2017-Janua.

Chung, Jonathan, and Thomas Delteil. 2019. "A Computationally Efficient Pipeline Approach to Full Page Offline Handwritten Text Recognition." Pp. 35–40 in.

Dhakar, Lokesh. 2017. "Color Thief."

Evans, Clarkj. 2001. "The Official YAML Web Site." *YAML: YAML Ain't Markup Language*. Retrieved June 10, 2020 (https://yaml.org/).

Fernández-Mota, David, Jon Almazán, Núria Cirera, Alicia Fornés, and Josep Lladós.

2014. "BH2M: The Barcelona Historical, Handwritten Marriages Database." Pp. 256–61 in *Proceedings - International Conference on Pattern Recognition*. Institute of Electrical and Electronics Engineers Inc.

Filestack. 2018. "Digitize Handwriting With Intelligent Character Recognition." Retrieved June 30, 2020 (https://blog.filestack.com/api/intelligent-character-recognition/).

Google. 2011. "About Google Books." 1–2. Retrieved June 9, 2020 (https://books.google.com/intl/en/googlebooks/about/index.html).

Google. 2020. "Google Lens - Search What You See." Retrieved June 9, 2020 (https://lens.google.com/).

Google Brain Team. 2015. "TensorFlow." Retrieved June 9, 2020 (https://www.tensorflow.org/).

Google Colab Team. 2019. "Welcome to Colaboratory!"

Graves, Alex. 2013. "Generating Sequences With Recurrent Neural Networks."

Graves, Alex, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks." Pp. 369–76 in *ACM International Conference Proceeding Series*. Vol. 148.

Graves, Alex, and Jürgen Schmidhuber. 2009. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks." Pp. 545–52 in *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*.

Grosicki, Emmanuèle, and Haikal El-Abed. 2011. "ICDAR 2011 - French Handwriting Recognition Competition." Pp. 1459–63 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*.

Gruber, John, and Aaron Swartz. 2004. “Daring Fireball: Markdown.” *Daring Fireball (Blog)*. Retrieved June 10, 2020 (https://daringfireball.net/projects/markdown/).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. “Deep Residual Learning for Image Recognition.” Pp. 770–78 in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vols. 2016-Decem.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long Short-Term Memory.” *Neural Computation* 9(8):1735–80.

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. “Densely Connected Convolutional Networks.” Pp. 2261–69 in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vols. 2017-Janua. Institute of Electrical and Electronics Engineers Inc.

Hull, Jonathan J. 1994. “A Database for Handwritten Text Recognition Research.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(5):550–54.

Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference.” Pp. 2704–13 in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society.

Johnson, Stephen C. 1975. “Yacc : Yet Another Compiler-Compiler.” *Computing Science Technical Report No. 32* 33.

Kingma, Diederik P., and Jimmy Lei Ba. 2015. “Adam: A Method for Stochastic Optimization.” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Kukich, Karen. 1992. “Techniques for Automatically Correcting Words in Text.” *ACM Computing Surveys (CSUR)* 24(4):377–439.

Laurent, Cesar, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua Bengio. 2016. “Batch Normalized Recurrent Neural Networks.” Pp. 2657–61 in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. Vols. 2016-May. Institute of Electrical and Electronics Engineers Inc.

Lauzon, Francis Quintal. 2012. “An Introduction to Deep Learning.” Pp. 1438–39 in *2012 11th International Conference on Information Science, Signal Processing and their Applications, ISSPA 2012*.

Li, Yi, Yefeng Zheng, David Doermann, and Stefan Jaeger. 2008. “Script-Independent Text Line Segmentation in Freestyle Handwritten Documents.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30(8):1313–29.

LibreOffice. n.d. “LibreOffice Dictionaries.” *2011*. Retrieved September 20, 2020 (https://cgit.freedesktop.org/libreoffice/dictionaries/tree/).

Lin, Xudong, Lin Ma, Wei Liu, and Shih-Fu Chang. 2019. “Context-Gated Convolution.”

MacFarlane, John. 2006. “Pandoc - About Pandoc.” Retrieved June 11, 2020 (https://pandoc.org/index.html).

Marti, U. V., and H. Bunke. 2003. “The IAM-Database: An English Sentence Database for Offline Handwriting Recognition.” *International Journal on Document Analysis and Recognition* 5(1):39–46.

MDN Contributors. 2020. “Unicode Property Escapes - JavaScript | MDN.” Retrieved June 11, 2020 (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Unicode_Property_Escapes).

Montreuil, Florent, Emmanuèle Grosicki, Laurent Heutte, and Stéphane Nicolas. 2009. "Unconstrained Handwritten Document Layout Extraction Using 2D Conditional Random Fields." Pp. 853–57 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR.*

Munroe, R. P. 2010. *Color Survey Results.*

Narang, Sharan, Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. "Mixed Precision Training." in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings.* International Conference on Learning Representations, ICLR.

Niblack, Wayne. 1985. *An Introduction to Digital Image Processing.* Strandberg Publishing Company.

Norvig, Peter. 2007. "How to Write a Spelling Corrector." 1–14.

Pan, Sinno Jialin, and Qiang Yang. 2010. "A Survey on Transfer Learning." *IEEE Transactions on Knowledge and Data Engineering* 22(10):1345–59.

Parr, T. J., and R. W. Quong. 1995. "ANTLR: A Predicated-LL(k) Parser Generator." *Software: Practice and Experience* 25(7):789–810.

Pham, Vu, Theodore Bluche, Christopher Kermorvant, and Jerome Louradour. 2014. "Dropout Improves Recurrent Neural Networks for Handwriting Recognition." Pp. 285–90 in *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR.* Vols. 2014-Decem. Institute of Electrical and Electronics Engineers Inc.

Project Jupyter. 2017. "Project Jupyter | Home." Retrieved June 9, 2020 (https://jupyter.org/).

Puigcerver, Joan. 2017. "Are Multidimensional Recurrent Layers Really Necessary for

Handwritten Text Recognition?" Pp. 67–72 in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR.* Vol. 1.

Python Team. 2018. "Python Release Python 3.7.2 | Python.Org." Retrieved June 9, 2020 (https://www.python.org/downloads/release/python-3610/).

Quirós, Lorenzo. 2018. "Multi-Task Handwritten Document Layout Analysis."

Rackham, Stuart. 2002. "Asciidoctor | A Fast, Open Source Text Processor and Publishing Toolchain for Converting AsciiDoc Content to HTML5, DocBook, PDF, and Other Formats." Retrieved June 10, 2020 (https://asciidoctor.org/).

Rice, Sv, Fr Jenkins, and Ta Nartker. 1995. "The Fourth Annual Test of OCR Accuracy." *1995 Annual Report of ISRI, …* 1(April):1–39.

Rojas, Raúl. 1996. *Neural Networks: A Systematic Introduction.*

Romero, Verónica, Alicia Fornés, Nicolás Serrano, Joan Andreu Sánchez, Alejandro H. Toselli, Volkmar Frinken, Enrique Vidal, and Josep Lladós. 2013. "The ESPOSALLES Database: An Ancient Marriage License Corpus for off-Line Handwriting Recognition." *Pattern Recognition* 46(6):1658–69.

Saidane, Zohra, and Christophe Garcia. 2007. "Automatic Scene Text Recognition Using a Convolutional Neural Network." Pp. 100–106 in *Proceedings of the 2nd International Workshop on Camera-Based Document Analysis and Recognition, CBDAR 2007.*

Samir, Ahmed. 2017. "Samir55/Image2Lines." Retrieved June 30, 2020 (https://github.com/Samir55/Image2Lines/blob/master/LICENSE).

Sauvola, J., and M. Pietikäinen. 2000. "Adaptive Document Image Binarization." *Pattern Recognition* 33(2):225–36.

Scheidl, Harald, Stefan Fiel Wien, and Harald Scheidl Robert Sablatnig. 2011.

"Handwritten Text Recognition in Historical Documents DIPLOMARBEIT Zur Erlangung Des Akademischen Grades Visual Computing Eingereicht Von." 90–96.

Serrano, N., F. Castro, and A. Juan. 2010. "The RODRIGO Database." Pp. 2709–12 in *Proceedings of the 7th International Conference on Language Resources and Evaluation, LREC 2010*.

Shi, Baoguang, Xiang Bai, and Cong Yao. 2017. "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* 39(11):2298–2304.

Shi, Baoguang, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. 2016. "Robust Scene Text Recognition with Automatic Rectification." Pp. 4168–76 in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vols. 2016-Decem.

Skalski, Piotr. 2012. "Make Sense!" Retrieved June 11, 2020 (https://www.makesense.ai/).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. "Sequence to Sequence Learning with Neural Networks." Pp. 3104–12 in *Advances in Neural Information Processing Systems*. Vol. 4. Neural information processing systems foundation.

Terrasa, Joaquín. 2020. "Bachelor Thesis' Demo." Retrieved June 25, 2020 (https://github.com/espetro/text-processing/blob/master/demo.ipynb).

Torvalds, Linus. 2005. "Git." Retrieved June 9, 2020 (https://git-scm.com/).

Voigtlaender, Paul, Patrick Doetsch, and Hermann Ney. 2016. "Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks." Pp. 228–33 in *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*.

Xu, Chen, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. “Alternating Multi-Bit Quantization for Recurrent Neural Networks.” in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Zhou, Victor. 2019a. “An Introduction to Convolutional Neural Networks.”

Zhou, Victor. 2019b. “An Introduction to Recurrent Neural Networks.” Retrieved June 30, 2020 (https://victorzhou.com/blog/intro-to-rnns/).

Appendices

Appendix A: Installation Guide

In order to keep the installation guide as minimal as possible, the required steps are also available at (Terrasa 2020). Note that the author of this research is using a Windows 10 OS, along with an Anaconda distribution for managing Python versions. Python 3.7 version has been used for the entire development of this work.

Bear in mind that any operating system can be used, as long as Python 3.7 and Swig 3.0 (Beazley 2007) can be installed. Other than that, you can find the package requirements at (Terrasa 2020). Just download the folder and open a terminal inside it; then, run *pip install -e .* to install the required packages, and you are ready to go.

Note: not all models have pre-trained weights already available, due to large binary files needed to store the model parameters. Nevertheless, all machine learning models include a description explaining which dataset to use, how it should be pre-processed and the methods used to train and test the models.