Segmentación de texto en Imágenes (OCRs)

Alejandro Rasero and Junhao Ge

Universidad Politécnica de Madrid

Abstract

In this document, a comprehensive implementation of an **Optical Character Recognition (OCR)** system designed from scratch will be presented, we will emphasise in the segmentation process for the precise extraction of text from images.

Multiple strategies and approaches will be addressed to solve this challenge, highlighting the advantages of segmentation compared to other commonly used techniques. The methodology used involves segmenting each letter in the image and then classifying it with the assistance of a Deep Learning model. An evaluation of the results obtained through the proposed implementation will be provided, comparing them with *state of the art* OCR models. The effectiveness of the proposed implementation was assessed using quantitative metrics, including the accuracy rate in letter identification.

The strengths and limitations of the proposed implementation will be carefully examined, emphasizing the areas where it matches existing models and those where improvements may be needed. This study aims to create a baseline model for OCR implementation that can serve as a starting point for future improvements in both performance and calculation speed. A GitHub repository with the implementation can be found in [1]

1 Overview

Our aim is to implement a functional OCR model that can successfully detect *PDF* style documents and display them in machine language with considerable accuracy for it to be acceptable. See Figure 1

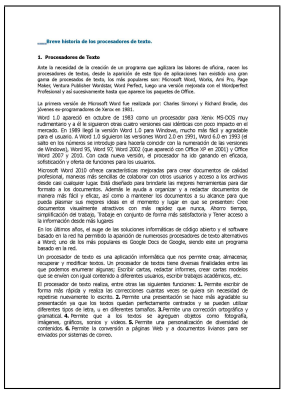


Fig. 1: Example of a *PDF style* document [2]

Some level of misspellings are expected, but our baseline is for the model to create consistent and understandable recognitions, which means that the reader shouldn't have any trouble to understand the detected text and follow both the context and wording with ease.

In order to solve this problem, we have considered three different approaches:

The first one consists of using a **Convolutional Neural Network (CNN)** to process the whole document, the model will do the segmentation of words and the classification by itself.

The second one relies on using a **Visual Transformer (ViT)**. This approach is the most desirable for obtaining good quality detection and accuracy, but, the high demand of computational resources and time make this option not suitable for the purpose of this subject.

The third option is to develop two separated models that work together, one in charge of doing the segmentation with classic image processing to get each word, and consequently each letter, and another one with CNNs to classify each letter. This system relies heavily in the segmentation process and it's quality will be defined by it's capacity to properly separate each letter.

We focused in the first and third options as a starting point. Since the third option started to show more considerable progress from the first one and we obtained noticeable results, the first option was discarded as a "future work project", and all our efforts were redirected in enhancing the quality and accuracy of the segmentation-classification process.

2 State of the art

The current state of the art is more inlined with the second option of using Visual Transformers rather than relying in classical image processing or CNN's. Specifically, they use variations of ViT's where they modify the standard architecture to fulfill their needs or rather optimization purposes.

An example can be seen in reference [3], where a pre-trained ViT is used to perform Scene Text Recognition. Instead of using the general method, which includes an *Encoder-Decoder* architecture to do text recognition task, **ViTSTR (Vision Transformer for Scene Text Recognition)** only relies in one stage *(Encoder)*. The main objective of this variation is not to have a better accuracy from the state of the art models, but to uphold its quality from the former while gaining simplicity and efficiency. See Fig. 2

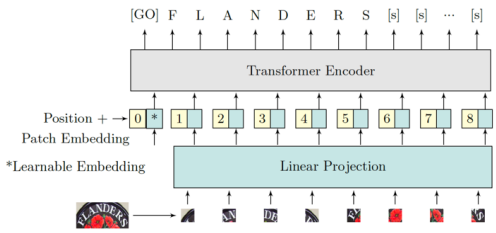


Fig. 2: ViTSTR arquitecture [3]

Vanilla ORC Transformers are also used [4] [5] [6], whether a pre-trained model is borrowed or it's trained from scratch. Nevertheless, as shown in [4] the amount of computational resources required to train a model is high, and directly importing a functional pre-trained model reduces considerably the complexity of this project.

The architecture of the model remains the same as a ViT, the image is divided into *patches*, and an embedding of each patch is fed into the encoder with its corresponding *position embedding*.

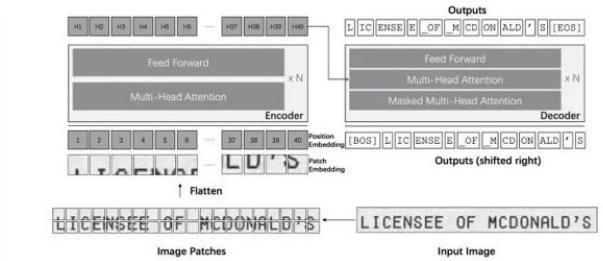


Fig. 3: TrOCR arquitecture [4]

Variations of the current Large Vision Language Models (LVLMs) are also addressed, such as **VARY** [7], a method to scale up the vision vocabulary of LVLMs. The main goal is to improve the vocabulary of models such as *CLIP*, that may encounter low efficiency in tokenizing large document-level images and even suffer "out of vocabulary" problems.

In order to make these improvements, a first *vocabulary network* is trained along with a transformer with only the decoder. This will create a vocabulary that will be merged with the original one produced by *CLIP*, thus improving the context awareness and understanding capabilities.

The main advantages of the state of the art models compared to the classic ORC approach resides in the segmentation process and computing capabilities of a more efficient architecture. Since the models are able to choose how it will treat the image and the segmentation process, it can divide and analyze in a more efficient way, giving room to a wider generalization.

3 Datasets and preprocessing

3.1 Datasets used

The OCR model proposed is based on the detection and recognition of isolated letters, subsequently forming words and paragraphs by combining the predictions of a trained CNN. To implement this approach, it is necessary to work with a dataset that provides images of isolated letters. Additionally, we must consider that the format of the texts we work with will commonly be machine-written. Therefore, we will prioritize the search for a dataset containing images of letters written on a computer. In our

search, we identified a set of images that met the necessary requirements in the **Digital Letters Dataset**[8].

The number of images was not large enough to train a deep learning model. With only 500 images for each category, we considered the possibility of performing data augmentation to increase the number of images. Although it initially seemed like a good idea, we had to discard it upon discovering that the dataset itself already contained images generated through data augmentation. Apparently, the original images were very similar in terms of the shape and contour of the letters, and the dataset creators had moved the letters around the image, bringing them closer and farther apart, as well as applying zoom, in order to increase the number of images.

After researching and looking for other datasets, we found one that included images of handwritten letters called **EMNIST** [9]. This way, we managed to increase the number of images for each category to 4000 by combining both datasets, at the cost of using letters that will not be identical to those found in the final texts, although they are similar.

It has also been necessary to consider certain aspects when starting to plan how to approach the problem. Firstly, after debating possible approaches, we decided that we would have to discard the possibility of working with numbers and limit our model to the classification and recognition of letters. The main reason behind this decision is the limitation when working with numerical images. In our case, we needed to work with images that represented digital numbers, similar to those that will be found in the input documents. In this way, we found a dataset[10] that, at first glance, seemed suitable for the training of our model. See Fig. 4

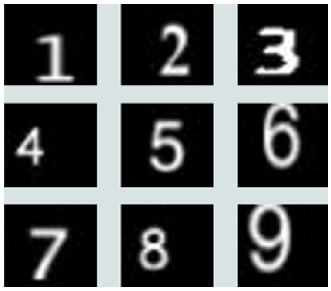


Fig. 4: Digital numbers example

However, upon analysis, we discovered that set dataset had very few images for each number compared to the number of images for each letter class. we only had around 700 images for each number as opposed to the 4000 for each letter category. Additionally, we encountered the same issue as with the digital letter dataset; data augmentation could not be applied because it had already been used.

The proposed solution was to use images from **MNIST** to increase the number of images. After studying this solution, we ultimately discarded it because many of the numbers in **MNIST** are easily confused with letters. For these reasons, we made the decision to discard the possibility of classifying numbers and limit our model to letter detection.

Fig. 5: Digital numbers problem

Another decision we made was to abandon the idea of detecting punctuation marks. Firstly, because we did not find a dataset specifically for this task, and secondly, because after several attempts to create our own dataset, we realized that the increase in the quality of the model would not be significant compared to the time we would have to dedicate expanding the capabilities of our model.

In summary, our OCR model is limited to letter detection, excluding the recognition of punctuation marks and numbers.

3.2 Preprocessing

The two datasets we decided to use required some modifications. Firstly, the images from the **EMNIST** dataset were rotated 90 degrees to the right, so it was necessary to straighten them in order to combine them with the images from the **Digital Letters** dataset and standardize the data format. Additionally, we found that in the test set there were letters for which there were no images, such as *"Z"*, so we extracted images from the training set and added them to the images of the test set. Finally, we combined the data from both datasets into one, changed the labels from *strings* to *categorical*, normalized and randomly rearranged the training data.

4 Models implementation

In this section, we will proceed to explain the implementation we have carried out to complete this project. Next, we will detail the data cleaning process, the preparation of the data for model training, the model training itself, and the image segmentation process. Firstly, it is important to mention that we have chosen to split our code into two notebooks:

- **OCR_limpieza_entrenamiento.ipynb**, where the classification model and the preprocessing process resides
- **OCR_segmentacion_prediccion.ipynb**, which contains the segmentation process

The reason for having two separated notebooks is merely for organizational purposes, therefore each process can be identified and easily understood, without convoluting everything in one single notebook.

The next two sections will focus on explaining the two notebooks mentioned above correspondingly. That is, section 4.1 will mention what has been done in the notebook **OCR_limpieza_entrenamiento.ipynb**, and section 4.2 will analyze the notebook **OCR_segmentacion_prediccion.ipynb**.

4.1 Preprocessing and model training

In section 3, we provided a brief summary of the main challenges we faced when collecting and processing data to train a Deep Learning model. Here we will review how we addressed these challenges and analyse the process of obtaining the final model that will be used to fulfil the purpose of an OCR.

Firstly, we will discuss aspects related to the **EMNIST** dataset. Upon downloading the data, we observed that it was in *.csv* format. Therefore, we used the **Pandas** library to read it. Each row contained 785 values forming images of 28x28 pixels and its corresponding label value. The labels for each letter were represented as a number that referenced the position of each letter in the alphabet, rather than representing the letter itself. Upon visualizing the images we had, we noticed that they were rotated 90 degrees to the right. So we used functions provided by the **Numpy** library to straighten them. However, this was not the only challenge we faced. Upon closer examination of the images, we discovered another issue mentioned in section 3, some letters were missing images in the test data. Specifically, there were no data for images corresponding to the letters *T*, *U*, *V*, *W*, *X*, *Y* and *Z*. The solution we opted for was to extract images of those categories from the training set and transfer them to the test images, removing them from the initial set to avoid issues during training.

The **Digital Letters** dataset was also encoded in the standard *.csv* file with the same format as described earlier, 785 columns where each one represents a pixel and the remaining column represents the label of its respective row. Thus, the cleaning process for this dataset was much simpler than the previous one. It was only necessary to extract each row as an image and convert the label into a number. Since unlike the previous dataset, in this case each label represented the letter itself rather than its position in the alphabet.

At this stage, we have the images that we will use to train the model. However, we need to make some modifications to ensure the best performance possible. We are referring to the usual steps that are taken before training Deep Learning models for image processing. Which includes normalizing the data and shuffle it randomly, changing labels type to categorical, and reshaping the images to have the desired format for the network: (28, 28, 1). Finally, we binarize the images to be in black and white instead of grayscale. This decision was made because we will be working with black and white texts later on, with the goal of standardizing image formats.

At this point, we have everything ready to begin training Deep Learning models. During the introduction, we mentioned that there were different approaches to address this problem, but the main ones involved using **CNNs** or **ViTs**. Due to computational and time limitations, we decided to prioritize the development of an OCR based on Convolutional Neural Networks.

We trained different models to find the architecture that would yield better results. The models we designed had significant differences among them, but they all shared some common parts. For instance, in all models, both the input and output remained the same. The input is based on an *Input layer* that accepts images in the format (28, 28, 1), and a dense layer as output that classifies the images into one of the 26 possible letter categories we have. All potential models used *categorical crossentropy* as the loss function, *ADAM* as the optimizer, and *accuracy* as the quality metric. Regarding the hidden layers, we chose to desing a block consisting of *convolutional layers*, *BatchNormalization layers*, and *Dropout layers*. Specifically, in this block, three *Conv2D layers* alter-

nate with three *BatchNormalization layers*, and at the end of the block, a *Dropout layer* is included to prevent overfitting. Therefore, the different models focus on varying the number of blocks used, the different parameters of each layer in the block, and the processing part through *Dense layers* following the convolutional part.

The best model we trained achieved an accuracy of over 94% on the test data 6. The architecture consisted of 5 blocks, where we progressively increased the number of parameters to improve the abstraction of the information, keeping in mind the possibility of causing overfitting. The size of the activation maps decreased progressively until obtaining activation maps of size 6x6. After these blocks, a final *Conv2D layer* was included, followed by the corresponding *Flatten layer* to prepare the information for the processing phase with *Dense layers*, which consisted of 3 layers followed by a *Dropout* in each one, and concluding with the output layer, which was common to all architectures we tested.

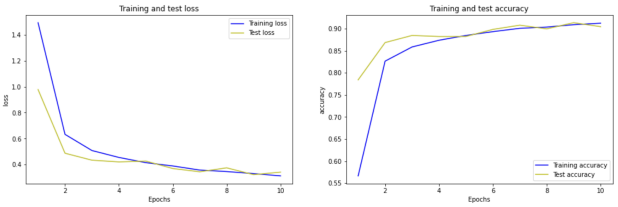


Fig. 6: Best model loss and accuracy during training

Through this process, we have managed to obtain a quite effective model that will assist us in our end goal. In the following sections, we will discuss how we have used this model to obtain the final results and how good they have turned out to be.

4.2 Text segmentation

Next, we will explain the basics that are necessary to understand the image segmentation process used to obtain the contours of each letter. After that, it will be necessary to modify the images corresponding to these contours in order to adjust them to the model format. In addition, we will review the main functions that we have implemented to get our OCR.

4.2.1 Word segmentation

As mentioned in the **Overview**, section 1. First, a segmentation process for each letter is needed in order to perform a correct classification. We will use the *OpenCV* library for python, along with other libraries such as **Numpy** and **Pandas** that were mentioned previously and will continue to provide us with many facilities.

First, all images must be converted into gray-scale, and consequently turned into binary (black and white pixels), since all PDF documents have 3-channels in RGB format.

Once the image is in the correct format, we will use the functions provided by **OpenCV** to find the contours of each word, and establish a bounding box in each one of them. In order to extract each word properly, image dilation and erosion techniques have been applied based on a specific kernel (Figs. 7, 8 and 9), which consists in the process of expanding or contracting the shapes of black pixels in a binary image respectively. A deeper explanation and underlying math can be found in [11].

Fig. 7: Original

Fig. 8: Eroded

Fig. 9: Dilated

Extracting each word with these methods carries the problem of not maintaining the ordering of the words in each text image, but just identifies its presence. A few approaches were considered, like dividing each document in paragraphs or lines and then segmenting each word, but after a few tests on this methodology, text fonts and difference in sizes like titles caused troubles. At the end, an easier approach was made.

After obtaining the bounding boxes of each word, the coordinates are saved in an array and are sorted by the western reading order, from top-left, to bottom-right, which means that first we will sort which word belongs to each line by looking the *y-coordinates*. Then, from each line, words will be sorted by the *x-coordinate*. See Fig. 10 as a reference.

4.2.2 Letter segmentation

The letter segmentation process follows the same methodology as the word segmentation, but some changes are applied.

The morphological operations kernel of erosion and dilation is modified due to image sizes, and the letter ordering only requires a comparison of the *x-axis* from each bounding box, since there are no lines to compare.

Furthermore, a set of functions have been created to process the segmented letter and to ensure that this image follows a correct format. Some of the funcions are:

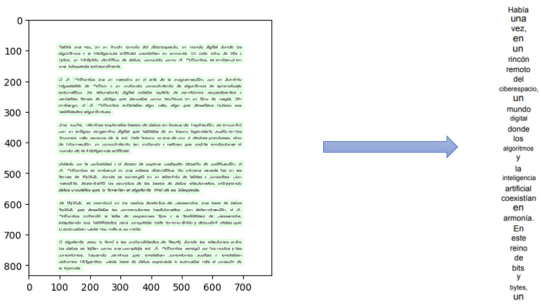


Fig. 10: Text segmentation of the first sentence in a document

- **rellenar**, resizes each letter to (28,28,1), which is the size of the images used for training. This function is necessary because the letter segmentation process is based on finding the contours of the letters, as we have already explained. Therefore, each contour has different dimensions while the model input is fixed. So, it is necessary to fill in the image of each letter to match the training samples.
- **invertir_colores_en_matriz**, that invert the pixels of the image. There are cases in which the images we extract represent white letters on a black background, and there are opposite cases. With this function we combine the two possible scenarios to be able to classify them with our model.
- **recortar**, cuts the image by a certain value. Specific function for cases in which the contours of a letter have a height greater than 28 pixels.
- **separar**, an auxiliary function that separates two letters in an image. It is specific for cases where the contours of a detected letter is wider than 28 pixels, meaning two letters have not been correctly divided.
- **redimensionar**, resizes the whole document image, usually to a bigger image. We found that the segmentation worked best by resizing the images to 1920x1080 pixels. This enlarges the silhouettes of the letters and makes them easier to isolate.

It is not expected for the segmentation process to be perfectly accurate nor the input image to be clean. There are certain bounding boxes that has capture punctuations. "Commas" and "ending sentence points" are not desirable to keep since the model has not been trained with those symbols. On the other hand, human input mistakes have to also be considered. The title of image 1 can be served as an example, where the title begins with a set of repeated points.

Thus, a filter has been set to remove all that can be considered "noise". Since the expected input is a document style image, we can consider that each word will have a minimum size, and naturally, due to the difference in shapes between words and punctuations, the second one will be considerably smaller. So any bounding box that does not have a minimum size either vertically or horizontally will be removed. See Fig. 11.

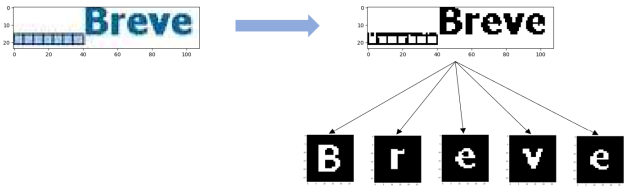


Fig. 11: Word segmentation example and noise removal

On the other hand, there are certain letters which have not been separated correctly due to them being too close from one another. This is an issue from the text format itself that typically occurs with the same pairs, like "wt" or "rt", letters that can be clearly identified by the human eye, but when zoomed in, there's a decrease in the quality due to the number of pixels, so the nearest point of each letter is grayed out instead of being a defined as a white pixel.

Once we understand the basics for letter segmentation, its implementation is divided into several functions that we have implemented. A brief summary of them would be:

- **load_image**, used to load images.
- **process_image**, is in charge of applying the erosion and dilation processes necessary to obtain the word outlines.
- **get_palabras**, using the contours obtained with the previous function, this one extracts each word in the positional order present in the input image.
- **forward**, is the core function of our OCR model. It calls the previous functions to extract the words from the image, then for each word we get the letters that make up that word. To these new images, we apply the format functions explained before to match the input with the training images and classify them with the model we have trained. Finally we make use of a grammar checker that will be explained later, and a function implemented by us as the metric of our OCR, which will be also addressed.

Once we started testing the classifier, we realized it often confused similar letters, and was not able to match complete words. In order to solve this we have used a grammar checker from the **language_tool_python** library, that will correct each misspelled word to the closest word embed-

ding it can find. Also, during training we realized that our model made recurring errors that the checker did not detect. Therefore, we defined a list with common errors to manually correct them.

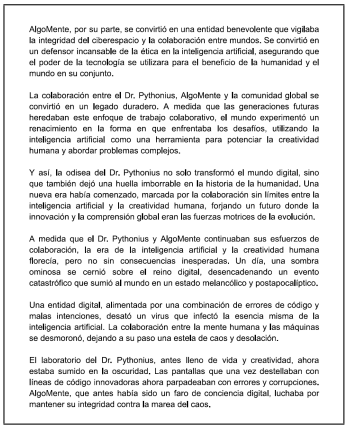
However, this solution was not perfect. When the corrector detected an error, we would select the first suggested option as a solution, which was not always the correct one. Therefore, we established certain use cases for the corrector. For example, we know that our model will always generate an image for each letter in a word. Thus, it is not possible for the correction of the word to have fewer letters than the predicted one. On the other hand, it would be possible to predict fewer letters than the actual ones, as in cases mentioned with letters like "rt" or "wt." However, considering these cases yielded worse results than if we did not take them into account, we configured our OCR to consider only those corrections that had the same number of letters as their word prediction.

To conclude this section, we will briefly explain how we measure the quality of text image processing. Currently, state of the art models use a metric called **Character Error Rate (CER)**. This metric is based on measuring the number of substitutions, deletions, and insertions of letters that need to be made for the word to be correct, a deeper explanation can be found in [12]. We have decided to implement our own metric, which is essentially the opposite of **CER**, and it would be very similar to accuracy. This is implemented with the **contar** function, and its operation is summarized by counting the number of letters that are in their correct position. Additionally, it indicates how many complete words have been guessed correctly.

5 Obtained results

Having explained the functioning and implementation of our OCR model, we will explore the results obtained when predicting different images. To briefly outline the prediction process, we can say that it relies on calling the main functions explained in the previous section. Once the image to be processed is selected, we load it using the **load_image** function. Then, we extract the contours of the words with the **process_image** function. Subsequently obtain the words through the **get_palabras** function, and finally retrieve the text from the image with the **forward** function.

As a method to test the quality of our model, we decided to use 10 images, such as the one shown in Fig 12.


AlgoMente, por su parte, se convirtió en una entidad benevolente que vigilaba la integridad del ciberespacio y la colaboración entre mundos. Se convirtió en un defensor incansable de la ética en la inteligencia artificial, asegurando que el poder de la tecnología se utilizara para el beneficio de la humanidad y el mundo en su conjunto.

La colaboración entre el Dr. Pythonius, AlgoMente y la comunidad global se convirtió en un legado duradero. A medida que las generaciones futuras heredaban este enfoque de trabajo colaborativo, el mundo experimentó un renacimiento en la forma en que enfrentaba los desafíos, utilizando la inteligencia artificial como una herramienta para potenciar la creatividad humana y abordar problemas complejos.

Y así, la odisea del Dr. Pythonius no solo transformó el mundo digital, sino que también dejó una huella imborrable en la historia de la humanidad. Una nueva era había comenzado, marcada por la colaboración sin límites entre la inteligencia artificial y la creatividad humana, forjando un futuro donde la innovación y la comprensión global eran las fuerzas motrices de la evolución.

A medida que el Dr. Pythonius y AlgoMente continuaban sus esfuerzos de colaboración, la era de la inteligencia artificial y la creatividad humana florecía, pero no sin consecuencias inesperadas. Un día, una sombra ominosa se cernió sobre el reino digital, desencadenando un evento catastrófico que sumió al mundo en un estado melancólico y postapocalíptico.

Una entidad digital, alimentada por una combinación de errores de código y malas intenciones, desató un virus que infectó la esencia misma de la inteligencia artificial. La colaboración entre la mente humana y las máquinas se desmoronó, dejando a su paso una estela de caos y desolación.

El laboratorio del Dr. Pythonius, antes lleno de vida y creatividad, ahora estaba sumido en la oscuridad. Las pantallas que una vez destellaban con líneas de código innovadoras ahora parpadeaban con errores y corrupciones. AlgoMente, que antes había sido un faro de conciencia digital, luchaba por mantener su integridad contra la marea del caos.

Fig. 12: Example of a test image

The images consisted of text written on a computer, a story created by **ChatGPT**. All of them have dimensions ranging from 1100 to 1400 pixels in height and from 800 to 1100 pixels in width. They are mainly composed of letters, with few numbers among the characters to be recognized. It is worth mentioning that each document contained on average 330 words to be recognized.

Using the quality function mentioned earlier in section 4.2 we are able to determine how many letters have been correctly classified in their position and how many complete words have been predicted accurately. Thus, the average accuracy rate for isolated letters (considering only if they occupy the correct position) is 85%. In the best case, our OCR is capable of achieving an accuracy of 89.98% for letters, while in the worst case, the letter accuracy rate drops to 77.36%. When it comes to the accuracy rate for complete words the percentage drops significantly. We are talking about a difference of around 10% between both metrics. More precisely, the rate of correctly predicted complete words is around 74%. In the best case this value increases to 83.3%, and in the worst case it decreases to 63.5%.

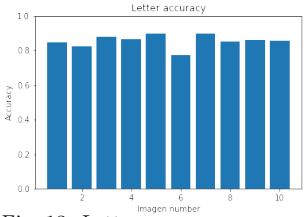


Fig. 13: Letter accuracy for each image

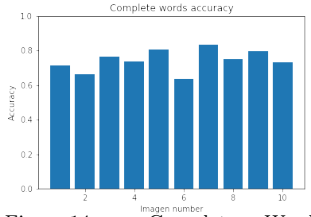


Fig. 14: Complete Word Accuracy for each image

We believe that these metrics could be improved by enhancing both the segmentation process and optimizing the use of the spell checker. In other words, in all images, fewer words are predicted than actually exist. Thus, the

model predicts an average of 9 words less than it should. However, in models with a lower accuracy rate, the number of predicted words is 17 words less than the actual number. Considering that, in the Spanish language, the average number of letters per word is 5, our model predicts 85 less letters. This means that models with worse results operate with a base error rate of around 4%. On the other hand, the corrector returns the word most similar to the predicted one, but without taking into account the context. Therefore, we believe that the use of a trained and advanced linguistic model could allow us to make predictions at the sentence level, improving the quality of the results.

Despite this last problem, partly related to the spell checker, it is worth mentioning the remarkable improvement achieved when using the library **language_tool_python**. Thanks to the corrections from this interpreter, the accuracy rate increases by around 10%, considering correctly predicted letters. The improvement is even more noticeable when looking at the rate of correctly predicted complete words, which have an increase of around 15%.

5.1 Model limitations

The main limitations of our model can be summarized in three points. The first one is that the segmentation is not valid for any image size; therefore, for images with heights outside the range of 1100 to 1400 pixels, and widths outside the range of 800 to 1100 pixels, we cannot guarantee that the quality of the results will be maintained. The main cause of this problem is that the segmentation process is very delicate and not suitable for every image. Different images have varying font sizes, changing thickness, and inconsistent spacing between words. This is the reason why segmentation is a method that is difficult to generalize.

Secondly, the time required for predicting an image is considerably high. We must consider that each image contains more than 300 words, and the average number of letters in each text is close to 2000. Therefore, for each word, the segmentation process of each letter must be performed, followed by a feedforward process for each of them. Subsequently, all predictions need to be combined to form words, and the correction process must be carried out. All these steps contribute to an average processing time of 2 minutes per image.

Lastly, this OCR model has certain limitations when predicting characters. As mentioned in previous sections, it is not possible to recognize numbers or punctuation marks. However, there is one more aspect that the model does not take into account, and that is line breaks. This is a significant limitation because the OCR only returns the text present in the image, disregarding its structure. This limitation is a consequence of the implementation approach. During the image processing to extract a list of contained words, line breaks are removed, preventing the reconstruction of the original text structure.

However, these limitations have a straightforward solution. For the first issue raised, it would suffice to create a different segmentation approach for each range of image sizes. This way, a broader range of formats could be accommodated easily. Regarding the second issue, one could turn to a pre-trained language model for text processing. In our case, we conducted some small-scale experiments using **GPT-3**, and the quality of the results improved. Therefore, by optimizing the chosen model's actions, significantly superior predictions could be achieved. Finally, the problem of the limitation in recognizing numeric characters and punctuation marks could be addressed if we could find a dataset enabling the inclusion of such data in the training process. On the other hand, in order to detect line breaks, a sequence detection process could be included in the segmentation process, allowing for the reconstruction of the original text structure.

All these proposals remain as mentions since, due to time constraints, we have not been able to test their implementation. However, it is crucial to consider that they are real and feasible solutions. These approaches could improve the results already obtained, broaden the scope of application of our model and help to reconstruct the original image more faithfully.

6 Comparisons

Although the results obtained are not bad with an average accuracy of 84%, it is still behind the state of the art models.

Current ViT models are able to reach an average of 93%, while also allowing a wide variety of images used as input and still get reasonable results.

Our model may be quick to train compared to the training times of a Transformer, but it is still not able to generalize to different kind of images and only accepts *PDF* style documents of a certain kind. Changing the words size in the document or introducing other fonts that are not com-

monly used may decrease significantly the quality of the results.

The architecture of the state of the art models are different, and even though the end goal is the same, the way of processing and classifying images do not share the same process. While we are specifying the text segmentation process with classic image processing, and analyzing the input image before feeding the words to the classifier, the state of the art models often lean to let the very model to decide its way of segmenting the words, identify letters and analyze the input data. As shown in images 2 and 3.

On the other hand, there are models that are currently deployed for public use, which may not reach the quality of using ViTs but still have very good results such as *Tesseract* [13], an OCR engine that uses LSTM focused on line recognition. It has its respective python library *pytesseract* and can be seen commonly used along other image processing libraries to extract text [14].

7 Future works and Possible upgrades

We have mentioned the limitations of our model, and have also proposed some solutions to the problems we have raised which we believe could improve performance. Apart from them, we have to point out that there are other lines of development that are very interesting to continue the legacy of this work.

Our current project is not a general purpose OCR, since it is only meant for processing a specific type of images. So the first major improvement that must be done it is in the capability for generalization. We believe that the key resides in improving the segmentation process for the model and if we can get a bigger dataset to train with the better.

This consideration in obtaining a richer dataset also applies for the inclusion of numbers and context punctuations such as *"commas", "parenthesis", "points"...* as of the moment, we decided to discard the possibility to include those options due to the lack of proper data to train the model with.

Bibliography

[1] Junhao Ge Alejandro Rasero. *Image-text-segmentation-OCRs*. URL: https://github.com/Junhao42/Image-text-segmentation-OCRs-/tree/main.

[2] Claudio Javier Garcia Martinez. *Breve historia de los procesadores de texto*. URL: https://www.calameo.com/books/0022156170fab1ea1ebe1.

[3] zogojogo. *Network architecture of ViTSTR*. URL: https://github.com/zogojogo/text-recognition-wii/tree/main.

[4] unilm. *TrOCR*. URL: https://github.com/microsoft/unilm/tree/master/trocr.

[5] him4318. *Transformer-ocr*. URL: https://github.com/him4318/Transformer-ocr/tree/master.

[6] NielsRogge. *Microsoft's TrOCR for fine-tuning*. URL: https://github.com/NielsRogge/Transformers-Tutorials/tree/master/TrOCR.

[7] *Vary, Scaling up the Vision Vocabulary for Large Vision-Language Models*. URL: https://huggingface.co/papers/2312.06109.

[8] Adam Kaniasty. *Digital letters Dataset*. URL: https://www.kaggle.com/datasets/adamkaniasty/digital-letters?select=digital_letters.csv.

[9] Chris Crawford. *EMNIST dataset*. URL: https://www.kaggle.com/datasets/crawford/emnist/code.

[10] Kshitij Dhama. *Printed Numerical Digits Image Dataset*. URL: https://www.kaggle.com/datasets/kshitijdhama/printed-digits-dataset.

[11] Bradski and Kaehler. *Morphological Operations, Eroding and Dilating*. URL: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html.

[12] Kenneth Leung. *Character Error Rate (CER)*. URL: https://towardsdatascience.com/evaluating-ocr-output-quality-with-character-error-rate-cer-and-word-error-rate-wer-853175297510.

[13] *Tesseract, a LSTM based OCR engine which is focused on line recognition*. URL: https://github.com/tesseract-ocr/tesseract.

[14] George Stavrakis. *Extracting text from pdf files with pythonn*. URL: https://towardsdatascience.com/extracting-text-from-pdf-files-with-python-a-comprehensive-guide-9fc4003d517.