



Book Cover Text Processing

University of Washington Data Science Capstone Project Report

Submitted By

Vamsy Atluri, Ryan Bald, Sayli Dighde & Srinivas Kondepudi

Introduction

The aim of our project was to come up with a solution that enables an Italian non profit library - Biblioteca Italiana - to catalog their books more easily. More specifically, our deliverable is a model that helps the library find cover pages of books they know the details of, like author name and title.

Using a combination of Computer Vision, Convolutional Neural Networks and Natural Language Processing techniques, we built a model that could extract the text written on an image.

Motivation

Biblioteca Italiana has an [online catalog](#) that lists all the books available in the library. But many of them do not have proper thumbnails which ideally would be cover pages of the books themselves.

The workflow at present involves volunteers clicking photos of books and aggregating them in a shared folder with generic file names for the images. In order to select an appropriate cover page for an entry in the catalog, the user has to browse through all images present and manually select the appropriate one.

This is a pipeline that will not scale when the number of books, and thereby images to sort through increases. While we did not have a pre specified input to use or output to strive for, the major motivation behind the project was to help improve this pipeline by making it more automated and less manual.

Data Summary

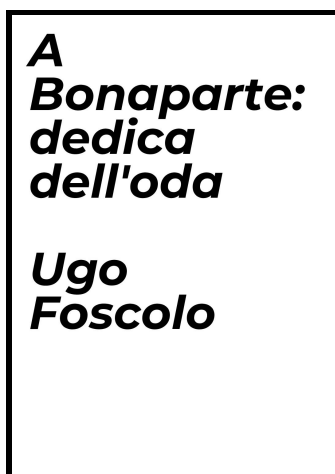
Our model is split into three segments. While we will be discussing these in much more depth in the next section, they are listed here to help understand the data sources used for each step. The three segments are:

- Crop all characters from an image
- Predict which letter each cropped character is most likely to be
- Segment the predicted strings into most likely words

For the character extraction model, we synthetically generated a set of cover pages using a combination to mimic real life pages. The data used was:

- a. A list of (author, title) entries scraped from the University of Rome's Italian library [website](#).
- b. Around 400 font files (.ttf) from [Google Fonts](#).

For each (author,title) pair, we used a combination of 7 random fonts and 8 random font sizes to generate 56 (7*8) title pages. We ended up with a dataset of around 150k title pages. An example title page is as follows:



For the character prediction model, we used two data sources:

- Handwritten letters from the [EMNIST dataset](#).
- Printed letters of varying fonts and emphases from the [Chars74K dataset](#).

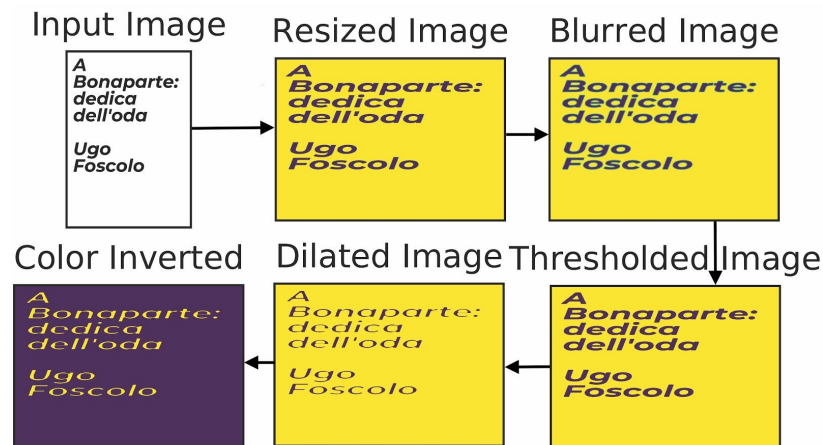
To train the word segmentation model, we used a corpus of 10000 Italian sentences obtained from the [Leipzig Corpora Collection](#).

Approach

Crop All Characters In Title Page

- Our initial plan was to use a deep learning network, more specifically to train a CNN on the generated title pages data such that it can learn how to differentiate between individual characters.
- But in order to train the model we needed labelled training data in the form of pre-calculated bounding box coordinates for each title page.
- For each of the 150k images, this meant drawing on average around 40~50 bounding boxes.
- Doing this manually was not possible and we couldn't come up with a feasible automated solution for this.
- We also explored using unsupervised training approaches but couldn't find methods amenable to our specific work case.
- Hence, we decided to shelve the deep learning approach and switched to using OpenCV.

Image Pre-Processing



Firstly we perform a series of preprocessing steps on the the image, primarily aimed at reducing noise and making it easier to extract individual characters. Figure 1 shows the steps in order of execution.

As shown, the steps involved are:

1. **Resizing:** We calculate the area of the title image covered by text and compare this area to the total size of the image. The image is then resized by a change factor depending on this ratio. For e.g. if an image is of size 500 units and the text only covers 10 units, we resize this image to make it 5 times larger than the original. This helps to increase the detectability of smaller fonts.
2. **Blurring:** To remove noise from the image, we use Gaussian blurring which is highly effective in removing gaussian noise.
3. **Thresholding:** We convert the image to a bimodal one in this step. Though the image is grayscale, we have pixels of different intensities. Thresholding converts all pixels to either 0 or 255 depending on a threshold. More specifically, we use adaptive thresholding which calculates the threshold for small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.
4. **Dilation:** Since we have continuous text characters which can be very close to each other, in some fonts we often get two characters merged in one single bounding box. To fix this, we perform dilation on the image where we define a kernel size and as the kernel is scanned over the image, we compute the maximal pixel value overlapped by the kernel and replace the image pixel with that maximal value. In simpler terms, when performed on white pixels on black background, dilation enlarges the white pixels.
5. **Image Color Inversion:** This step is required from an OpenCV perspective because the function to find contours requires white text on a black background.

Get Bounding Boxes for Characters

Once we have our image in the requisite format, we use the `findContours()` function in OpenCV to generate bounding boxes for individual characters.

A contour can be best explained as a contiguous area of similar pixel values. Using this function we get a set of (x,y,w,h) coordinates that can define each characters' bounding box. While (x,y) identify the bottom left corner of each bounding box, w and h identify the width and height of the box respectively.




Extract Characters in Order


Once we get the coordinates for bounding boxes in the previous step, cropping them is very straightforward in OpenCV. But simply sorting this list of coordinates does not give us the characters in their correct order of appearance in the text.


To solve this, we used sklearn's [DBSCAN](#) algorithm to cluster points based on the y coordinate values. This helps to differentiate between lines of text. Each set of y coordinates belonging to characters in the same line of text is classified as belonging to the same cluster.

Once we have the list of coordinates clustered by lines, we go through them one line at a time and crop the characters from each line in ascending order of the x coordinates.

Performing this operation on the image above gives us extracted characters in the order shown below, with each character storing the value of the cluster it belongs to or in other words its label.

Line 1: 

Line 2: 

Line 3: 

Classify Each Cropped Character as a Letter

- We started by using a basic neural network for our classification model, following [this tutorial](#).
- Our first big improvement with the model was changing the format of our training data from black text on a white background to white text on a black background, providing our model with cleaner inputs since there is more non-text space than text-space in the images we were using and the color black corresponds to a pixel value of 0.
- We continued tuning the basic neural network for minor improvements but our next big improvement was when we switched to a convolutional neural network to take advantage of the matrix structure of our data.
- After more tuning of the CNN, we arrived at our final model, briefly described below.

Train CNN Model

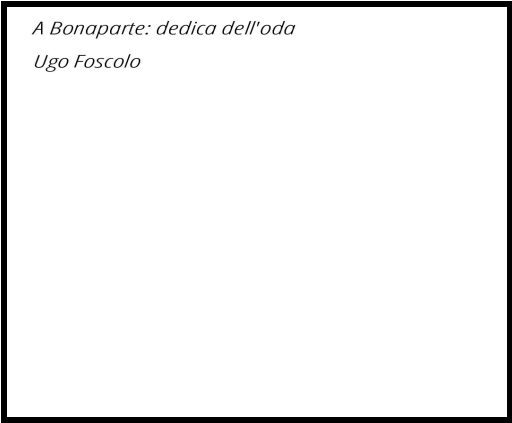
We used a 2-layer CNN (64-32) with a 4x4 kernel size (on 28x28 input images) and a cross-entropy loss function, which was trained for 2 epochs on the character dataset.

Use Trained Model to Predict Letters for Each Character

Once the CNN model is trained, we pass each cropped character to the model which calculates the probabilities of the image being a . letter and returns the letter which has the maximum probability.

Return Predicted Letters by Line

Let's consider the sample title page below:



A Bonaparte: dedica dell'oda
Ugo Foscolo

The characters in the page were already clustered by their y-coordinates. This leads to 2 clusters of cropped images. Now we predict the character each image is likely to be and return the results in the same cluster format.

For this sample page below, we get the following output:

Cluster Label (Line Number)	Predicted Characters
Line 1	'ABONAPARTEDEDICADELYODA'
Line 2	'UGOHOSCOLO'

Segment Predicted Strings Into Most Likely Sequence of Words

- One of the more significant issues we faced in this project was the inability to detect spaces in the title page.
- While we are able to detect characters and crop them from the page, we are unable to identify the presence of a space between two characters which would have helped us break the text down into its constituent words.
- To solve this issue, we built an NLP model that when trained on a corpus of Italian texts, could break a string of letters into the most likely sequence of words.
- Initially trained on a corpus of 10k sentences, we found a better dataset of 1 million sentences which we ended up using in our final model.

The way the model is built is it performs the following steps:

1. Given an input string of letters, we break the string into a list of splits. That is, each string is broken down into all possible combinations of separate words.

2. Then for each valid split, it calculates the probability of each constituent word in that split which is determined by drawing on the probability distribution of all the words in the corpus used to train the model.
3. Then, it calculates the total probability of each sequence by taking a product of the individual probabilities of the splits.
4. Finally, the model returns the sequence of words with the highest total probability.

Example

One of the longest titles in our database was “Della nuova scuola drammatica italiana”, which in English means “Of the new Italian drama school”. Putting this title through the segmentation function without any spaces we get the following results:

```
# Trying on 'Della nuova scuola drammatica italiana'
segment('Dellanuovascuoladrammaticaitaliana')

['D', 'ella', 'nuova', 'scuola', 'drammatica', 'italiana']
```

For the example page from which we extracted characters above, we get:

```
title_page_x = ['ABONAPARTEDEDICADELYODA', 'UGOHOSCOLO']
```

```
for word in title_page_x:
    print(segment(str.lower(word)))

['a', 'bonaparte', 'dedica', 'del', 'yoda']
['ugo', 'ho', 'scolo']
```

Results

After the pipeline followed above, we are able to achieve the following:

1. Extract all the characters present in a title image and cluster them by the line they present in.
2. Predict what each character is and return the letters in each line.
3. Use NLP to split the predictions into a list of words present in the title image.

While we are able to get satisfactory results with both our dataset images as well as other system generated images with text, a real world image taken by a phone camera has too



much noise which our model is not able to eliminate completely. This results in incorrect bounding boxes and improper cropping.

Our CNN classification model was extensively tuned for the ideal split between printed and handwritten characters in the training data. We found that using a 75% printed/25% handwritten split of the training data resulted in a 95.5% validation accuracy and a 92.5% test accuracy on printed characters from fonts that were not seen in the training data. These are good numbers for an isolated CNN model, but the differences between our train/test data (individual characters with no surrounding noise) and the actual input to our classification model (cropped characters from words) caused for lower accuracies when classifying extracted characters from our generated title pages.

Overall, we were not able to achieve our initial aim of identifying the author and title of each book by using the title page. But on the positive side, our model produces a list of words that are most likely to be present in the title page. By associating this list with the book's title page image, we made it much easier for Biblioteca Italiana to search for the correct cover page by filtering the results. Instead of going through the entire folder to select the right image, they can use keywords like the author name and title to search for images that have these words in them.


Future Work

Certain improvements to the model can be made in order to improve the accuracy of our results. While some could not be implemented due to time constraints, we were not able to gain the technical know-how to implement others. A list of feature additions that are marked for future work at this point are:

1. Letters with tittles/superscripts like 'i' and 'j' are extracted as two characters. A way to group them together is required.
2. Real world images with significant noise give erroneous bounding boxes. Using better denoising is a possible fix.
3. Expanding classification model to predict special characters will make it more useful and generalized.
4. The word segmentation model works better on natural text than proper nouns like author names. Improving this model is essential to get better predictions.

Concluding Statements

While we did not achieve all the targets we set out for and are not completely satisfied with the results, we have set in place a pipeline that can be built on and improved.



Our key takeaway would be the experience gained from working in the areas of Computer Vision, Neural Networks and NLP, especially considering our team had little exposure to these fields before the project.

References

1. [Peter Norvig's NLP guide](#)
2. [OpenCV Documentation](#)
3. [Software Carpentry Blogs on Image Processing](#)
4. [Image Processing Overview with OpenCV](#)
5. [Stanford's Text Tokenization Tutorial](#)
6. [The Chars74K image dataset](#)
7. [The EMNIST Dataset](#)
8. [Tensorflow Documentation](#)
9. [The Leipzig Corpora Collection](#)