Final Project
Language Identification Using OCR

Eytan Ivan Reuven
Alexander Gerovichev

1. Main Goal

Our main goal in this project is to identify the **language** of a paragraph using **Optical Character Recognition(OCR).**

2. Introduction

Our project has two main parts: **Character Recognition** and **Language Identification**. First, our algorithm finds each of the characters in the paragraph, using an OCR Method. Then, the algorithm compares the characters to the ones on his dataset and returns the language of the paragraph.

2.1 OCR

Optical Character Recognition is the process by which an electronic(i.e., computer) device recognizes printed or written letters or numbers. Using OCR software allows a computer to read static images of text and convert them into editable, searchable data. OCR typically involves three steps: opening an image of a document in the OCR software, recognizing the document in the OCR software, and then saving the OCR-produced document in a format of your choosing. [A, B]

In our project, We'll target the recognition of the text. Moreover, we would like to identify the language in which the text is written.

2.2 Language Identification

Automatic language detection is the first step toward achieving a variety of tasks like detecting the source language for machine translation, improving the search relevancy by personalizing the search results according to the query language, providing uniform search box for a multilingual dictionary tagging data stream from Twitter with appropriate language etc. While classifying languages belonging to disjoint groups is not hard, disambiguation of languages originating from the same source and dialects, as we try to implement in this project, still pose a considerable challenge in the area of natural language processing. [C]

We identify the language by comparing the letters of the text to a list of letters from a known language. We're implementing the comparison using Moments and Skeletonize.

2.3 Moments

The method of moments is to use statistical properties of objects to match.
Image moments are a weighted average of image pixel intensities.
Higher order moments can be used as unique shape descriptors for an object, as we do in this project. [G]

2.4 Skeletonize

The aim of the skeletonization is to extract a region-based shape feature representing the general form of an object. [H]

The skeleton is thin. It contains much less points than the original elongated object (therefore it is an object reduction). The skeleton represents local object symmetries, and the topological structure of the object. [H]

The skeleton is very sensitive to changes, and will use this fact in our project.

2.5 Importance of the Topic

Language identification is one of the most important fields in nowadays computer science research.

Language identification has become an important issue as more and more language data are making their way online. Since there is much online text that may not pass immediately before a human's eyes, computational systems need to be built that can perform this task. [D]

3. Approach and Method

Our algorithm is based on five main stages:

1) Collecting a dataset,
2) Preprocess the above dataset(with stages 3 and 4),
3) Implement an OCR algorithm,
4) Feature extraction:
    a) Moments
    b) Skeleton's moments
    c) Special Region [see 3.4] shape pxls ratio
5) Identify the language.

In addition, we took some assumptions on the input:

- The input will be a **clean text**(i.e., 2 colours image) in **English**, **German** or **Icelandic**.
    - The first isn't really an assumption, but a boundary for the project. Our goal is to classify the letters, not find the text in a noisy image.
    - The second part is a simplification of the main goal. The algorithm will be the same even if we take all the germanic languages, but due to the fact that this is part of a course, we'll simplify the dataset and the run time.
- The paragraph is a **random** paragraph of a **random** language(from the list above).
    - The randomization of the paragraph gives us a normal distribution of the letters in the input text. From this, we can deduce that with high probability, if a paragraph is in German/Icelandic, it will contain at least 1 special letter.
    - The second randomization, i.e. the randomization of the language, gives us the ability to assume that if our algorithm detects both Icelandic and German equally, we can return German because more people are speaking German and therefore there is a higher probability the paragraph is in German,
- The font size of the paragraph is 28-35
    - We would like a font that is big enough s.t we can crop the image with high accuracy. In addition, the font of the dataset is 31 and we want the dataset to represent the actual input texts.

We'll go through each of the stages and explain the process.

3.1 Collecting the Dataset

Through our attempt to find a dataset, we noticed that there was no dataset of German special letters. Therefore, we prepare a mini-dataset of each of the German special characters: ***äÄüÜöÖßẞ***. For the exact same reasons, we prepare a mini-dataset of each of the Icelandic special letters: ***öÖðÐæÞáÁéÉíÍóÓúÚýÝ***. We found out that ***öÖ*** are in both Icelandic and German, and due to the fact that more people speak German(and the second assumption), we decided that we'll identify those letters as German. [K, L]

We prepared a dataset of 3 images per letter from 3 different fonts: ***Arial***, ***David*** and ***Times New Roman***. Because we wanted that the identification of the letters wouldn't be biased, we prepared the same dataset for English letters, although a dataset of English letters is available as open source.

3.2 Preprocess the Dataset

The dataset contains all the letters in English , the special letters in German and all the special letters in Icelandic. Each image from the dataset got a label:

- English letters get a label between 100 and 199 (based on the Ascii value).
- German letters get a label between 200 and 299
- Icelandic letters get a label between 300 and 399

Moreover, all the letters in the dataset go through stages 3.3 and 3.4.

3.3 Implement an OCR algorithm

Our OCR implementation was based on a paper from P.Subbuthai, A.Periasamy and S.Muruganand [F] from 2012. In their paper, they offer 7 steps, from reading an image to a complete OCR in clean text(i.e., 2 colours image). The steps were:

A. Get the input Image as a gray image.
B. Convert the image to binary image.
C. Find the edges in the image.
D. Dilate the edges of the image.
E. Convert the image to binary image again.
F. Fill the image in the correct places.
G. Crop the image.

We mainly implemented that using existing tools available as open source.
The following libraries (of openCV) were in use:
Firstly, to use our algorithm we resize the image with ***cv2.resize()***, this allows us to get a bigger image that looks like the original one by resampling the pixel area relation of the source image and reconstruct it to a bigger image. with this we were able to catch the edges much more accurately.

Before: After:

To convert the image to binary image we used ***cv2.threshold()*** which helped us to change the pixels only to white or black as we do not need any other color to recognize the character form.
Then we used the ***Canny's algorithm*** which we learned in class to get the edges of every character.
Example:

Next step was to use the ***cv2.dilate()*** which increases the white region in the image and size of the foreground object increases, and then again we convert the image to binary image.

Then we fill every hole in each character (a,o,p..) with ***cv2.floodFill()*** so we will be able to extract it as one character.

A problem we encountered is that the fill is not whole as we can see in the image above. Our solution for that was to convert the black pixel to white and vice versa. Then we know the pixel (0,0) is connected to the background. So we can extract the background, by simply doing a ***cv2.floodFill()*** operation from pixel (0, 0). Pixels that are not affected by the ***cv2.floodFill()*** operation are necessarily inside the boundary. The flood-filled image when inverted and combined with the thresholded image gives the foreground mask we needed.

For our final step we converted back the white and black pixels and used ***cv2.findContours()***, this library finds all the continuous points of an object in an image which is every character we already outlined with the previous steps. With our contours we could get boundingBox on each character and crop it out for our identification. We used a threshold of -35 on the bounding box to be able to catch special characters as one unit for example ***ä***. The "extra" 35 is the base for the "Special Region"[See 3.4].

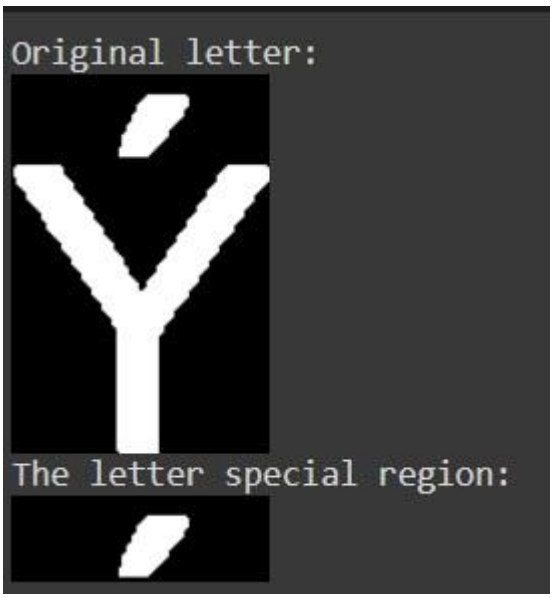
The use of filled characters in an image allows us to conclude more details about the structure of a character and get higher accuracy when using the moments. Therefore, we'll use them for stage 3.4.

3.4 Feature Extraction

For this section will define a region in the image called "Special Region".
The special region is a K*(image.width) region that we crop from the image, while K is a parameter that depends on the size of the font (in our project, K = 35).
This is the top K*(image.width) part of the image.
This is the region where the difference between ***äÄ*** and ***aA*** is found.

We'll use this region in the feature extraction.

Example: Special Region



3.4.1 Moments

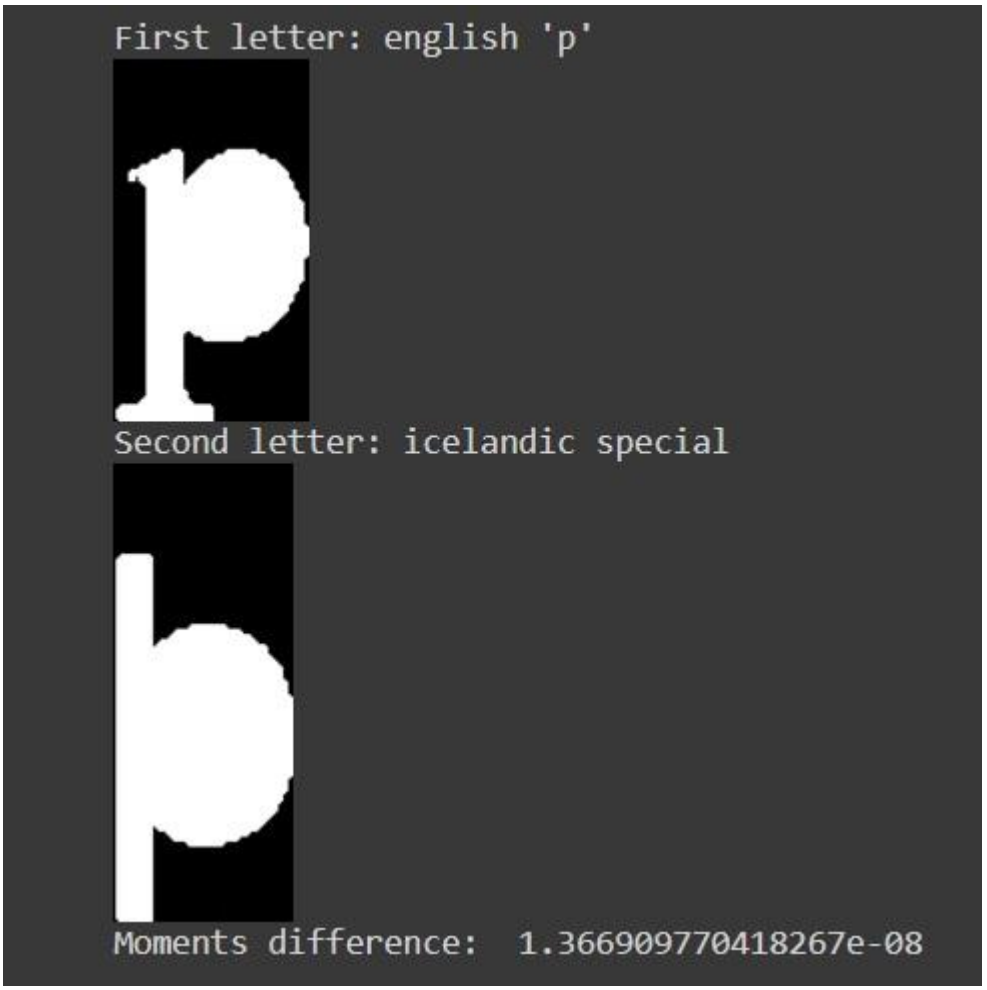
For each filled letter[see 3.3], we'll extract the 2nd and 3rd order normalized centralized moments (NCMs).
In addition, we'll extract the moments only of the special region.

This is done by using ***cv2.moments()*** function as shown in P.S13. [J]

As shown in the introduction, moments can be used as unique shape descriptors for an object. We would like to exploit that and to keep the moments of our filled letters and the field special region.

3.4.2 Skeleton's Moments

This is a surprising part, but a necessary one. If we already have the NCMs of the filled image, then why do we need the moments of the skeleton? Well, this is due to the fact that sometimes the difference in the moments is too small, and because the filled image of the dataset and the filled image of the input are not always in the same size, this is a cause of a problem.

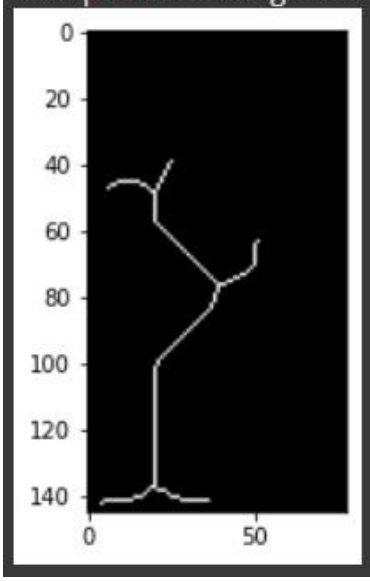


The solution is skeleton. As we know from the lecture notes[I], a small change in the shape creates a very different skeleton. Therefore, if two different images have similar moments, then the skeleton will be the difference. We'll calculate the moments of the skeleton(the same way as in 3.4.1).

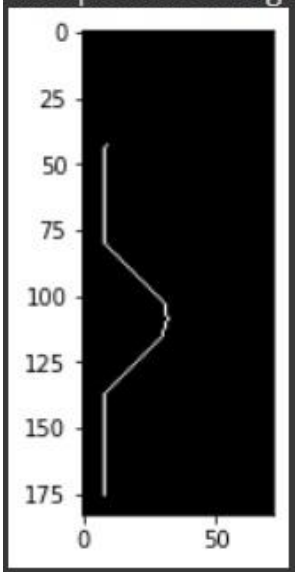
We'll calculate the skeleton using ***skimage.skeletonize()***
We'll do this to both the skeleton of the filled letter and the special region.

Example: Moments of Skeletons and the differences.

```
First letter Skeleton: english 'p'
<matplotlib.image.AxesImage at 0x7f4101b5ed90>
```




```
Second letter Skeleton: icelandic special
<matplotlib.image.AxesImage at 0x7f4101b3ffd0>
```



```
Moments difference:  25.05443477162987
```

3.4.3 Special Region's Image Shape Pxl Ratio

This is the X-factor with the English language. Without this, more often than not an English letter would be classified as an german or Icelandic. In English , this section is a pure zero, so it helps to ensure that an English letter will be classified as English.

This section counts the white pxls in the special region of a binary image (black is background) and divides the result by the total pxls amount.

3.5 Identify the Language

After all the data, both the input paragraph and the dataset go through the feature extraction, the algorithm compares each features vector of the letter from the input to each features vector of the dataset(using SSD), and returns the letter with the minimum difference.

Then, after we mapped each letter in the input to each letter from the dataset, will calculate the result as follow:

We calculated the number of labels, s.t if the label we got is less than 200 then this is English, if it is between 200-300 it is German and for labels bigger than 300 we get Icelandic .
After we get our numbers of labels for each language our rule to identify which language we got is if we get the number of English labels is non zero but the other two are equal or less than 1, this is English. If we get a bigger number of German labels than Icelandic, we get the German language and vice versa (without taking consideration English language as they both have English characters). In case of an equal number of German and Icelandic letters, we'll return German due to the fact that more people speak German (and the 2nd assumption).

Note that the identification of the letter u (with all the Icelandic and German variations) wasn't accurate, so we'll ignore them while calculating the result.

3.6 Weaknesses

Our algorithm have few weaknesses:

- It does not classify numbers.
- The dataset is very small.
- The fonts can only be between 28 and 35.
- Can't correctly classify the letter u (and the variations of u).
- It can't work if two or more languages have the same alphabet.

It's important to know them when using this project in the future.

4. Results

4.1 Language Identification Result

We can see in the table below the results on different test images with different text sizes. we only got an error on English text of characters 29, we had complete success distinguishing between German and Icelandic . There are some errors like in test image german31 we got only 4/5 German characters and found 1 Icelandic although there is not supposed to be. Overall the results are promising, we were able to achieve our goal and with some more work we can improve it even more.

Test Image	Expected German	Expected Icelandic	Result English	Result German	Result Icelandic	Success
english 28	0	0	167	0	1	Yes
english29	0	0	181	0	2	No
english30	0	0	179	0	0	Yes
english31	0	0	182	0	0	Yes
english32	0	0	168	0	0	Yes
english33	0	0	170	0	0	Yes
english34	0	0	170	0	0	Yes
english35	0	0	160	0	0	Yes
german28	5	0	162	5	2	Yes
german29	5	0	154	5	1	Yes
german30	5	0	166	4	1	Yes
german31	5	0	169	4	1	Yes
german32	5	0	167	4	3	Yes
german33	5	0	155	4	1	Yes
german34	5	0	154	4	2	Yes
german35	5	0	145	3	2	Yes
icelandic28	0	15	151	2	14	Yes
icelandic29	0	15	158	3	10	Yes
icelandic30	0	15	153	1	13	Yes
icelandic31	0	14	155	1	13	Yes
icelandic32	0	13	148	1	15	Yes
icelandic33	0	14	155	1	13	Yes
icelandic34	0	13	150	1	13	Yes
icelandic35	0	13	149	1	13	Yes

Special Letters Identification Results:

Note: proof in the Jupyter Notebook.

Test Image	Expected English	Expected German	Expected Icelandic	Result English	Result German	Result Icelandic	Un-Identified
All Special Combine	0	8	16	1	6	16	1

We can see that beside the "u" variations (that we manually excluded), the only letters that were missing were small a with two dots above it, and capital A with two dots above it. Those together recognized as a G. This is a cropping error(i.e., both letters were counted as one). Due to the fact that this pair is unlikely to appear together, we are fine with that.

Expected Formula:
German: 8 German special - 2 “u” variations + 2 intersection with Icelandic = 8
Icelandic: 20 Icelandic special - 2 “u” variations - 2 intersection with Icelandic = 16

5. Conclusions

The project started with the problem of obtaining characters from a given clean text, of course there are advanced libraries that do this like OpenCV ,but we wanted to implement our own algorithm and to cope ourselves with this challenge. After some affortens we were able to build our OCR algorithm and extract the characters as we wanted with very good accuracy. The next challenge was to identify what is the correct label for the extracted character, we thought a simple pca [E] would be good for this job, but we were disappointed to find out it does not classify the labels correctly(probably because of the small database). Although it seems a simple task for the pca our model accuracy was 70% and even after we tried to multiply the same data and got 95% accuracy, the labels were misclassified. Maybe there was overfitting. We had to think of a different approach and after a lot of efforts, we found a direction that could work, which was to calculate the moments of the original letter image and the moments of the skeleton of the letter. With our final approach we were able to get very promising results and to inreach our experience with Computational Vision and Digital Image Processing.

6. Future Work

We believe that a lot of future work can be done in this.
We strongly believe that a future work should include more than 3 languages, and extend the database to more fonts per letter.

In the weaknesses segment we said that this algorithm wouldn't work if two or more languages have the same alphabet (for example, dutch and english). We would like to suggest a way to solve this problem: Optical 2-Characters Recognition (O2CR). The new algorithm will identify 2 characters at once, and by using statistical analysis on the 2-chars and the known statistical analysis of the language it will determine the language based on the result of said analysis.

7. Bibliography

[A] https://dictionary.cambridge.org/dictionary/english/ocr
[B] https://guides.library.illinois.edu/OCR
[C] http://cs229.stanford.edu/proj2015/324_report.pdf
[D] https://www.cis.upenn.edu/~nenkova/Courses/cis430/languageIdentification.pdf
[E] http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/PrincipalComponentAnalysis.pdf
[F] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.4545&rep=rep1&type=pdf
[G] http://www.cs.columbia.edu/~allen/F15/NOTES/computervision.pdf
[H] http://www.inf.u-szeged.hu/~palagyi/skel/skel.html
[I] https://www.cs.bgu.ac.il/~icbv211/wiki.files/ICBV-Lecture-Notes-72-Object-Recognition-2-2D-Shape-Analysis.pdf
[J] https://www.cs.bgu.ac.il/~icbv211/wiki.files/11-01-2021-Tutorial13-ShapeAnalysis.pdf
[K] https://en.wikipedia.org/wiki/Icelandic_language
[L] https://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

8. Appendix

Operation Instruction

1. Open the notebook in Google Colab.
2. Mount Drive - The reading part is based on getting the images in a separate directory for each language. If you change the hierarchy, please keep each language in a separate directory. The order you need as we submitted:ICBV211-123362-Images_ICBV/ICBV211-123362-DatasetForTrain for data set to load.
3. Example - load image locally to the notebook. Examples can be found in ICBV211-123362-Images_ICBV/ICBV211-123362-english or ICBV211-123362-Images_ICBV/ICBV211-123362-german or ICBV211-123362-Images_ICBV/ICBV211-123362-iceland there are images for test language recognition.

4. Run all cells.

5. Calculating the Language - In this cell we can see the result of our language.