AUA
American University *of* Armenia

Image Processing Project

Arman Gasparyan

Handwriting Image Processing

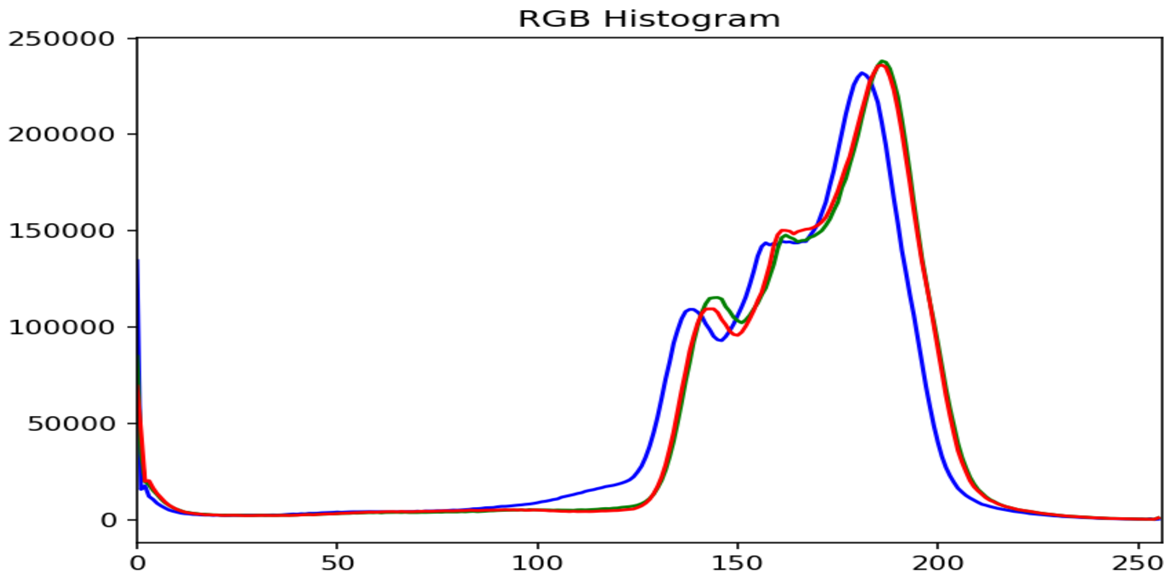
In this project, I am diving into the world of handwriting to figure out how to read and understand it better. I start by organizing scanned images of handwritten pages, giving each page a special code. Then, we use clever methods to get rid of any printed text and focus only on the handwriting. We also look at the size and orientation of the pages, exploring things like straight lines and different regions of the handwriting. Our goal is to make it easier for computers to recognize and understand handwriting, which could be really helpful in analyzing documents and improving how we interact with handwritten information.

Stage 1

Detailed Description:

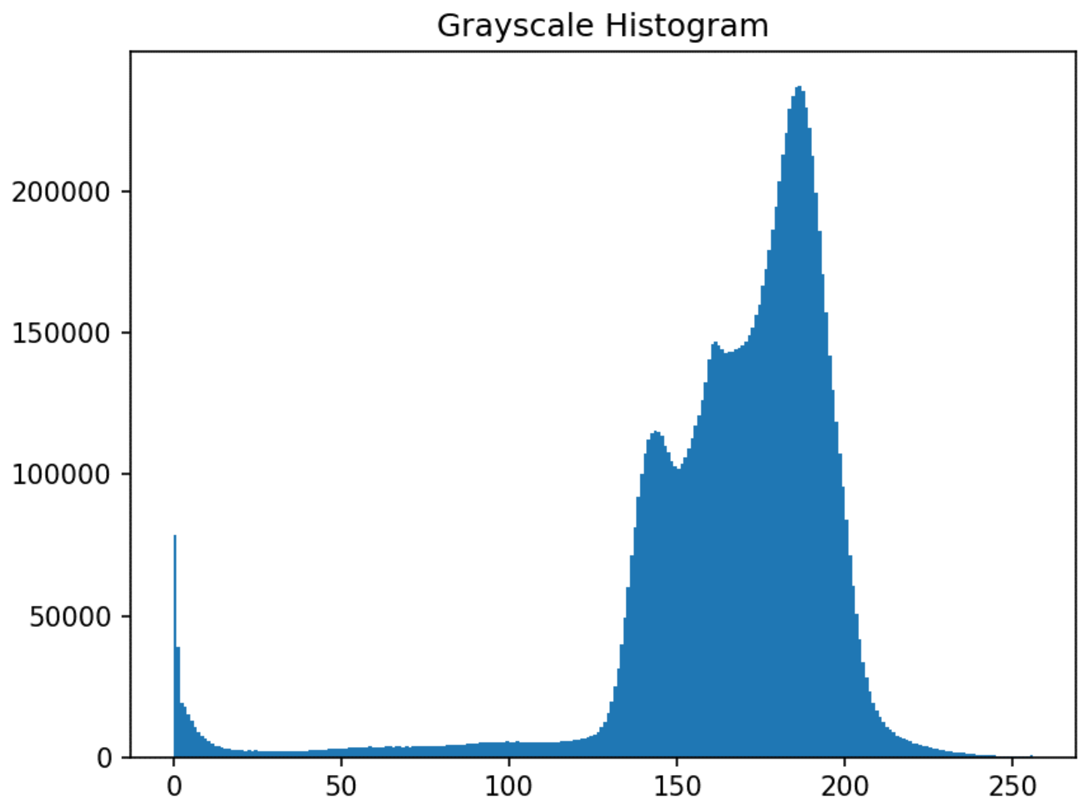
- Reading and Displaying RGB Histogram:

The script begins by reading an image file from the specified path. It then generates and displays the RGB histogram using Matplotlib. The histogram illustrates the distribution of pixel intensities for each color channel (red, green, and blue).



- Reading and Displaying Grayscale Histogram:

The image is converted to grayscale, and its histogram is displayed. Grayscale histograms represent the distribution of pixel intensities in a single channel, providing insight into the overall brightness and contrast of the image.



- Text Removal Using Thresholding and Morphological Operations:

The image is converted to grayscale, and a binary threshold is applied to segment the text from the background. Morphological closing is then performed to further remove noise and connect nearby text regions, resulting in a cleaned binary image.

- Contour Detection and Handwriting Extraction:

Contours are detected on the cleaned binary image using the cv2.findContours function. These contours are then filled, creating a binary mask representing the handwriting. A bounding box is calculated around the handwriting mask, and the region is cropped from the original image.

- Cropping and Thresholding Handwriting Region:

The cropped region containing handwriting is converted to grayscale, thresholded, and displayed as a binary image.

- Displaying Intermediate Results:

The script includes visualizations of intermediate results, such as the cleaned binary image, contours on the cleaned image, and the grayscale handwriting region.

- Libraries Used:

cv2 (OpenCV):

OpenCV is a powerful computer vision library that provides tools for image and video processing. In this script, it is used for reading an image (cv2.imread), performing color space conversions, thresholding, morphological operations, contour detection, and drawing contours.

numpy (NumPy):

NumPy is a fundamental package for scientific computing in Python. It is used for array manipulations, particularly in this script for creating a kernel used in morphological operations.

matplotlib:

Matplotlib is a popular plotting library for creating static, animated, and interactive visualizations in Python. In this script, it is used for plotting and displaying histograms, as well as visualizing the cleaned image, extracted handwriting, and contours.

So, this code provides a comprehensive example of using image processing techniques to isolate and extract handwritten regions from an image. It combines histogram analysis, thresholding, morphological operations, and contour detection to achieve the desired result.

Stage 2

This Python code utilizes computer vision and image processing techniques to evaluate the size and orientation of the page in an image with removed printed text. The script mainly relies on the OpenCV and NumPy libraries for image manipulation and Matplotlib for visualization.

- Techniques and Libraries Used:

1. **Image Loading and Grayscale Conversion**:

The script starts by loading an image with removed printed text using the OpenCV library (`cv2.imread`).

The image is then converted to grayscale (`cv2.cvtColor`) to simplify further analysis.

2. **Edge Detection using Canny Algorithm**:

The Canny edge detector is applied to the grayscale image (`cv2.Canny`).

The Canny edge detector is an algorithm that identifies edges in an image based on gradient intensity changes. It is widely used for edge detection in computer vision applications.

3. **Hough Line Transform for Orientation Detection**:

The Hough Line Transform is employed to detect lines in the edge-detected image (`cv2.HoughLines`).

The Hough Line Transform is a technique used for line detection. In this context, it helps identify dominant lines in the image, representing the orientation of the page.

4. **Orientation Calculation**:

The script calculates the main orientation of the page by finding the median angle of the detected lines.

5. **Gaussian Blur and Sobel Filtering for Font Size Evaluation**:

Gaussian blur is applied to the grayscale image (`cv2.GaussianBlur`). This reduces noise and enhances the performance of the subsequent edge detection.

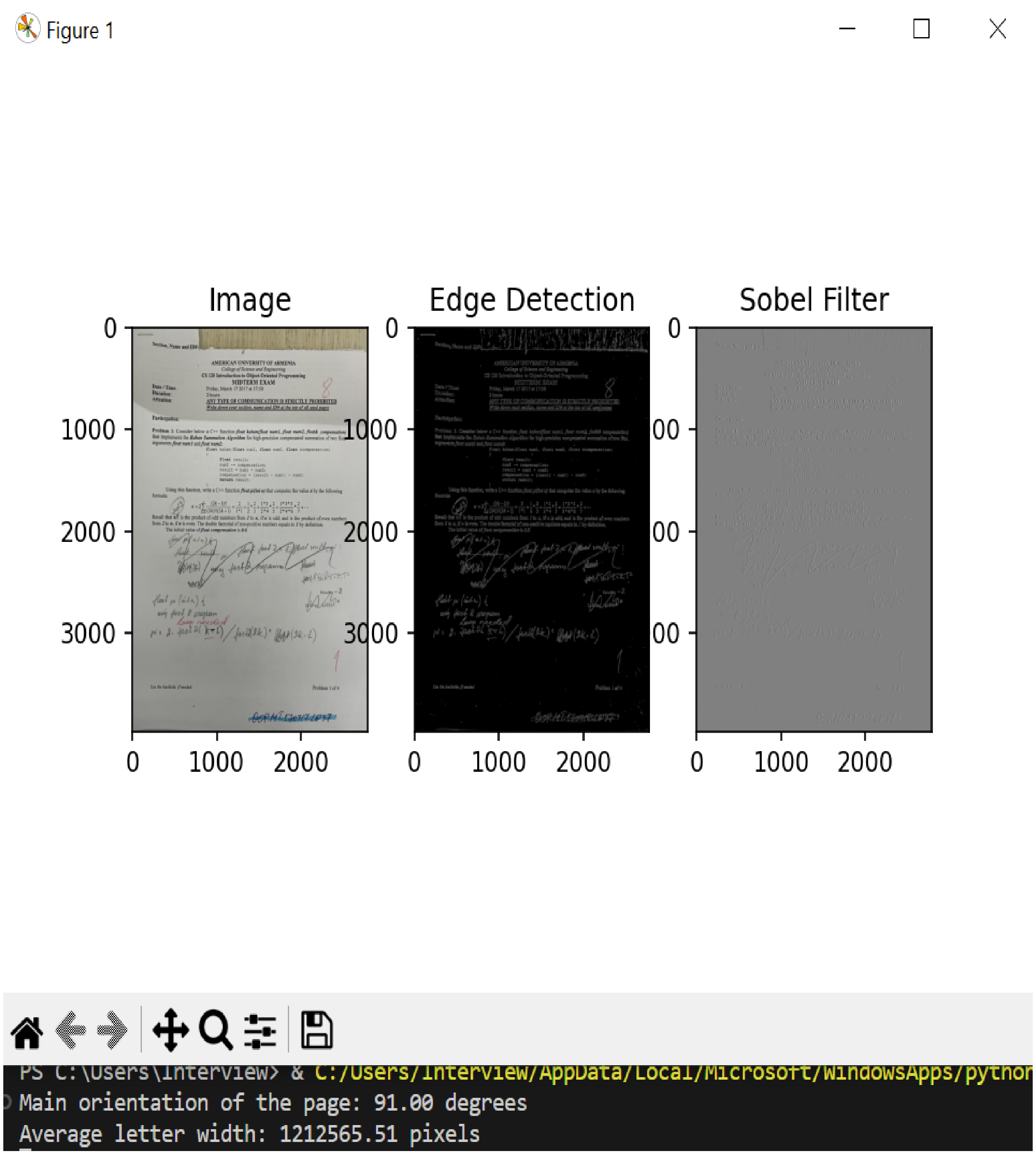
Sobel filtering is used for edge detection along the x-axis (`cv2.Sobel`). Sobel filtering highlights areas of rapid intensity change, helping to identify text boundaries.

6. **Average Letter Width Calculation**:

The script calculates the average letter width based on the Sobel-filtered image. It sums the absolute values of the Sobel-filtered image along the vertical axis and then calculates the mean.

7. **Visualization using Matplotlib**:

The Matplotlib library is utilized to create a subplot to visualize the original image, the result of the Canny edge detection, and the Sobel-filtered image.



```
PS C:\Users\Interview> & C:/Users/Interview/AppData/Local/Microsoft/WindowsApps/pytho
Main orientation of the page: 91.00 degrees
Average letter width: 1212565.51 pixels
```

This script provides insights into the size, orientation, and font characteristics of the page in the input image. The combination of edge detection, Hough Line Transform, and Sobel filtering facilitates the extraction of relevant features for further analysis or normalization in handwritten text recognition tasks. Adjusting parameters, such as thresholds and kernel sizes, can be done to optimize the script for different types of images and applications.

Stage 3

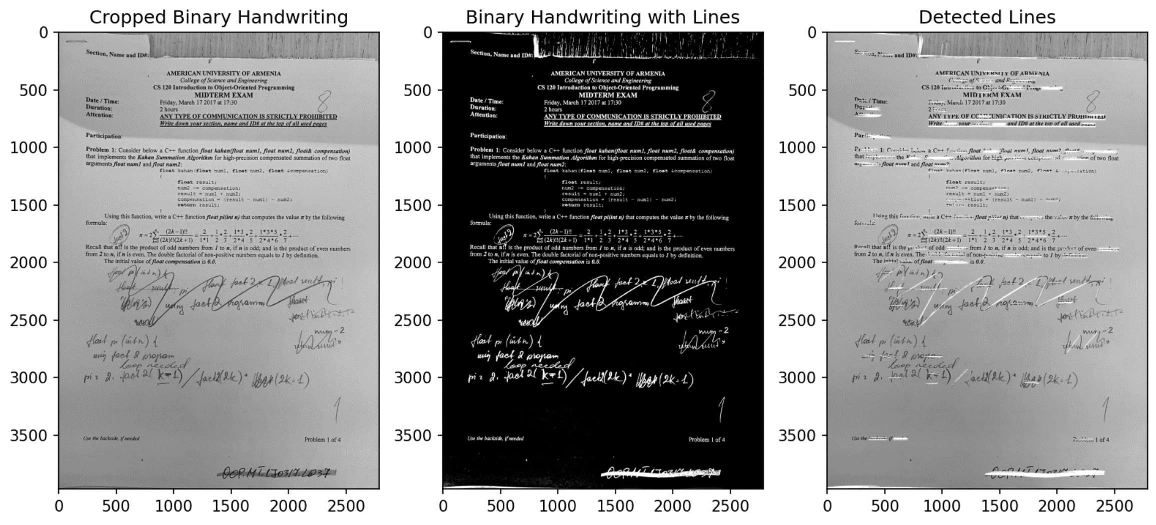
In this stage the Python script utilizes image processing techniques to analyze and visualize structural features in a cropped binary handwriting image. The primary libraries used are OpenCV for image processing and NumPy for numerical operations, while Matplotlib is employed for result visualization.

1. The process begins by loading the cropped binary handwriting image and applying adaptive thresholding to enhance edges and create a binary representation of the text. Subsequently, the probabilistic Hough Transform is utilized to detect lines in the binary image. This technique is advantageous for identifying lines even in the presence of noise and gaps.

2. Detected lines are then overlaid onto a copy of the original image for visualization. The Matplotlib library is employed to display the original cropped binary handwriting image, the binary image with detected lines, and the image with drawn lines in a side-by-side format.

3. Following the visual representation, the script performs an analysis of the detected lines to extract features such as the median angle and median length. These features provide valuable insights into the structural aspects of the handwriting, including characteristics such as baseline, slant, and spacing.

In summary, this code combines image processing techniques and visualization tools to comprehensively analyze and interpret structural handwriting features in a binary image. The use of OpenCV and NumPy facilitates efficient image manipulation, while Matplotlib aids in presenting the results in a clear and informative manner. Adjustments to parameters and further refinement can be applied based on specific use cases and image characteristics.



Stage 4

This stage aims to process cropped handwriting samples using a combination of morphological filters, character separation techniques, and skeletonization to detect baselines and assess label readability. Let's delve into a detailed explanation of each step:

Image Preprocessing:

The code begins by loading the cropped handwriting sample and converting it to a binary image using adaptive thresholding. This initial step simplifies the subsequent analysis by emphasizing edges and structures in the image.

Morphological Operations (Closing):

Morphological closing is employed to fill gaps and smooth out the binary regions. This step is crucial for creating more connected and convenient binary regions, making it easier to identify and separate individual characters.

Contour Detection and Filtering:

Contours are then extracted from the binary image, representing distinct shapes within the handwriting. To separate characters, contours are filtered based on their area. This filtering eliminates small and large contours, focusing on regions likely to correspond to individual characters.

Morphological Operations (Closing and Opening):

Further morphological operations, both closing and opening, are applied to refine the binary image and improve character separation. Closing helps consolidate nearby contours, while opening aids in breaking connections between characters.

Skeletonization:

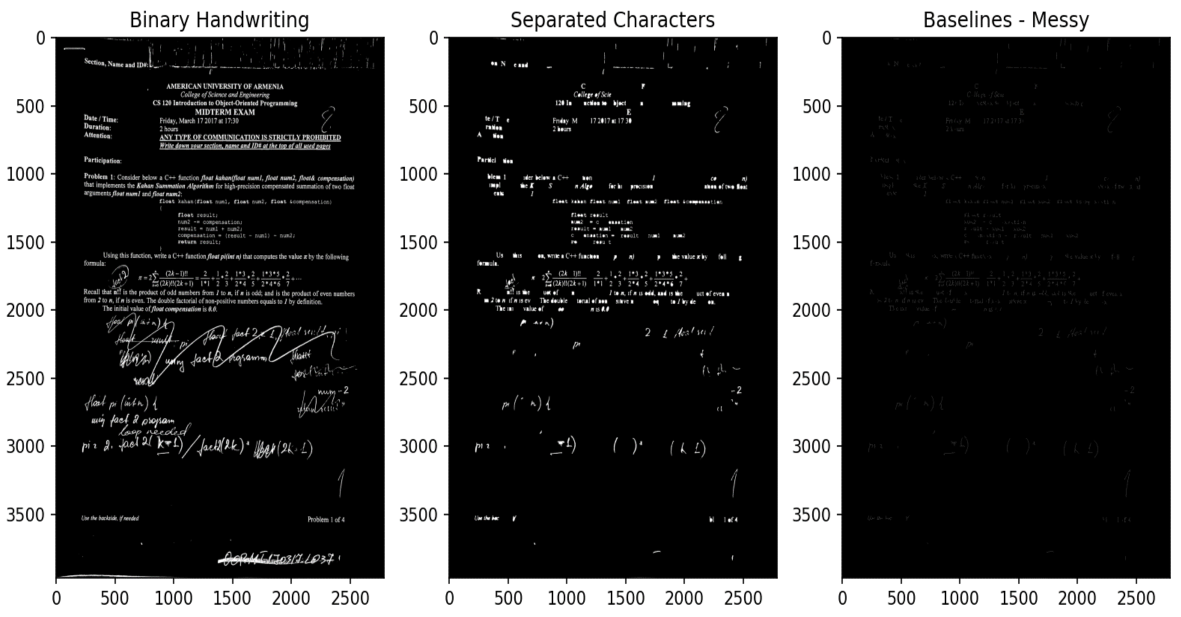
The code utilizes skeletonization to simplify the binary regions further. Skeletonization reduces each character region to a single-pixel-wide representation, emphasizing the baselines and overall structure of the handwriting.

Label Readability Assessment:

To assess label readability, the code evaluates the sum of pixel values in the skeletonized image. If the sum is greater than zero, the label is categorized as "Messy," indicating a more complex writing style. Conversely, a zero sum suggests a "Clear" writing style.

Results Display:

The final processed images, including the original binary handwriting, separated character regions, and the skeletonized representation, are displayed using Matplotlib. This visualization helps in understanding the effectiveness of the applied techniques.



Stage 5

The given Python script is designed to identify specific characters, such as "i," "f," "r," "t," "w," etc., within a binary image. The script utilizes the OpenCV library for image processing and NumPy for numerical operations. Let's break down the key components and their functionalities:

Image Loading and Thresholding:

The script begins by loading a binary image from a specified file path. This image is then thresholded, converting it into a binary representation where pixel values above a certain threshold are set to white, and those below are set to black.

Contour Detection:

The contours of the binary image are identified using the findContours function provided by OpenCV. Contours represent the boundaries of distinct shapes within the image, which, in this case, correspond to individual characters.

Labeling Function:

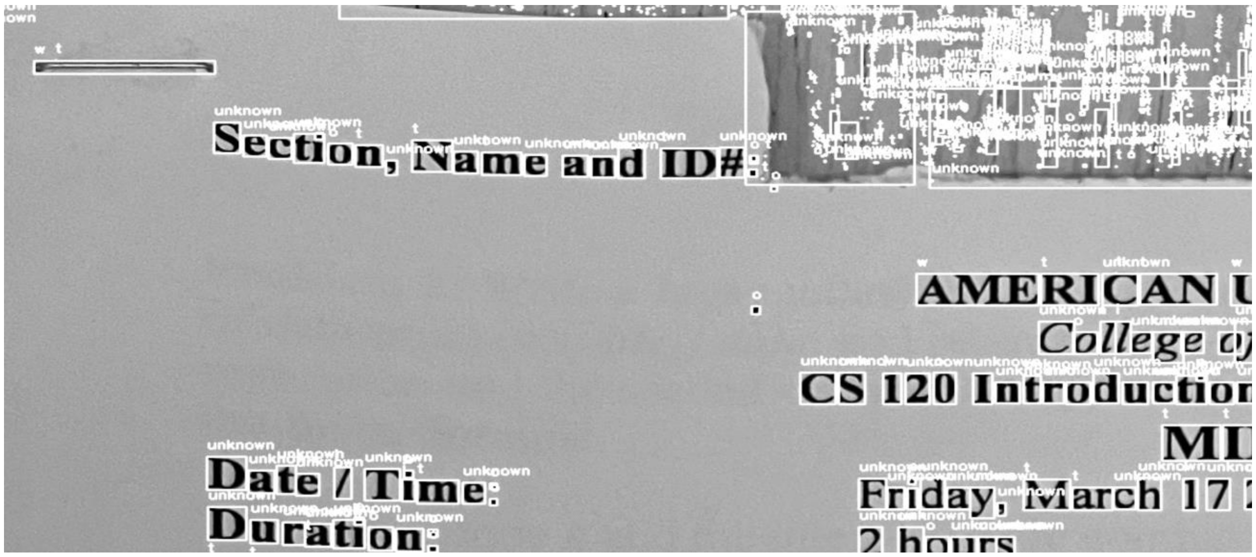
The script defines a function called label_character to analyze the geometric and statistical properties of each detected character. The function calculates the aspect ratio, circularity, and horizontal position of the character based on its bounding box.

Character Analysis and Labeling Loop:

A loop iterates through each identified contour, calling the label_character function to determine the properties of the character. The script then assigns a label ("i," "f," "r," "t," "w," or "unknown") based on these properties and draws a bounding box around the character on the original image.

Displaying Results:

The final step involves displaying the modified image with bounding boxes and character labels. The script utilizes OpenCV's imshow function for visualization.



Unfortunately, I was not able to reach the desired result in this stage.

Conclusion

To sum up, this project comprises multiple stages aimed at automated handwriting analysis. Beginning with the organization of scanned images, each stage addresses distinct challenges in document processing. Notably, it includes automated text removal, page feature evaluation, and the application of advanced techniques such as the Hough Transform and morphological filters. The final stage involves character detection and labeling based on geometric and statistical properties of binary regions. Throughout, the project integrates methods for efficient storage, text extraction, and character identification, demonstrating a comprehensive approach to enhancing document understanding through image processing and pattern recognition.

External Sources:

- “Using Handwriting Features to Predict Student Performance in IT and Engineering Domain” Capstone Project, Naira Khachatryan, May 23, 2022
- Image Processing in Python, “Geeks for Geeks” https://www.geeksforgeeks.org/image-processing-in-python/
- ChatGPT https://chat.openai.com/ (In order to learn Python syntax and libraries)