CAIRO UNIVERSITY
FACULTY OF ENGINEERING
(CHS)

CMPS446: Image Processing

Report - Team 2

PREPARED BY	Ahmed Yasser	ID	1200108
PREPARED BY	Amr Shams	ID	1190563
PREPARED BY	Malek Essam	ID	1200479
PREPARED BY	Ossama Nasser	ID	1200790
SUBMITTED TO	Eng Muhab		

Table Of Content

Project Description **3**

Boundaries & Limitations **3**

Symbols to detect 4

Preprocessing **5**

Initial Attempt: Basic Preprocessing 5

Advanced Noise Removal 5

Edge and Line Preservation 6

Deskewing and Orientation Detection 6

Final Preprocessing Pipeline 7

Segmentation **8**

1. Initial Attempt (StaffLineRemoval Class)
2. Removing Staff Lines 8
3. Current Approach 8

Classification **9**

1. Initial Approach (HOG + HU Moments + KNN) 9

3. Second Approach (SIFT + KMeans + SVM) 9

5. Final Approach (HOG + KNN on Cleaned Dataset) 9

6. Pre Trained Neural Network (NN): 10

Tools and Important Notes 10

PostProcessing **11**

Parser 11

Some Results & Examples **12**

Workload Division **14**

Project Description

This project focuses on the development of an advanced image processing pipeline for musical symbol recognition. The goal is to automate the extraction and classification of musical symbols from images of sheet music, enabling their digital representation for further processing or analysis. Sheet music images typically contain various symbols, staff lines, and other elements that need to be accurately segmented and classified to interpret the underlying musical content.

The preprocessing phase plays a critical role in enhancing the input image quality, ensuring that symbols are clearly visible and separable from the background. Techniques such as grayscale conversion, noise reduction, and thresholding are employed to simplify the image and improve the visibility of the musical symbols. These preprocessing steps prepare the image for more advanced operations, including segmentation, where symbols are isolated from the staff lines and other elements, and classification, where each symbol is identified.

Given the complexity of musical notation, the pipeline employs a combination of traditional computer vision techniques and machine learning approaches. Initial efforts focused on basic methods like thresholding and morphological operations, while more advanced approaches, such as adaptive thresholding, edge detection, and deskewing, were later explored to handle varying lighting conditions and align skewed images.

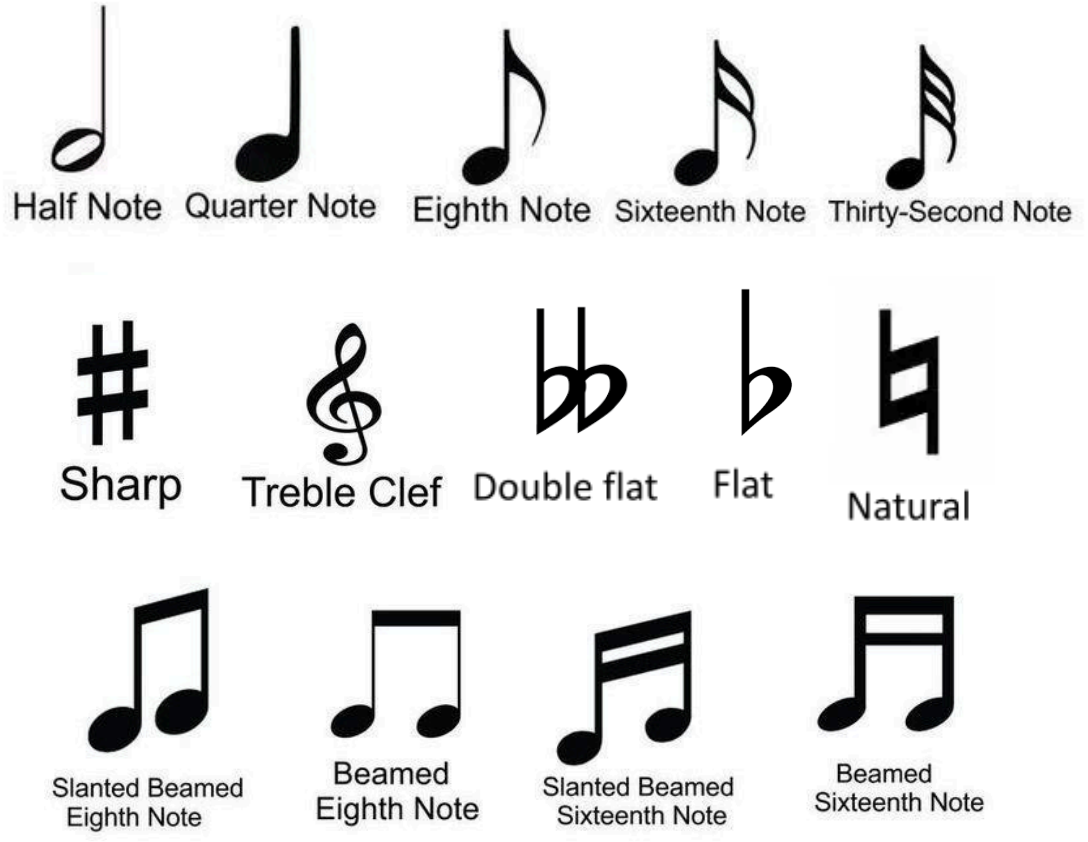
This project aims to create a robust and efficient pipeline for sheet music analysis, improving the accuracy of symbol recognition and reducing the manual effort required for music transcription. By leveraging both image processing and machine learning, it paves the way for automatic transcription systems that can significantly aid musicians, composers, and music analysts.

Boundaries & Limitations

Symbols to detect:

Regarding the symbols that we would be able to detect we were limited by the availability of labeled dataset of a reasonable size to be able to train our model and achieve appealing results. Then after detecting these symbols we use the staff lines to get the notes output itself and display it to the user.

In addition regarding to the images that our project can deal with, the project achieves the best results with images of music notes sheets if it is scanned however, regarding cases like having a picture of the sheet by phone or handling handwriting it would work but with less accuracy depending on the preprocessing needed on the image.



Preprocessing

Effective preprocessing is crucial for preparing images for segmentation and symbol recognition. We experimented with various techniques to enhance the quality of input images and improve downstream processing. Below is a summary of our approaches and findings.

Initial Attempt: Basic Preprocessing

Objective: Remove noise and enhance contrast for better symbol visibility.

Techniques Used

- **Grayscale Conversion:** Simplified image data by reducing it to a single channel.
- **Median Filtering:** Reduced salt-and-pepper noise.
- **Global Thresholding (Otsu's Method):** Applied a binary threshold to segment foreground and background.

Process

1. Converted images to grayscale for simpler processing.
2. Applied median filtering to smoothen noise while preserving edges.
3. Used Otsu's thresholding to binarize the image.

Findings

- Median filtering was effective for removing small noise but caused slight blurring.
- Otsu's thresholding struggled with uneven illumination and overlapping elements.
- Limited improvement in symbol clarity.

Advanced Noise Removal

Objective: Address uneven illumination and retain symbol details.

Techniques Used

1. **Gaussian Filtering:** Reduced high-frequency noise while preserving low-frequency features.
2. **Adaptive Thresholding:** Applied a locally calculated threshold to handle varying lighting conditions.

Process

- Smoothened the image using Gaussian filtering ($\sigma = 1.0$).
- Divided the image into smaller regions and applied adaptive thresholding for each.

Findings

- Adaptive thresholding significantly improved symbol visibility under uneven lighting.
- Gaussian filtering removed noise effectively but slightly blurred thin lines.

Edge and Line Preservation

Objective: Remove staff lines and enhance edges for symbol isolation.

Techniques Used

1. **Morphological Operations:** Applied opening (erosion followed by dilation) to remove small noise and thin lines.
2. **Canny Edge Detection:** Detected edges while preserving sharp transitions in pixel intensity.
3. **Local Histogram Equalization:** Enhanced contrast for faint symbols.

Process

1. Removed small elements using morphological operations (structuring element size 3x3).
2. Detected edges using the Canny algorithm (low threshold=50, high threshold=150).
3. Equalized local contrast in regions to emphasize symbols.

Findings

- Morphological operations worked well for line removal but required careful parameter tuning.
- Canny edge detection preserved symbol boundaries effectively.
- Histogram equalization boosted faint symbols but increased noise in certain areas.

Deskewing and Orientation Detection

Objective: Correct skewed images and ensure proper orientation.

Techniques Used

1. **Hough Transform:** Detected lines to calculate the skew angle.
2. **Orientation Detection:** Determines the overall orientation of the image.
3. **Rotation:** Rotated the image to correct the skew.
4. **Perspective Transformation:** Corrected the perspective of the image.

Process

1. Detected lines using the Hough Transform and calculated the skew angle.
2. Determine the orientation angle using the minimum area rectangle method.
3. Calculated the deskew angle based on the Hough angle and orientation angle.
4. Rotated the image to correct the skew.
5. Applied perspective transformation to correct any distortions.

Findings

- Deskewing significantly improved the alignment of symbols.
- Orientation detection ensured that the images were correctly oriented.
- Perspective transformation corrected distortions, enhancing symbol clarity.

Final Preprocessing Pipeline

Objective: Create a robust pipeline for consistent preprocessing across images.

Techniques Used

1. **Binarization:** Combined global and adaptive thresholding for flexibility.
2. **Denoising:** Applied Gaussian filtering for smooth noise reduction.
3. **Contrast Enhancement:** Used histogram equalization to boost symbol clarity.
4. **Deskewing and Orientation Detection:** Corrected skew and ensured proper orientation.
5. **Perspective Transformation:** Corrected distortions for better symbol clarity.

Process

1. Converted images to grayscale.
2. Applied Gaussian filtering ($\sigma = 1.0$).
3. Binarized the image using adaptive thresholding for regions with uneven lighting.
4. Enhanced contrast using histogram equalization.
5. Used morphological operations to remove thin lines while preserving symbols.
6. Detected lines using the Hough Transform and calculated the skew angle.
7. Determine the orientation angle using the minimum area rectangle method.
8. Calculated the deskew angle based on the Hough angle and orientation angle.
9. Rotated the image to correct the skew.
10. Applied perspective transformation to correct any distortions.

Findings

- The pipeline provided consistent preprocessing results across different images.
- Enhanced both symbol clarity and edge sharpness.
- Improved input quality for downstream segmentation tasks.

Segmentation

We tested different methods to handle or crop symbols from images:

1. **Initial Attempt (StaffLineRemoval Class):**
   The StaffLineRemoval class was used to remove staff lines while preserving symbols. It relied on Run-Length Encoding (RLE), morphological operations, and projection techniques to calculate line thickness and spacing, remove lines, and clean the image. However, this method wasn't suitable for our problem since we initially treated lines as noise, not realizing their importance in affecting results.

2. **Removing Staff Lines:**
   - Applied Gaussian filtering to smooth the image.
   - Found contours and grouped them into seven lines.
   - Used a local histogram to crop white spaces.
   - Applied HoughLines and morphological operations for segmentation.
     This approach yielded poor accuracy, and HoughLines was ineffective for symbol recognition.

3. **Current Approach:**
   - Detected lines using a local histogram for each row; rows with >80% histogram width were considered real lines.
   - Calculated thickness and spacing by grouping lines.
   - Removed lines using erosion with a simple structuring element.
   - Iterated through the image to find structures and bounding coordinates (x-min, y-min, x-max, y-max) for cropping and extracting information.

This naive solution was specifically designed for our problem, balancing simplicity and performance, while more accurate but slower gray-image methods were deemed unnecessary.

Classification

1. **Initial Approach (HOG + HU Moments + KNN):**
    - **Features:** We extracted Histogram of Oriented Gradients (HOG) features and added Hu moments to handle rotated images. These features were fed into a K-Nearest Neighbors (KNN) classifier.
    - **Performance:** On a dataset of 27,985 rotated and handwritten images, the model achieved 42.57% accuracy. Increasing the dataset size to 80,124 images, divided into 25 classes, improved accuracy to 61%. However, when applying this pipeline to a large dataset (420,000 images and 210 classes), accuracy dropped to 51%, as KNN struggled with the high number of classes and large datasets.
2. **Conclusion:** The combination of HOG + Hu moments with KNN was sensitive to dataset size and the feature quality played an important role in classification accuracy.

3. **Second Approach (SIFT + KMeans + SVM):**
    - **Features:** We used Scale-Invariant Feature Transform (SIFT) to extract features, followed by KMeans clustering to generate a frequency array of the features. This frequency array was used to train a Support Vector Machine (SVM).
    - **Performance:** On the large handwritten dataset, this method achieved 62% accuracy. On the smaller dataset, accuracy dropped to 53%.
4. **Conclusion:** SIFT features did not perform well due to similarities between the symbols in the two datasets, suggesting that SIFT is not ideal for this problem.

5. **Final Approach (HOG + KNN on Cleaned Dataset):**
    - **Dataset Cleaning:** We augmented the dataset to add more samples of already existing images with modifications, having 82,234 images with 17 labels. We also removed irrelevant symbols and tried to add symbols like barlines and a/32 as they were still very rare in the dataset.
    - **Performance:** With this dataset, the HOG + KNN pipeline achieved accuracy of 95% on 20% sample test data from the dataset. (80% training, 20% test)

6. **Pre Trained Neural Network (NN):**
    - We used sklearn MLP pre-trained model , to compare its results with our KNN implementation.
7. **Observation:** We noticed that the model handles rare symbols in a much better way and it doesn't get biased toward more repetitive symbols in the training set.

KNN + HOG Results

	precision	recall	f1-score	support
a_1	0.98	0.99	0.99	377
a_16	0.83	0.83	0.83	372
a_2	0.79	0.95	0.87	486
a_32	0.73	0.69	0.71	193
a_4	0.90	0.79	0.84	564
a_8	0.95	0.92	0.93	438
barline	0.98	0.97	0.98	1356
chord	0.93	0.97	0.95	212
clef	0.99	0.98	0.98	661
.	0.99	0.97	0.98	891
&&	0.00	0.00	0.00	4
##	0.98	0.95	0.97	175
&	0.95	0.99	0.97	765
	0.97	0.99	0.98	704
#	0.99	0.98	0.99	931
\meter<"4/2">	1.00	0.25	0.40	4
\meter<"4/4">	0.99	0.86	0.92	91
...				
accuracy			0.95	8224
macro avg	0.88	0.83	0.84	8224
weighted avg	0.95	0.95	0.95	8224

MLP Classifier

	precision	recall	f1-score	support
a_1	1.00	1.00	1.00	377
a_16	0.97	0.97	0.97	372
a_2	0.99	1.00	0.99	486
a_32	0.95	0.93	0.94	193
a_4	0.99	0.99	0.99	564
a_8	1.00	0.99	1.00	438
barline	1.00	1.00	1.00	1356
chord	1.00	1.00	1.00	212
clef	0.99	1.00	1.00	661
.	1.00	1.00	1.00	891
&&	0.80	1.00	0.89	4
##	1.00	1.00	1.00	175
&	1.00	0.99	1.00	765
	0.99	1.00	1.00	704
#	1.00	0.99	1.00	931
\meter<"4/2">	1.00	1.00	1.00	4
\meter<"4/4">	1.00	1.00	1.00	91
accuracy			0.99	8224
macro avg	0.98	0.99	0.99	8224
weighted avg	0.99	0.99	0.99	8224

Tools and Important Notes

- **Training Platform:** We used Kaggle to train the model, utilizing a parallelized KNN model on a GPU for faster computation.
- **Testing:** The model was tested on 20% of the dataset to evaluate its performance.
- **KNN Implementation:** The KNN model was parallelized for efficient training on large datasets, taking advantage of GPU acceleration to speed up the process.

PostProcessing

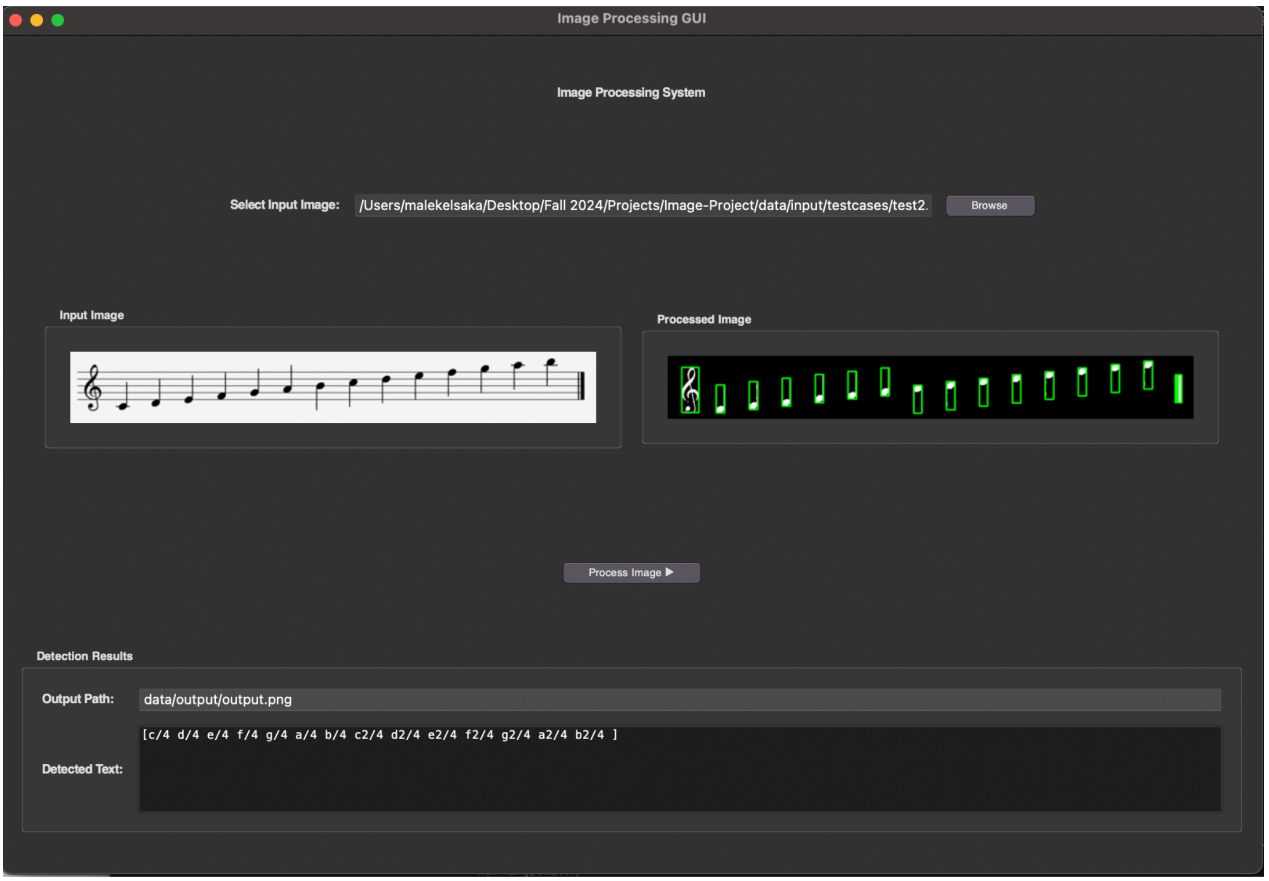
Since our model only provides 18 labels/classes, we use naive methods to detect the remaining symbols. Each class is handled with a specific approach:

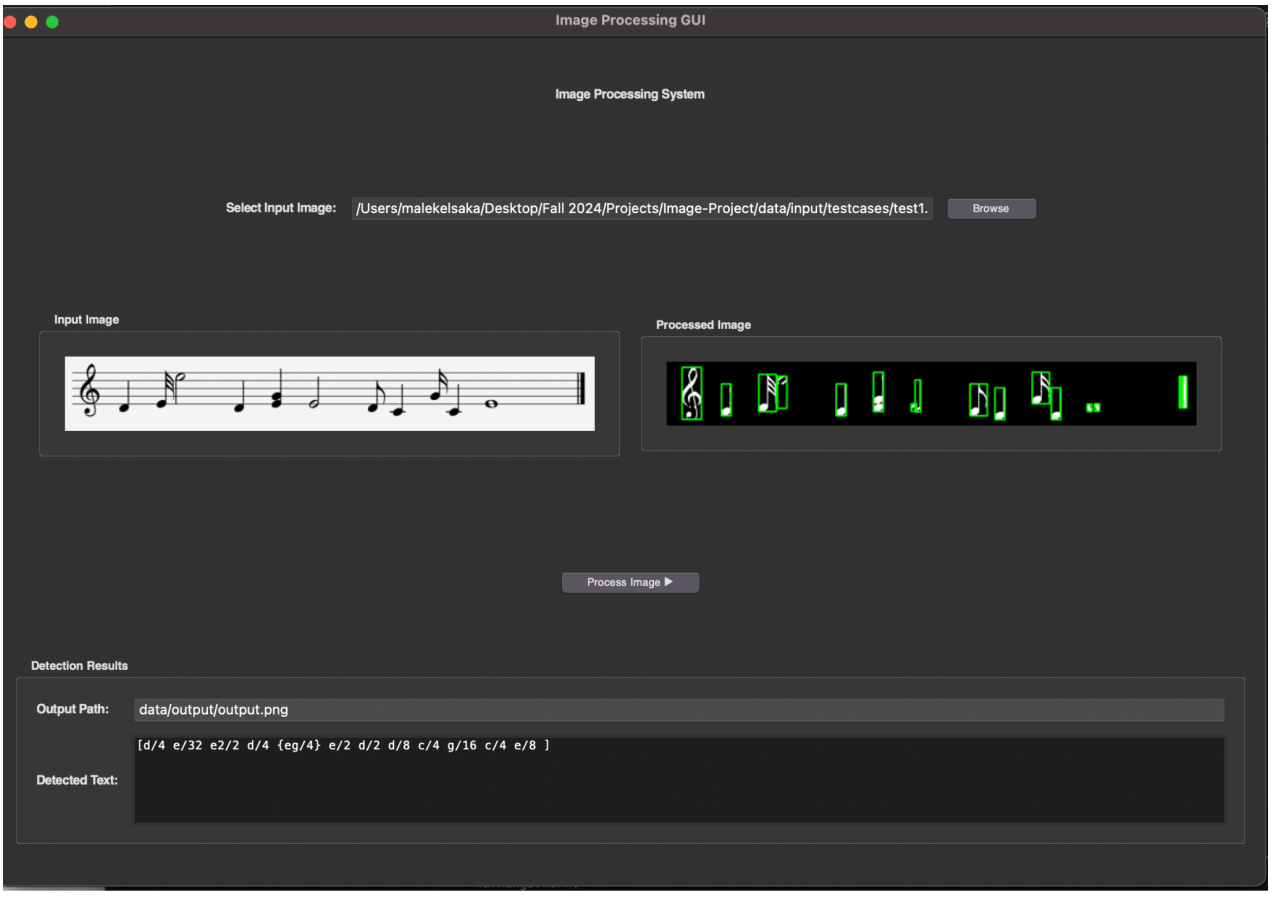
- **Beamed Notes:** We apply an improved method where morphological opening with a horizontal kernel removes the stem. A thin vertical window running horizontally helps us detect the tallest black run, representing the note head. We save the length, start, and end positions, filter out offsets caused by the beam, and extract the maximum from each group to determine the note head's start and end.
- **Number of Beams:** To estimate the number of beams, we use a method similar to the one used for calculating stafflineHeight and staffspaceHeight.
- **Chords:** We calculate the x-projection of the image. If the peak is in the middle, we infer note heads on both sides of the stem. The stem is then removed using the method applied for beamed notes. The number of note heads is determined by the height of the remaining shape, and the positions are sorted alphabetically.
- **Other Notes:** For other symbols, we perform a series of x and y projections.

Parser

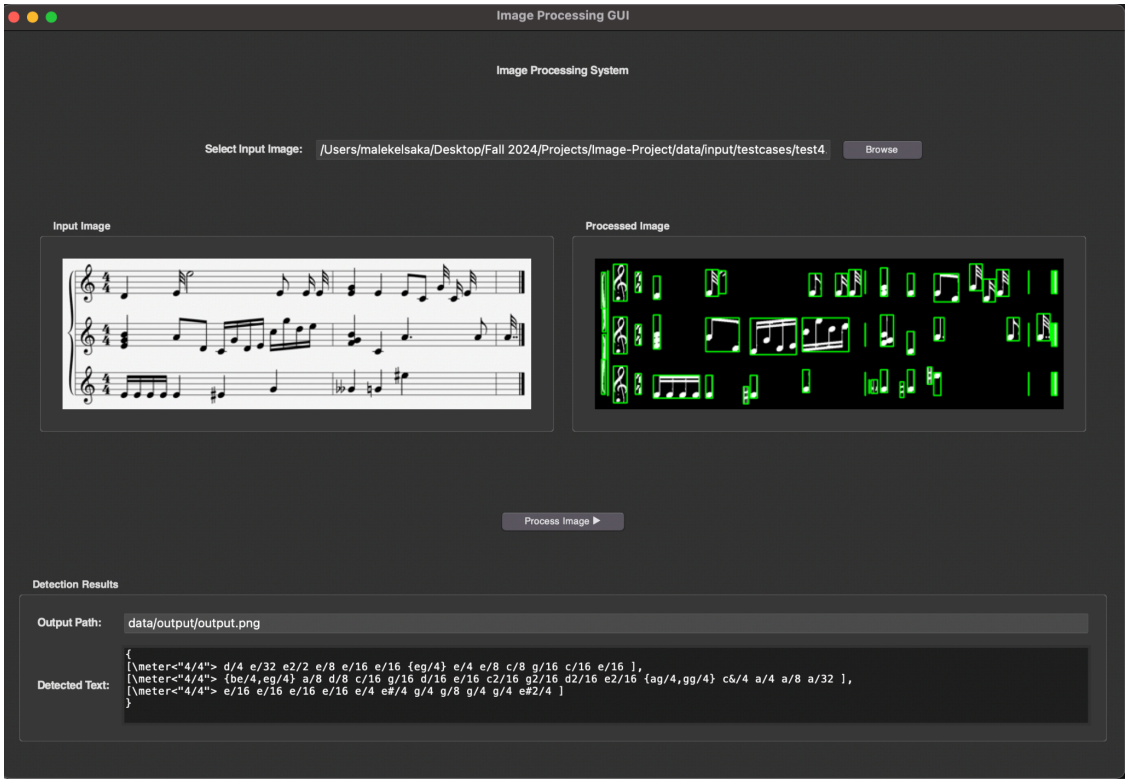
Each musical symbol has a specific meaning based on its position relative to the staff lines, and it is given a unique name. We use a naive method to detect both the symbol and its relevant staff line information. This approach helps us map each symbol to its corresponding meaning by analyzing its position relative to the staff lines. The parser identifies the symbol and retrieves the necessary staff line information, allowing us to accurately classify and interpret the symbol's meaning.

Some Results & Examples

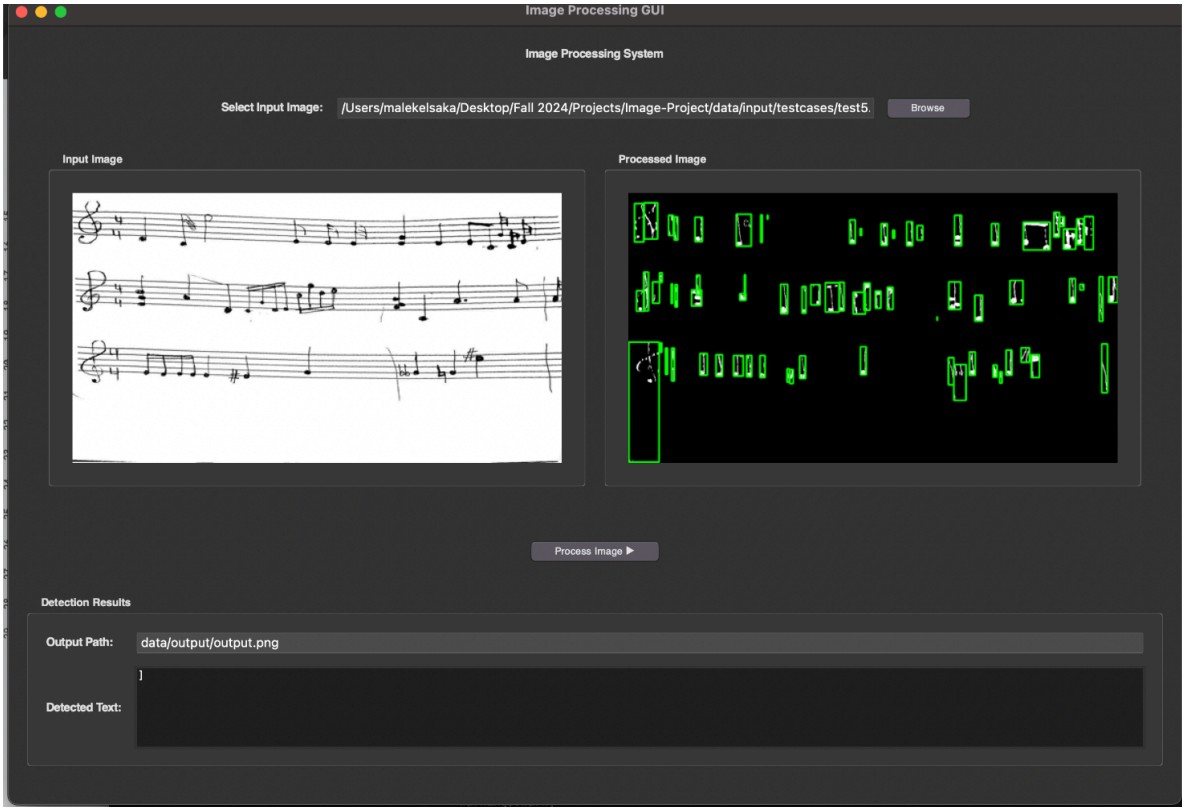




Some Results & Examples



Down below is an example for a fail case with handwritten notes and messy lines, although some of the symbols are segmented, the parser couldn't interpret the symbols at the post processing phase.



Workload Division

Team Member	Workload Division
Ahmed Yasser	25%
Amr Shams	25%
Malek Essam	25%
Ossama Nasser	25%