CSCE 5222 Feature Engineering

Project Plan/Report

Group 01

Members: Jack Duffield, Mustafa Syed, Troy Krupinski

Link to GitHub: [GitHub](https://github.com/TroyKrupinski/CSCE5222_Project) (Current main file: NewMain.py)

1. Problem statement In this project we are developing a program in Python meant to automatically identify acceptable components based on visual pattern properties from a sample set of images supplied by a client. These components considered “good” have a clear stripe or cross patterns will be classified as good, and otherwise are considered bad due to either over or under-exposure - which can obscure the pattern. To better apply our methodology and refine our approach, we reference recent work by Chen et al. (2021), who present a robust framework for visual inspection processes. The study emphasizes techniques like adaptive thresholding and frequency analysis, which are crucial techniques we will be utilizing throughout the project.
2. Data used

**State the data used in this project including ground truth for evaluation:**

The data we employ in this project is a sequence of 20 grayscale images, each measuring 1000 pixels by 1000 pixels. This set of images is a set of 27 components with each image having different external lighting to simulate real-world scenarios. Each element in the images contained in this dataset has different surface properties that are apparent in the form of stripes, cross patterns, or the lack thereof.

**Expected Ground Truth:**

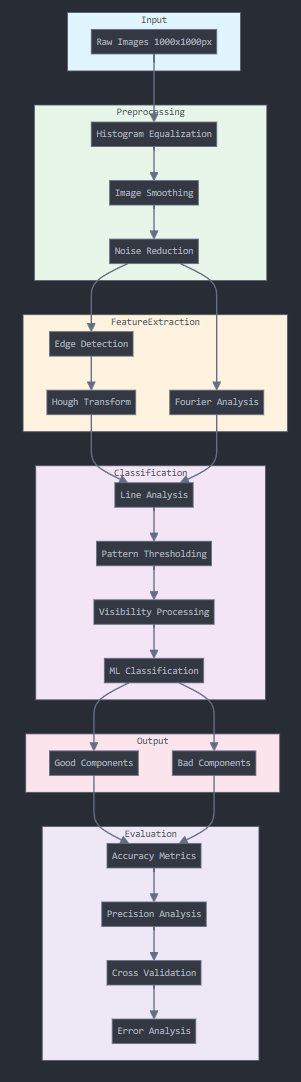
The ground truth for this dataset is established by manually annotating each component based on the visibility and clarity of its surface patterns. The ground truth will be defined by creating a simple shape which geometrically exactly replicates the pattern we’re classifying. Components with distinct stripe or cross patterns are labeled as "good," whereas those without such patterns are labeled as "bad." This manual annotation serves as a benchmark for evaluating the performance of the automated classification system developed in this project.

For each labeled component, the ground truth includes the bounding box coordinates (x, y, width, height) to validate the localization accuracy. This ensures that the algorithm correctly identifies and isolates the target component.

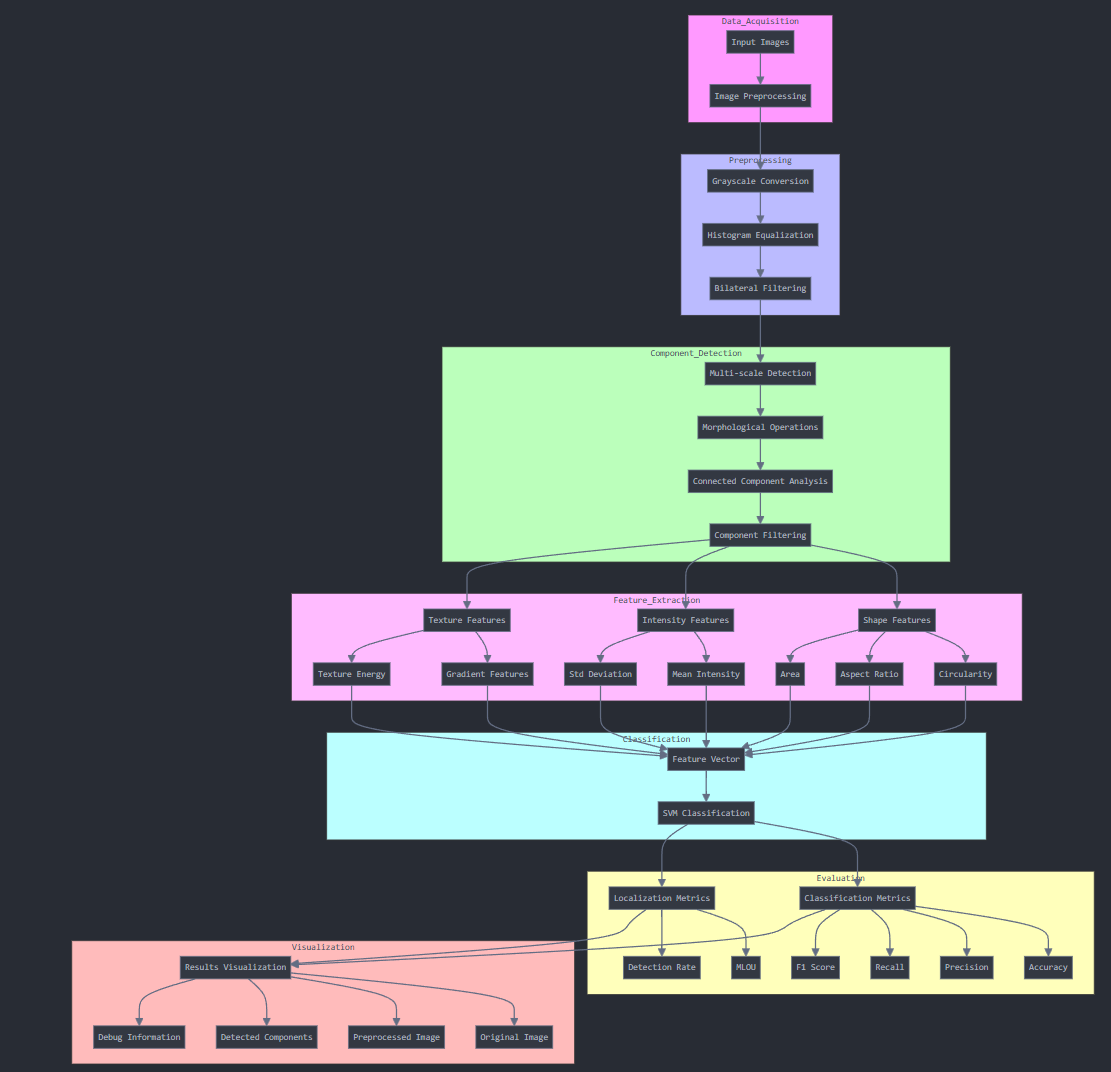
ground*truth\_boxes = [ [(100, 100, 50, 50), (200, 200, 50, 50)] # Example for each image for* in range(20) ]

**IoU Threshold for Detection Validation:**

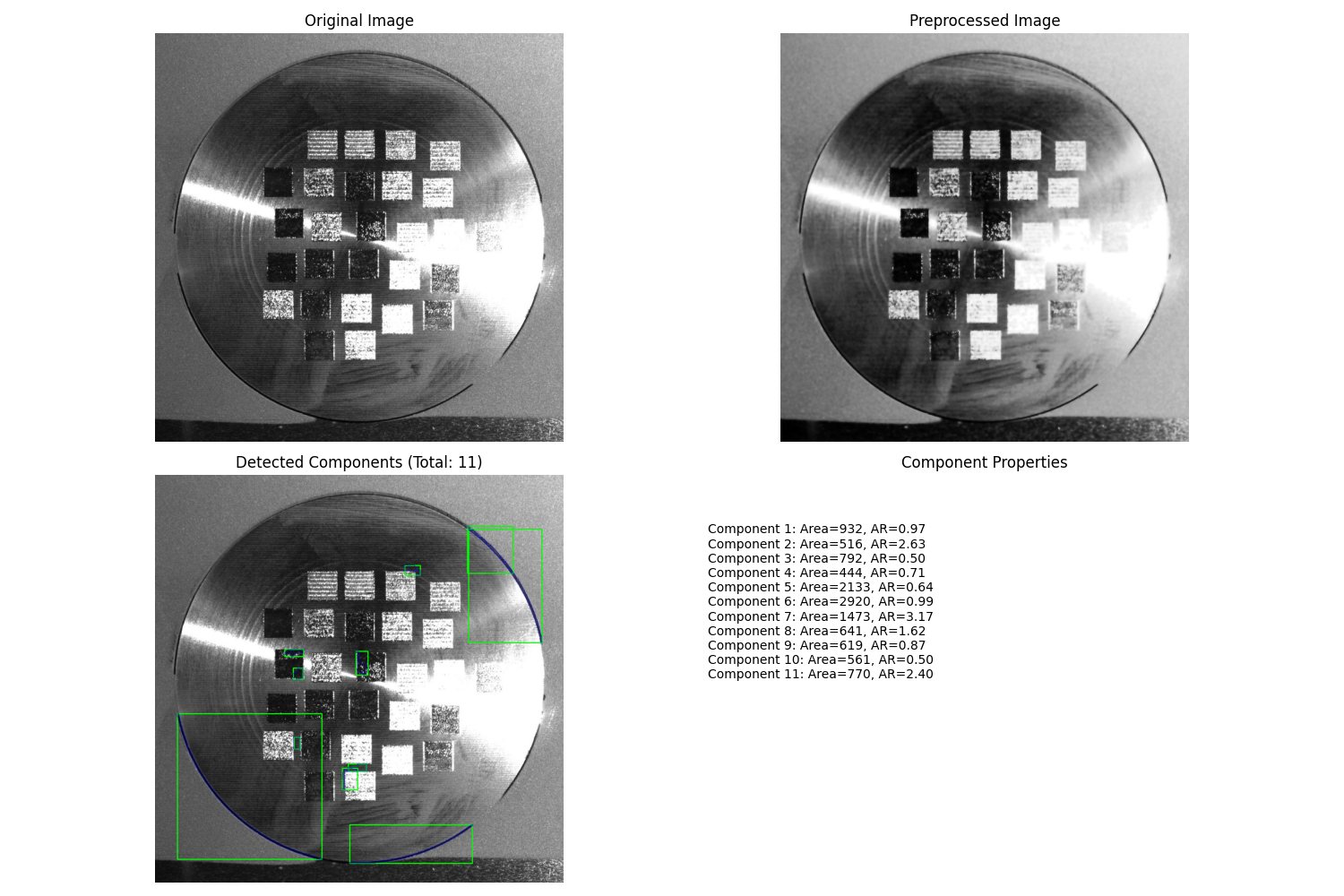
The Intersection over Union (IoU) metric is used to compare the predicted bounding box of each component with the ground truth bounding box. A high IoU score (e.g., IoU ≥ 0.5) indicates accurate localization, as that is the score we currently use / are experimenting with.



1. Updated Method



This project will be implemented using Python and a multitude of libraries; leveraging libraries such as OpenCV for image processing, NumPy for numerical operations, Scikit-learn for machine learning algorithms, and Pandas for data manipulation.

1. **Preprocessing**
   1. Convert images to grayscale: Images are converted to grayscale if they are not already in grayscale format.
   2. Histogram Equalization: This step standardizes contrast and lighting across images, mitigating issues caused by overexposure or underexposure in the dataset.
   3. Noise Reduction: The bilateral filter is applied for noise reduction while preserving edges, which ensures that critical pattern details remain intact. (Note: Gaussian and median filtering were previously considered but replaced by bilateral filtering in the updated code.)
2. **Component Detection (updated Nov. 14th)**
   1. **Preprocessing for Component Isolation**Each image undergoes preprocessing, which includes:
      1. Histogram Equalization to standardize lighting across images.
      2. Noise Reduction using a bilateral filter, which effectively preserves edges while reducing noise, ensuring that potential patterns are not disrupted by the filtering process.
      3. 
   2. **Multi-Scale Detection**To improve detection accuracy and compensate for variations in component size across images, the algorithm uses multiple image scales. Images are resized iteratively, and each scale is processed individually:
      1. Adaptive Thresholding converts each scaled image to a binary format, making patterns easier to identify.
   3. **Morphological Operations** such as opening help to remove small noise artifacts and enhance the detection of larger structures that may contain the target patterns.
      1. The binary images undergo connected component analysis, which groups pixels that are likely part of the same component. The analysis returns statistics, including bounding box coordinates, aspect ratio, and area for each component. These values help filter out components that are too small, too large, or don’t match the aspect ratio criteria for target components.
   4. **Filtering Based on Component Properties  
      Detected components are filtered based on specific properties:**
      1. Area and Aspect Ratio Thresholds ensure that components meet predefined size and shape criteria.
      2. Scaling Corrections adjust bounding box sizes and locations to fit the original image dimensions, ensuring accurate component locations regardless of scale.
      3. The component is then ready for the feature extraction stage if it passes all filtering criteria.

1. **Feature Extraction**

**Stripe / Cross Pattern Detection:** The use of methods like the Canny edge detector as well as the Sobel filter to identify linear/intersecting line patterns. To further refine our feature extraction techniques, we take from Li et al. (2020), whose study highlights key techniques used in our methodology such as edge detection, adaptive filtering, and feature extraction. All of which are critical for our end goal of classification.

**To further analyze** this, we will use methodologies like the **hough line transform**, which can be used to detect and classify lines within specific components. This can further help identify if patterns resemble either stripes or crosses.

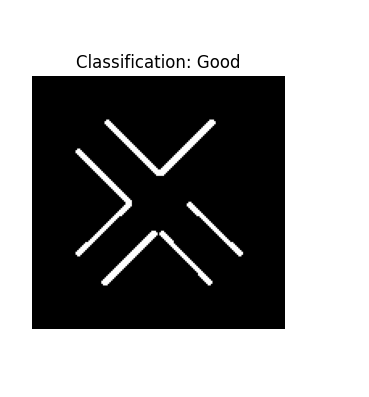
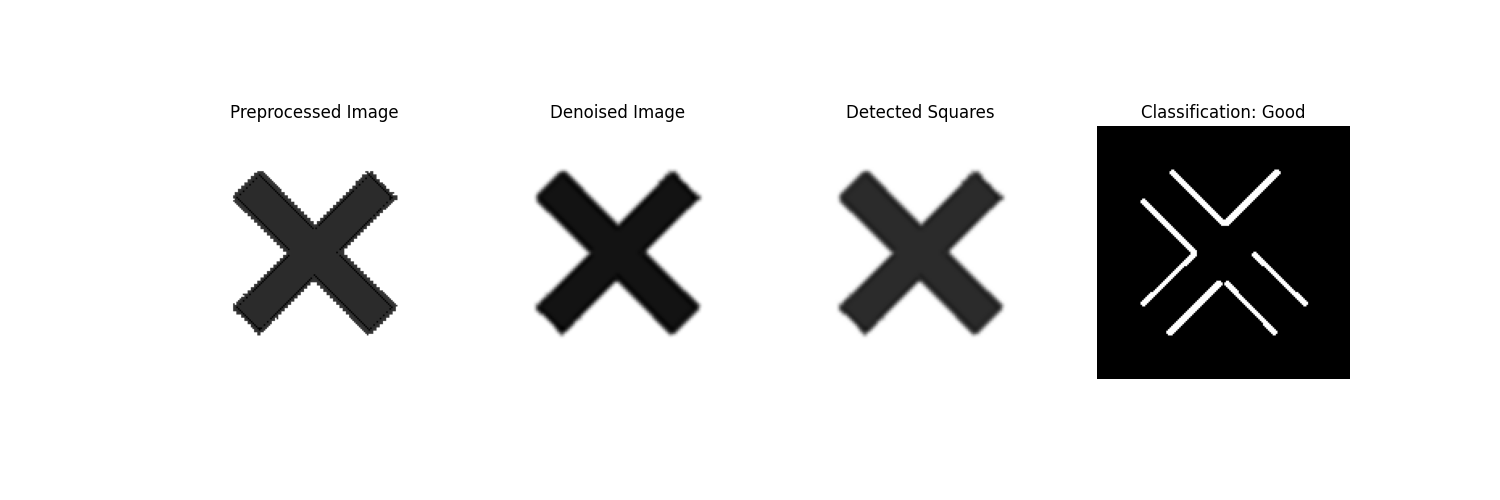
**Frequency analysis**: A **Fourier transform** can also reveal certain frequencies in the pixel intensity variations, which can lead to indications of stripe patterns - which can be very useful in our methodology,

1. **Pattern Classification**

After the extraction of line features, we will utilize a simple classifier like SVM or K-means clustering to distinguish the patterns between striped, cross-pattern, and no-pattern components based on the number of detected lines as well as the orientation of the lines.

**Pattern Thresholding / Classification:** After feature extraction and instead of using thresholding - we utilize SVM to distinguish between striped, cross-pattern and no-pattern components. The classifier utilizes detected characteristics of the lines, including number and orientation to classify patterns.

IE stripe patterns which are defined by multiple parallel lines or specific consistent line frequency, cross pattern detection by identifying intersecting lines, or no pattern.

****

**Handling poor pattern visibility:** In this case, where the lighting of the image affects pattern visibility, we will apply more adaptive thresholding. This is so components with ambiguous patterns due to poor lighting will be further processed by applying brightness and contrast adjustments.

1. Evaluation
2. We will use metrics such as **accuracy** to quantify the proportion of components classified as either “good” or “bad”. **Precision** will also be used to measure the true positives against the false positives. We will also consider scores such as F1 and dues on accuracy & precision.
3. Intersection over Union will also be used in evaluation.
4. Due to a rather small set of 20 images, **cross-validation** techniques will also be used to generalize pattern detection under varying lighting conditions. **Error analysis** will also be implemented to examine misclassified components to improve the preprocessing and feature extraction stages to ensure our methods are robust.

5. Timetable

*\*each major coding activity deadline also includes an evaluation step*

| **Activity/Task** | **Deadline** | **Group Member(s) Responsible** |
| --- | --- | --- |
| Finish /submit project plan | 11/6 | Jack, Troy, Mustafa |
| Preprocessing | 11/10 | Mustafa, Jack |
| Feature extraction | 11/13 | Troy, Mustafa |
| Pattern classification | 11/13 | Troy |
| Final evaluation metrics and error analysis | 11/20 | Jack |
| Finish and revise the final report | 11/20 | Mustafa, Jack |
| Prepare presentation | 12/1 | Mustafa, Jack |

**References:**

1. Chen, H., Zhang, Y., & Zhang, Y. (2021). "Automated Visual Inspection in Manufacturing Using Machine Learning: A Review." *IEEE Transactions on Automation Science and Engineering*, 18(4), 1688-1708.
2. Li, X., Chen, Y., & Li, Q. (2020). "Deep Learning-Based Defect Detection for Industrial Applications: A Survey." *IEEE Access*, 8, 157951-157970.