CSCE 5222 Feature Engineering

Project Plan/Report

Group 01

Member: Jack Duffield, Mustafa Syed, Troy Krupinski

Link to Google Colab: [CSCE 5222 Project.ipynb](https://colab.research.google.com/drive/1i1beQesZNdeoWlbyk4uuI6I7F83CQPKE?usp=sharing)

Link to GitHub: [GitHub](https://github.com/TroyKrupinski/CSCE5222_Project)

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. 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.

**“Good” component “Bad” component**



1. Method

Give details of the method and any parameters used.

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**

**Grayscale Conversion:** Convert images to grayscale, if not already grayscale.

**Histogram Equalization:** Application of **histogram equalization** to standardize contrast/lighting across images, which will help mitigate the overexposed and underexposed images in the dataset.

**Noise Reduction:** Application of **image smoothing** to reduce noise which can interfere with pattern detection. To remedy this, we will apply either **Gaussian or median filtering**, which can help us enhance the true patterns of the images in the dataset by removing variance in intensity.

1. **Feature Extraction**

**Contour Detection:** Utilize contour detection methods to identify all contours within the image.

**Shape Approximation**: Approximate each contour to a polygon using techniques like the Ramer–Douglas–Peucker algorithm (cv2.approxPolyDP in OpenCV).

**Quadrilateral Filtering**: Filter the approximated polygons to retain those with four sides (quadrilaterals).

**Aspect Ratio and Convexity Check**: Further filter quadrilaterals by checking if they are convex and have an aspect ratio close to 1:1, identifying them as squares.

**Isolation of Square Regions:** Once squares are detected, isolate these regions for focused analysis.

**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.

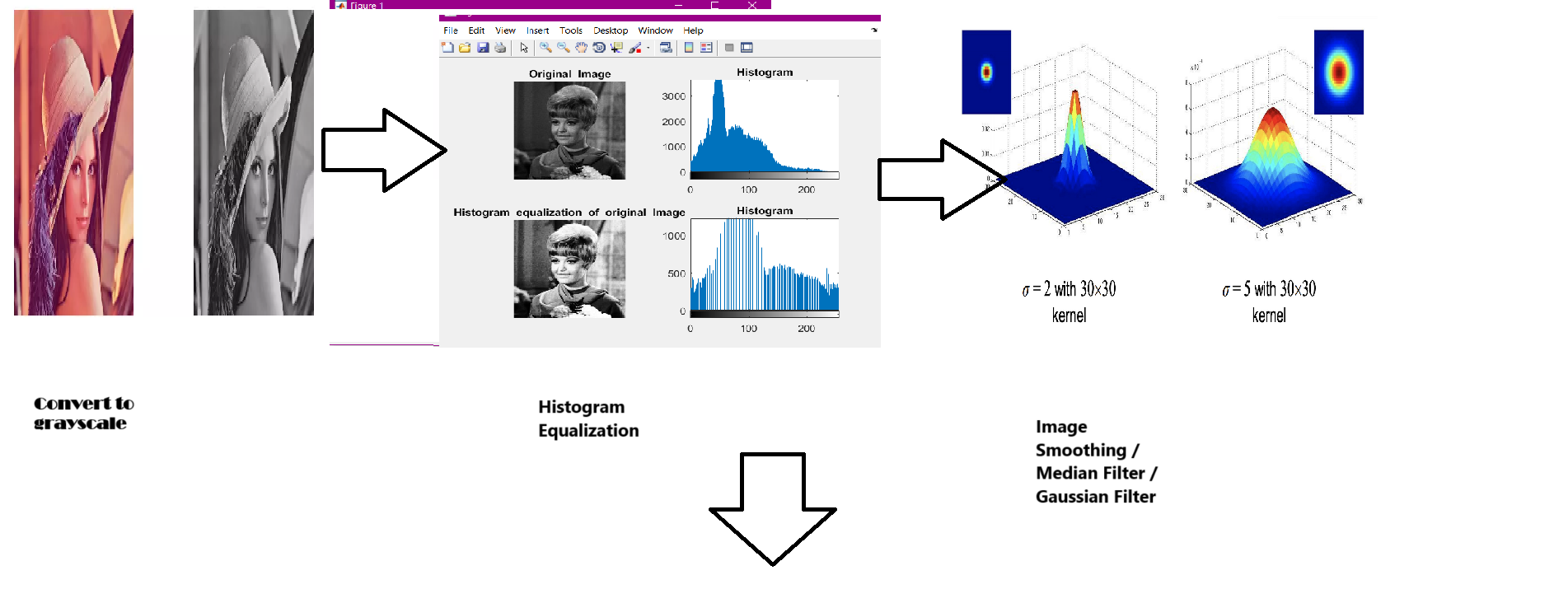
**Frequency analysis**: A **Fourier transform** can also reveal certain frequencies in the pixel intensity variations, which can lead to indications of stripe patterns - which can be very useful in our methodology,

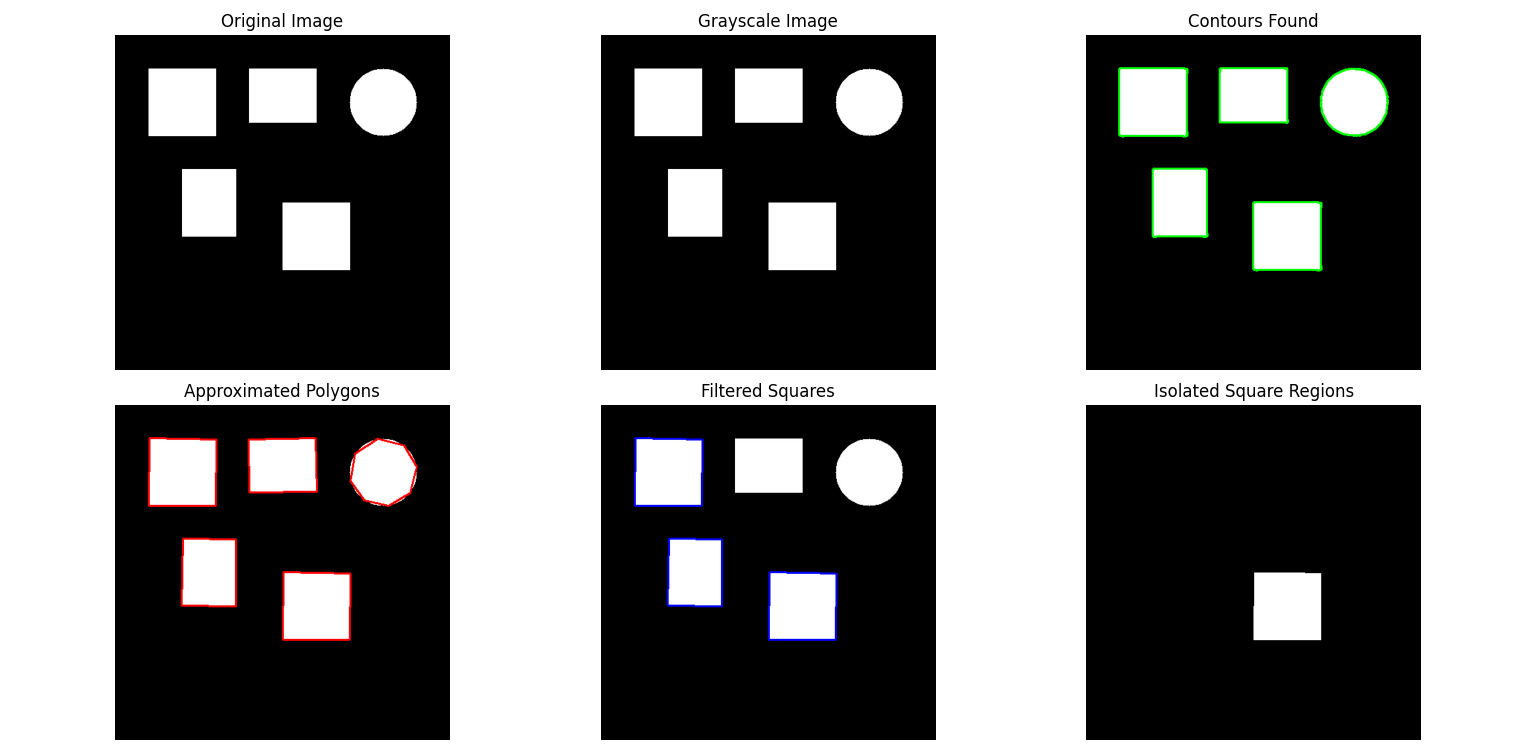
1. **Pattern Classification**

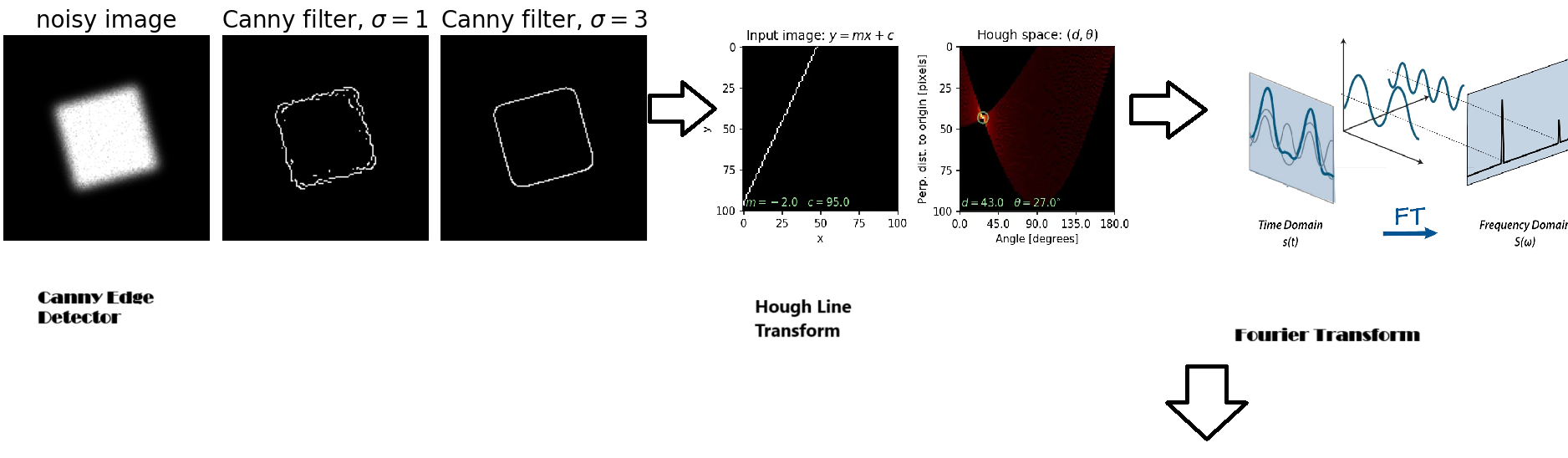
After the extraction of line features, we will utilize a simple classifier like SVM or K-means clustering to distinguish the patterns between striped, cross-pattern, and no-pattern components based on the number of detected lines as well as the orientation of the lines.

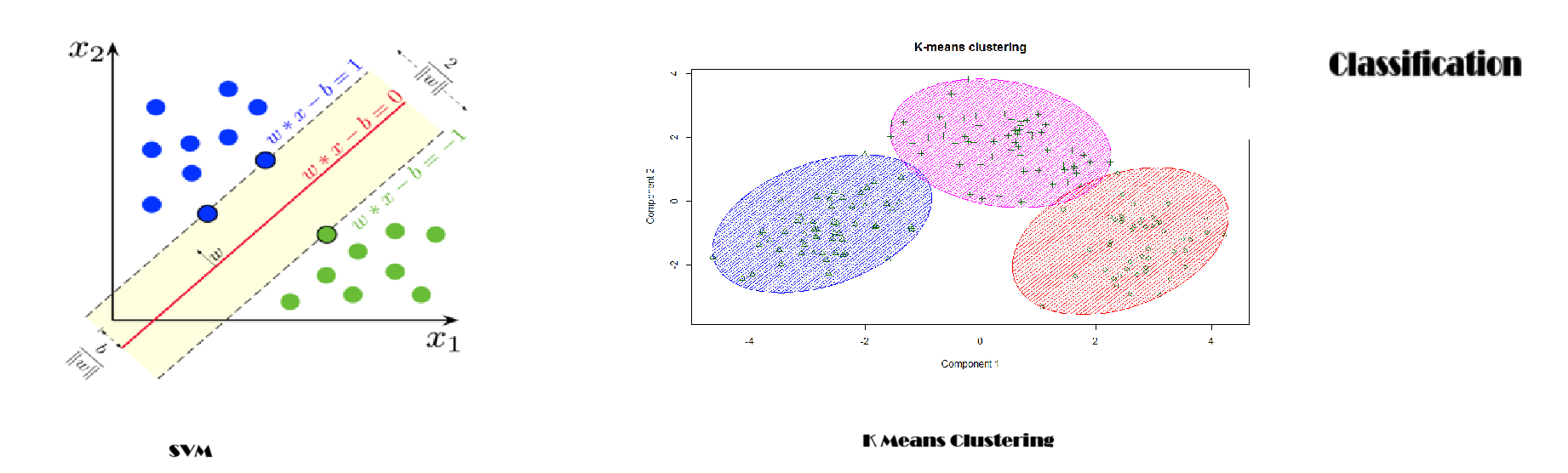
**Pattern Thresholding / Classification:** We will define specific thresholds to classify certain pattern types. For stripe patterns, it will be detected when multiple parallel lines or consistent line frequency is observed. Cross patterns will be detected when intersecting lines form a cross or some type of “X” shape. Components without any type of line detection above our predefined threshold will be classified as lacking an acceptable pattern and will be marked as “bad”.

**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 amphibious patterns due to poor lighting will be further processed by applying brightness and contrast adjustments.









**Workflow Summary**

**Preprocess the Image:**

Convert to grayscale.

Apply histogram equalization.

Reduce noise with Gaussian blur.

**Detect Squares:**

Find contours.

Approximate contours to polygons.

Filter for quadrilaterals (squares) based on aspect ratio and convexity.

Isolate square regions.

**Detect Patterns within Squares:**

Apply edge detection within squares.

Detect lines using Hough Transform.

Analyze lines for orientation and count.

**Classify the Component**

**Evaluate Performance:**

Compare classification results with ground truth labels.

Calculate evaluation metrics

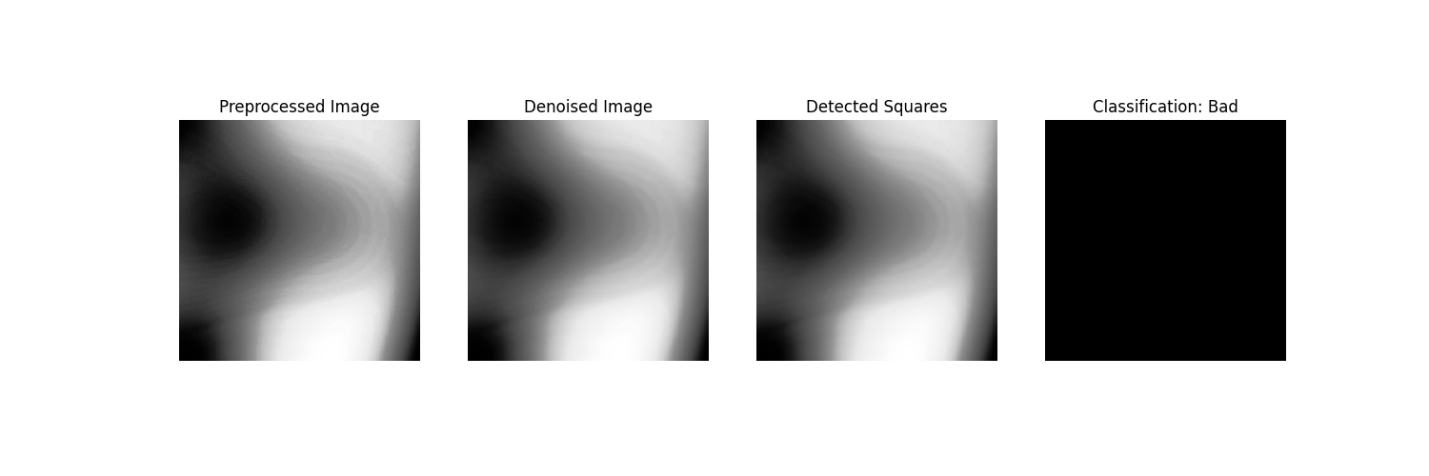
1. Evaluation
2. We will use metrics such as **accuracy** to quantify the proportion of components classified as either “good” or “bad”. **Precision** will also be used to measure the true positives against the false positives. We will also consider scores such as F1 and dues on accuracy & precision.
3. 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.

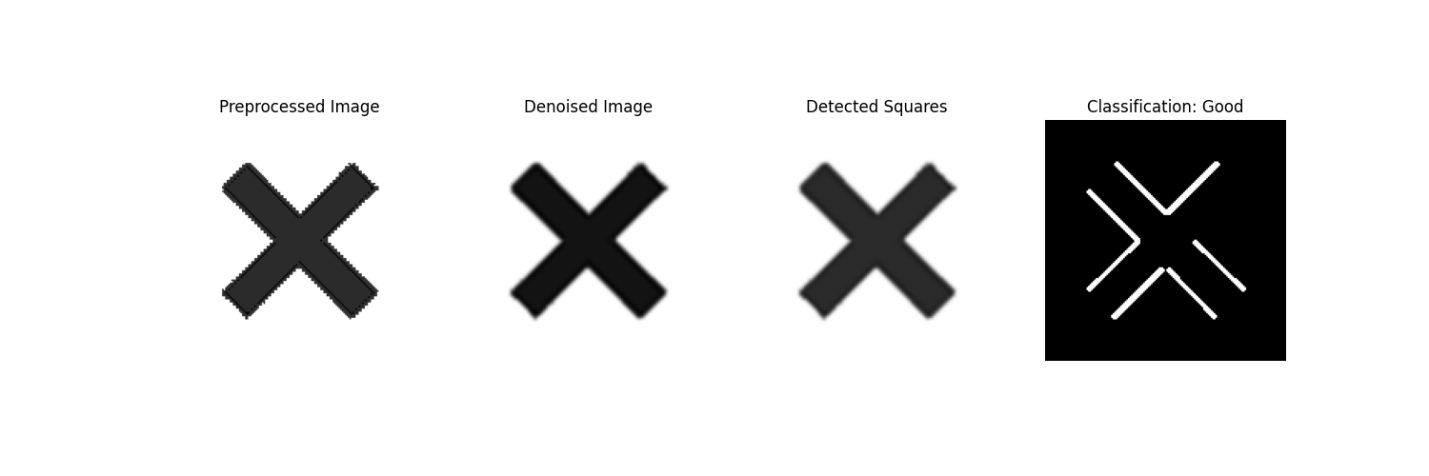
5. Timetable

|  |  |  |
| --- | --- | --- |
| **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 |
| Pattern classification | 11/13 | Troy |
| 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.**





The following are examples of inserting tables and figures. You can duplicate them and make changes for yours.

**Example output:**

*Unique labels in ground truth: {0, 1}*

*Unique classes in training labels: [0 1]*

*Processing Image: C:\Users\dunke\Desktop\CSCE5222\_Project\fabrizio-coco-u8w5OdRFPn8-unsplash.jpg*

*Image - Ground Truth: 0, Predicted: 1*

*Processing Image: C:\Users\dunke\Desktop\CSCE5222\_Project\good.png*

*Image - Ground Truth: 1, Predicted: 1*

*Unique classes in training labels: [0 1]*

*Processing Image: C:\Users\dunke\Desktop\CSCE5222\_Project\sample\_shapes.jpg*

*Image - Ground Truth: 1, Predicted: 1*

*Processing Image: C:\Users\dunke\Desktop\CSCE5222\_Project\bad.png*

*Image - Ground Truth: 0, Predicted: 0*

*Cross-Validation Classification Metrics:*

*Accuracy: 0.75*

*Precision: 0.67*

*Recall: 1.00*

*F1 Score: 0.80*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Size | Type | Class | Accuracy | Precision |
| Image 1 | 798x482 | .png | Good | 1 | 1 |
| Image 2 | 241x249 | .png | Bad | 0 | 0 |
|  |  |  |  |  |  |