Final Project Report

Lighting-Invariant Feature-Based Disease Detection in Soybean Leaves

Team Members: Rohit Anand & Yensong Li

Course: Computer Vision (CSE-5524)

Course Instructor- Prof. Joe Barker

---

1. Overview and Goals

This project aimed to develop a lightweight and interpretable computer vision pipeline for detecting and classifying four different soybean leaf classes. There was one Healthy and three diseased classes, i.e. Soyabean rust, Bacterial blight, and Frogeye, and features for each class were extracted using classical image processing techniques. We avoided deep learning methods to maintain interpretability and focus on course-relevant concepts. A key challenge we addressed was the variation in lighting conditions, which significantly affects color-based image features. Our approach combined robust preprocessing with creating biologically inspired color and shape features; we then did Lighting variation augmentation using scikit-image and then compared lighting alteration test results using Mahalanobis distance-based classifier and Random Forest classifier.

2. Datasets Collection (*Rohit*)

We used the image dataset collected from Datadryad.org website [1], consisting of four classes: Healthy, Rust, Bacterial Blight, and Frogeye leaf diseases. We took 100 RGB images of each of the 4 classes and did preprocessing and features extraction into a dataframe. Images were selected such that background was either easy to subtract using basic thresholding techniques or was minimally present altogether with most of each image consisting of soybean leaves only. Images included diverse lighting conditions that made preprocessing and lighting normalization important for achieving reliable feature extraction and classification.

Healthy Frogeye Bacterial Blight Soyabean Rust

3. Methodology (*Rohit*)

3.1 Preprocessing

To ensure balanced representation across all classes, we performed a stratified split of the dataset, allocating 70 images for training and 30 for testing per disease category namely soybean rust, bacterial blight, frogeye, and healthy. This approach allowed us to optimize and fine-tune our feature extraction pipeline using only the training set, ensuring that the features were both robust and generalizable, while minimizing the risk of overfitting on unseen test data.

To prepare our dataset for robust feature extraction, we implemented a structured image preprocessing pipeline. Starting with class-wise folders containing raw images, each image was first resized to a fixed size of 256×256 pixels to ensure uniformity across the dataset. We then converted the images from RGB to HSV color space, which separates chromatic content from intensity—making it more suitable for dealing with lighting variations. To enhance local contrast, especially under non-uniform illumination, we applied CLAHE (Contrast Limited Adaptive Histogram Equalization) specifically on the Value (V) channel. This helped make lesion textures more distinguishable. We also applied a Gaussian blur to smooth out minor noise and artifacts, improving the quality of region segmentation.

Next, we used predefined HSV threshold values to create a binary mask that isolates the green leaf area effectively filtering out background and non-leaf regions. This mask was then applied to the original image using a bitwise operation to retain only the relevant portion of the leaf. Finally, the processed images were saved into a folder structure under a new output directory, preserving the original class labels for ease of training and evaluation.

3.2 Feature Extraction (Color and Shape features)

To enable robust classification of soybean leaf diseases, we extracted both color and lesion-based shape features from each preprocessed image. For color features, we converted images to the HSV color space and calculated the mean and standard deviation of the Hue and Saturation channels. These descriptors capture variations in leaf pigmentation, which can differ across disease types and lighting conditions. Simultaneously, we extracted shape features by thresholding grayscale images to segment lesion regions and applying contour analysis. For each detected lesion, we computed key morphological attributes such as area, eccentricity (indicating elongation), and circularity (measuring roundness). We also recorded the lesion count, average lesion area, and the percentage of the leaf area affected. These shape features are crucial for identifying the spatial footprint and geometry of disease symptoms. All extracted attributes—both color and shape—were stored in structured CSV files along with the corresponding image filenames and class labels. This combined feature set formed the foundation for downstream dimensionality reduction and classification, helping us distinguish disease classes under varied lighting conditions.

Healthy Frogeye Bacterial blight Soyabean rust.

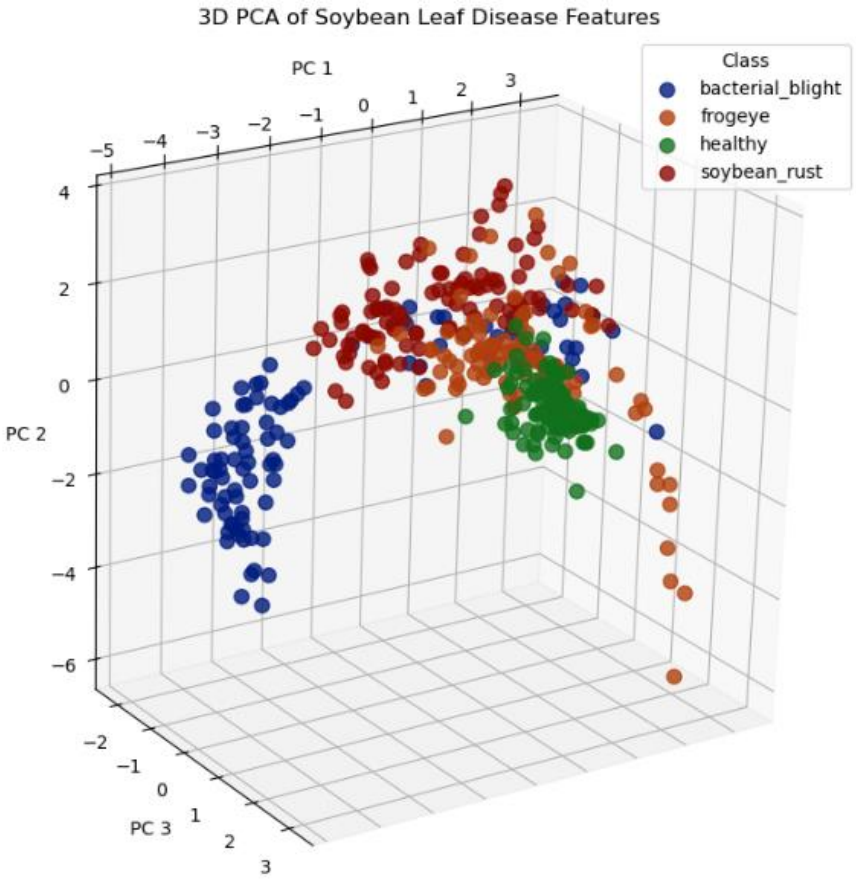
```python
import os
import cv2
import numpy as np
import pandas as pd

# Path to preprocessed images
input_dir = r'C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\Disease Project\Preprocessed_goodbackgroundProject_dataset'
features = []

for label in os.listdir(input_dir):
    class_dir = os.path.join(input_dir, label)
    for file in os.listdir(class_dir):
        path = os.path.join(class_dir, file)
        img = cv2.imread(path)
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        h, s, _ = cv2.split(hsv)

        mean_h = np.mean(h)
        std_h = np.std(h)
        mean_s = np.mean(s)
        std_s = np.std(s)

        features.append([file, mean_h, std_h, mean_s, std_s, label])

df = pd.DataFrame(features, columns=["filename", "mean_H", "std_H", "mean_S", "std_S", "class"])
df.to_csv(r"C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\color_features2.csv", index=False)
```

```python
import os
import cv2
import numpy as np
import pandas as pd

# Paths
input_dir = r'C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\Disease Project\Preprocessed_goodbackgroundProject_dataset'
output_csv = r'C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\lesion_shape_features2.csv'

# Output structure
features = []

def compute_circularity(area, perimeter):
    if perimeter == 0:
        return 0
    return (4 * np.pi * area) / (perimeter ** 2)

def compute_eccentricity(contour):
    if len(contour) < 5:
        return 0
    ellipse = cv2.fitEllipse(contour)
    major_axis = max(ellipse[1])
    minor_axis = min(ellipse[1])
    if major_axis == 0:
        return 0
    return np.sqrt(1 - (minor_axis / major_axis) ** 2)

# Process each image
for label in os.listdir(input_dir):
    class_path = os.path.join(input_dir, label)
    if not os.path.isdir(class_path):
        continue

    for file in os.listdir(class_path):
        path = os.path.join(class_path, file)
        img = cv2.imread(path)
```

Code Structure for extracting color and shape features

3.3 PCA

Principal Component Analysis (PCA) was used for dimensionality reduction and visualization. To get a better sense of how well our extracted features could separate different disease classes, we used 3D PCA. We first standardized the features, so everything was on the same scale, then reduced them to just three main components that capture the most important patterns in the data. Each point in the 3D plot represents one leaf image, and we colored them based on their disease type to easily see the grouping. We also tweaked the viewing angle to make the separation between classes more visible. This gave us a clear visual check that our color and shape features are actually helping distinguish between the different soybean diseases.



PCA plot of Combined features of Non-altered Light image datasets

3.4 Lighting variation augmentation

To make our model more robust to real-world conditions, we simulated lighting variations on all the images in our dataset. This script goes through every image file in the dataset and slightly alters its lighting using natural lighting profiles like warm sunlight near the horizon or softer noon daylight tones. Each image is first normalized (scaled between 0 and 1), and then a random lighting intensity is applied based on preset parameters that simulate realistic environmental conditions. Depending on the selected intensity, the script modifies the image using either a sunset-like

(orange-toned) or noon-like (white-toned) lighting vector. This simulates how the same leaf might appear under different times of day or weather conditions. The processed images are then clipped back to valid pixel values and saved in PNG format to avoid data loss. Importantly, the script preserves the original folder structure when saving the altered images, making it easy to keep track of which image belongs to which class. This augmentation helped us test how well our feature extraction pipeline holds up when lighting is inconsistent, just like it would be in a real farm setting.

```python
import os
import random
import numpy as np
from skimage import io
from skimage.color import rgba2rgb
from skimage.util import img_as_ubyte

# PATHS
input_root = r'C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\Disease Project\imgs_with_good_background'
output_root = r'C:\Users\anand.252\OneDrive - The Ohio State University\OSU\Summer 2025\Computer Vision\Group Project\Disease Project\rohit_lighting_altered'

# Lighting simulation parameters
gen_seed = 34352
lower_frac = 0.5
upper_frac = 1.5
switch_frac_threshold = 1.0
min_ambient_frac = 0.03

sun_horz_lighting_color_vec = np.array([1.0, 0.81, 0.65]).reshape(1, 1, 3)
noon_lighting_color_vec = np.array([1.0, 0.96, 0.93]).reshape(1, 1, 3)

random.seed(gen_seed)

# Traverse and process all images
for root, _, files in os.walk(input_root):
    for fname in files:
        if fname.lower().endswith((".jpg", ".jpeg", ".png")):
            input_path = os.path.join(root, fname)

            try:
                img = io.imread(input_path)

                if img.ndim == 2:  # Grayscale
                    print(f"Skipping grayscale image: {input_path}")
                    continue
                elif img.shape[-1] == 4:  # RGBA to RGB
                    img = rgba2rgb(img)

                # Normalize
                img_raw_normed = img.astype(np.float64) / 255.0

                # Lighting modification
                selected_intensity_frac = random.uniform(lower_frac, upper_frac)
                if selected_intensity_frac >= switch_frac_threshold:
                    res_img_raw = (
                        selected_intensity_frac * noon_lighting_color_vec * img_raw_normed
                        + min_ambient_frac * noon_lighting_color_vec
                    )
                else:
                    res_img_raw = (
                        selected_intensity_frac * sun_horz_lighting_color_vec * img_raw_normed
                    )

                # Final image conversion
                fin_procd_img = np.round(np.clip(res_img_raw, 0.0, 1.0) * 255.0).astype(np.uint8)

                # Compute relative path and save
                rel_path = os.path.relpath(input_path, input_root)
                rel_path_png = os.path.splitext(rel_path)[0] + ".png"
                output_path = os.path.join(output_root, rel_path_png)

                os.makedirs(os.path.dirname(output_path), exist_ok=True)
                io.imsave(output_path, fin_procd_img)

            except Exception as e:
                print(f"❌ Error processing {input_path}: {e}")

print("✅ All images processed and saved with lighting variations.")
```

Lighting alteration code structure

4. Methodology (*Yensong*)

4.1 Train-Test Split and Feature Extraction

We first split our data in a stratified manner based on each disease class (soybean rust, bacterial blight, frogeye, and healthy) into 70 training images and 30 test images for each class. This way, we can tune our feature extraction pipeline on only the training images to maintain maximum quality features extracted while avoiding any overfitting.

For feature extraction, to handle lighting inconsistency we first converted each image to HSV color space, isolating chromaticity (Hue and Saturation) from brightness (Value), and then subsequently throwing out all Value channel information, thus leaving us with just the Hue and Saturation channel as numpy matrices for each image. Then, we define background pixels as a numpy boolean matrix mask using Saturation channel thresholds depending on the filename being processed, and we define healthy pixels also as a boolean matrix mask using a fixed Hue channel range. Both the Saturation channel thresholds and Hue channel range were tuned only on images set aside for training purposes to prevent any overfitting. Next, sick regions are isolated using the healthy and background matrix masks and small noise artifacts are removed from the sick regions using connected component filtering. Finally, the following five features are computed for each image:

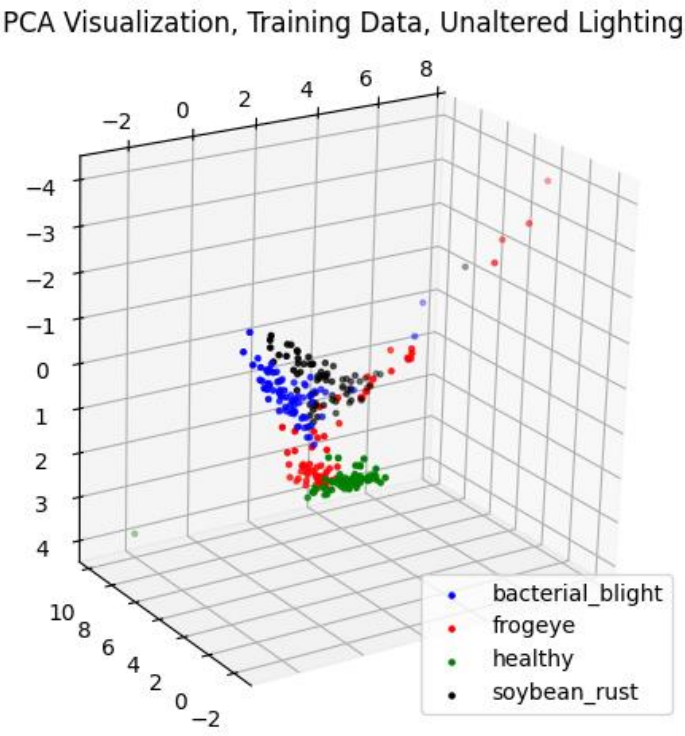
1. Standard deviation of Hue of non-background areas
2. Number of lesions in connected component filtered sick regions
3. Mean eccentricity of lesions in filtered sick regions
4. Mean lesion size in filtered sick regions normalized based on original image dimensions
5. Percent of non-background regions which correspond to sick regions

The above five features are then extracted for all images and saved to a CSV file as a dataframe for training the two types of classifiers we had set out to explore the performance of under various lighting conditions, namely the Mahalanobis distance-based and Random Forest classifiers. Please refer to outputs of cells 1-6 of "cv_proj_ted_preprocessing.html" for intermediate image and image-like results obtained during feature extraction, as well as cell 7 of that same HTML file for the actual code used to extract all the features as described above.

4.2 PCA

Since PCA was a major component of our project, we implemented PCA partially from scratch using only basic numpy functions based on code we had written for homework 3 for this computer vision course. The goal of conducting PCA as stated previously in Section 3.4 of this report was to get a rough idea of how separable the extracted features are based on the features extracted from the images. The output plot generated from the PCA implemented partially from scratch was compared to the output plot generated by built-in PCA functionality from scikit learn's decomposition module to ensure that PCA was implemented correctly. Please see on the next page

for the PCA plot of Yensong's features whose extraction was detailed in the previous section of this report, as well as cell 3 of "cv_proj_ted_portion_ted_preprocess.html" for code used to implement PCA partially from scratch. Only training data was visualized to ensure that we did not make decisions based on the visualization that would cause our subsequent Mahalanobis and Random Forest classifier models to overfit.



4.3 Classifier Training

A custom Mahalanobis distanced based classifier was implemented mostly from scratch using basic numpy functions and "trained" on the entire training data set via computing class-wise mean vectors and covariance matrices for each class. No hyperparameter tuning was necessary as the Mahalanobis classifier implemented relies solely on class means and covariance matrices derived directly from the training data, making its decision boundaries fully determined by the data distribution rather than adjustable parameters. Full implementation of the relatively simple "training" algorithm as well as test results generation can be found in cell 5 of "cv_proj_ted_portion_ted_preprocess.html"

As for the Random Forest Classifier which was used as a performance baseline to be compared against the Mahalanobis Classifier in unaltered lighting and test-time altered lighting conditions, it was the RandomForestClassifier directly imported from scikit learn's ensemble module. Hyperparameter tuning was conducted on the Random Forest Classifier via 8-fold cross validation hyperparameters grid search with classifier accuracy being the metric to maximize during tuning. Tuning was done across at least 9000 sets of hyperparameters including the default ones to ensure

that the model configuration chosen was both optimal and robust, avoiding bias toward any single hyperparameter setting while maximizing overall performance. Since our dataset is very balanced, with there being equal number of data points per class being classified, accuracy was the chosen metric because it is the most direct one to optimize because we wanted our classifiers to be able to classify correctly as much test data as possible and we weren't faced with any of the typical issues that may come with an imbalanced data set, such as high accuracy scores masking poor effective classifier precision and recall performance. Full implementation of the straightforward hyperparameter tuning process can be found in cell 7 of "cv_proj_ted_portion_ted_preprocess.html".

For all classifiers tested, training was only done on features extracted from lighting unaltered training images for testing each classifier with both features extracted from lighting altered and lighting unaltered images, to ensure that there would be no data leakage from the artificial lighting augmentation done on the test images.

4.4 Feature Extraction Retuning for Test-time Altered Lighting Testing

To ensure that Yensong's feature extraction pipeline was still valid for lighting altered images as produced by Rohit's lighting augmentation work, the feature extraction pipeline logical framework was kept the same compared to feature extraction on unaltered images but the background extraction Saturation threshold and Healthy Hue range was retuned on training images only that have gone through the same exact lighting alteration process as the test images. Retuning was done on training images only as always to prevent classifier overfitting and data leakage. Unfortunately, due to time constraints Rohit couldn't tune his respective preprocessing pipeline on the same set of training images, leading to poor classifier performance as demonstrated later under test-time lighting alterations. All code associated with this section of the report can be found in "cv_proj_ted_preprocessing_lighting_altd.html".

5. Results Descriptions and Results Analysis (***Shared between Yensong and Rohit***)

5.1 Results Descriptions Figures

Figures begin on the next page.

Lighting Unaltered Mahalanobis Test Results, Rohit's Features

	precision	recall	f1-score	support
bacterial_blight	0.93	0.83	0.88	30
frogeye	0.51	0.83	0.63	30
healthy	1.00	0.57	0.72	30
soybean_rust	0.81	0.73	0.77	30
accuracy			0.74	120
macro avg	0.81	0.74	0.75	120
weighted avg	0.81	0.74	0.75	120

Lighting Unaltered Random Forest Test Results, Rohit's Features

	precision	recall	f1-score	support
bacterial_blight	0.86	0.83	0.85	30
frogeye	0.68	0.70	0.69	30
healthy	0.86	0.80	0.83	30
soybean_rust	0.81	0.87	0.84	30
accuracy			0.80	120
macro avg	0.80	0.80	0.80	120
weighted avg	0.80	0.80	0.80	120

Lighting Altered Mahalanobis Test Results, Rohit's Features

	precision	recall	f1-score	support
bacterial_blight	0.94	0.53	0.68	30
frogeye	0.33	0.97	0.49	30
healthy	0.00	0.00	0.00	30
soybean_rust	0.50	0.23	0.32	30
accuracy			0.43	120
macro avg	0.44	0.43	0.37	120
weighted avg	0.44	0.43	0.37	120

Lighting Altered Random Forest Test Results, Rohit's Features

	precision	recall	f1-score	support
bacterial_blight	0.72	0.60	0.65	30
frogeye	0.42	0.90	0.57	30
healthy	0.68	0.63	0.66	30
soybean_rust	0.00	0.00	0.00	30
accuracy			0.53	120
macro avg	0.45	0.53	0.47	120
weighted avg	0.45	0.53	0.47	120

Figures continue on the next page.

Lighting Unaltered Mahalanobis Test Results, Yensong's Features

	precision	recall	f1-score	support
bacterial_blight	0.76	0.93	0.84	30
frogeye	0.82	0.90	0.86	30
healthy	1.00	0.83	0.91	30
soybean_rust	0.88	0.73	0.80	30
accuracy			0.85	120
macro avg	0.86	0.85	0.85	120
weighted avg	0.86	0.85	0.85	120

Lighting Unaltered Random Forest Test Results, Yensong's Features

	precision	recall	f1-score	support
bacterial_blight	0.96	0.87	0.91	30
frogeye	0.82	0.90	0.86	30
healthy	1.00	0.93	0.97	30
soybean_rust	0.88	0.93	0.90	30
accuracy			0.91	120
macro avg	0.91	0.91	0.91	120
weighted avg	0.91	0.91	0.91	120

Lighting Altered Mahalanobis Test Results, Yensong's Features

	precision	recall	f1-score	support
bacterial_blight	0.68	0.83	0.75	30
frogeye	0.58	0.83	0.68	30
healthy	1.00	0.47	0.64	30
soybean_rust	0.77	0.67	0.71	30
accuracy			0.70	120
macro avg	0.76	0.70	0.70	120
weighted avg	0.76	0.70	0.70	120

Lighting Altered Random Forest Test Results, Yensong's Features

	precision	recall	f1-score	support
bacterial_blight	1.00	0.50	0.67	30
frogeye	0.51	0.77	0.61	30
healthy	0.87	0.67	0.75	30
soybean_rust	0.68	0.83	0.75	30
accuracy			0.69	120
macro avg	0.76	0.69	0.70	120
weighted avg	0.76	0.69	0.70	120

5.2 Comparing Results via Statistical Testing Procedure

To more rigorously access the performance of the various classifiers using the various features extracted under the various conditions, standard McNemar's Testing was performed directly on the raw classifier output labels with ground truth labels as reference. The comparison axes that were done were:

1. Mahalanobis classifier outputs vs Random Forest Classifier outputs under the lighting unaltered conditions with Yensong's extracted features
2. Mahalanobis classifier outputs under artificially altered lighting conditions vs unaltered lighting conditions with Yensong's extracted features
3. Random Forest classifier outputs under artificially altered lighting conditions vs unaltered lighting conditions with Yensong's extracted features
4. Mahalanobis classifier outputs under the lighting unaltered conditions with Yensong's extracted features vs Rohit's extracted features
5. Random Forest classifier outputs under the lighting unaltered conditions with Yensong's extracted features vs Rohit's extracted features

For the sake of time other possible comparison axes were not done, so the above list is not exhaustive. In addition, due to lack of readily available Python libraries providing McNemar's version of Two One-Sided T-Tests and time constraints, we couldn't perform rigorous equivalence testing for demonstrating that either two different models performed similarly under the same test-time lighting conditions or that the same model performs similarly under different test-time lighting conditions. We could only demonstrate when two different models perform significantly differently under the same lighting conditions or that the same model performs significantly differently under different lighting conditions. The p value threshold used to determine statistically significant differences was less than or equal to 0.05.

5.3 Statistical Testing Results Overview and Discussion

Under unaltered lighting with Yensong's extracted features, none of the four classes showed significant difference between Mahalanobis and Random Forest classifiers i.e. all p values were greater than 0.05 here. With Yensong's extracted features, both classifiers degrade in performance, but Mahalanobis classifier only has significant degradation in the healthy and frogeye classes with p values of 0.00739 and 0.00258 respectively, whereas Random Forest had significant degradation in all four classes, with all p values for all classes being less than 0.02. We suspect that the reason behind Random Forest's poor showing here is that it had overfit on the training data due to it being "too good" at capturing subtle patterns and noise in the unaltered training set, which did not generalize well to the lighting-altered test images

Across both classifiers, Yensong's features outperformed Rohit's features with unaltered lighting images in at least the frogeye class (p value of 0.00018 for Mahalanobis classifier and p value of 0.04139 for Random Forest), and for the healthy class Yensong's features outperformed Rohit's features for Random Forest with p value of 0.02148). Random forest was more sensitive to better features because it leverages multiple decision trees that rely on feature splits, so having more informative and well-separated features directly improves its ability to create consistent and accurate decision boundaries. In addition, while we didn't conduct rigorous statistical tests to compare the performance of Yensong's against Rohit's under altered lighting conditions, the fact that all classifier's accuracy under altered lighting was greater than 65% for Yensong's features

and less than 55% for Rohit's features strongly suggests that Yensong's features as extracted are more lighting invariant. Since one of our objectives of this project was to find at least one classifier and one feature extraction method respectively out of the two different classifiers and two different feature extraction methods tested which was robust to lighting changes, we declare our project to have been partially successful at least with Yensong's extracted features with the custom Mahalanobis classifier implemented given everything discussed thus far. The complete failure of Random Forest to adapt to test time lighting changes was likely due to it overfitting on the training data by capturing subtle patterns too well as we alluded to earlier.

6. Improvements and Future Work (*Rohit*)

If given more time and resources, we would revisit our original proposal and implement a Riemannian manifold distance-based classifier as intended. This approach could potentially capture more nuanced geometric relationships among feature distributions, especially shape-based descriptors. Additionally, to strengthen our evaluation, we would incorporate McNemar's Two One Sided T-Tests alongside the standard tests for more statistically rigorous comparisons of classifier performance across different feature sets and methods. Another key area for refinement would be a deeper analysis of Rohit's vs. Yensong's extracted features not just qualitatively, but through controlled testing on classification performance. In terms of data, we recognize the limitation of using artificially simulated lighting variations. Ideally, we would augment the dataset with more diverse, real-world field images under varying natural lighting conditions to improve generalizability. Finally, while our current focus was on classical computer vision, we would also consider exploring hybrid models, such as combining our extracted features with lightweight machine learning classifiers like SVM or shallow CNNs, to assess whether they offer performance gains without sacrificing interpretability.

7. Division of Work During Project

Task	**Rohit**	**Yensong**
Datasets Collection	Collected	
Preprocessing and Feature extraction	Designed and implemented	Designed and implemented
Lighting variation augmentation	Code + structure	Validation
PCA & Visualization	Shared	Shared
Classifiers		Implemented & tested
Result evaluation	Shared	Shared

References

[1] https://datadryad.org/dataset/doi:10.5061/dryad.41ns1rnj3