Tomato Leaf Disease Classification

281 Computer Vision Final Project

Akaash Venkat
UC Berkeley
akaash.venkat@berkeley.edu

Nico Loffreda
UC Berkeley
nicolof88@berkeley.edu

Stephanie Cabanela
UC Berkeley
scabanela@berkeley.edu

Github Repo Link: https://github.com/scabanela-ischool/281-final-project

Introduction

According to a 2022 report by the United Nations, the world population is projected to increase from 8 billion today to 9.7 billion by 2050[1]. Factors like global population growth and climate change contribute to the increasingly urgent issue of food security. Currently, infectious diseases reduce the potential yield of crops by an average of 40% and go as high as 100% for many crop growers in the developing world[2]. The widespread distribution of smartphones among crop growers globally offers the opportunity to combat and prevent plant diseases by developing mobile disease diagnostic tools to algorithmically identify crop diseases through machine learning. The purpose of this project is to apply computer vision techniques to classify the leaves of tomato crops into different disease categories.

The Data

Our dataset is from the online platform PlantVillage and contains over 50,000 images of healthy and infected leaves of different planted crops across 38 categories. The images are in png format and each image has a size of 256 x 256 pixels. This dataset is publicly available through Kaggle[3]: PlantVillage Dataset. Since the number of categories in the starting dataset was too large, we narrowed our focus down to one plant that had a substantial amount of images across different disease categories, and ultimately went with a subset of images of healthy and infected leaves of tomato plants.

Classification Task

The task was to categorize each tomato leaf into one of ten categories:

1. Bacterial Spot
2. Early Blight
3. Healthy
4. Late Blight
5. Leaf Mold
6. Septoria Leaf Spot
7. Spider Mites Two-spotted Spider Mite
8. Target Spot
9. Tomato Mosaic Virus
10. Tomato Yellow Leaf Curl Virus

Dataset Description

The dataset provides images in three forms: grayscale, segmented, and color. Below in Figure 1, we can see an image of each category of healthy and infected tomato leaves, in the three forms provided by the dataset:

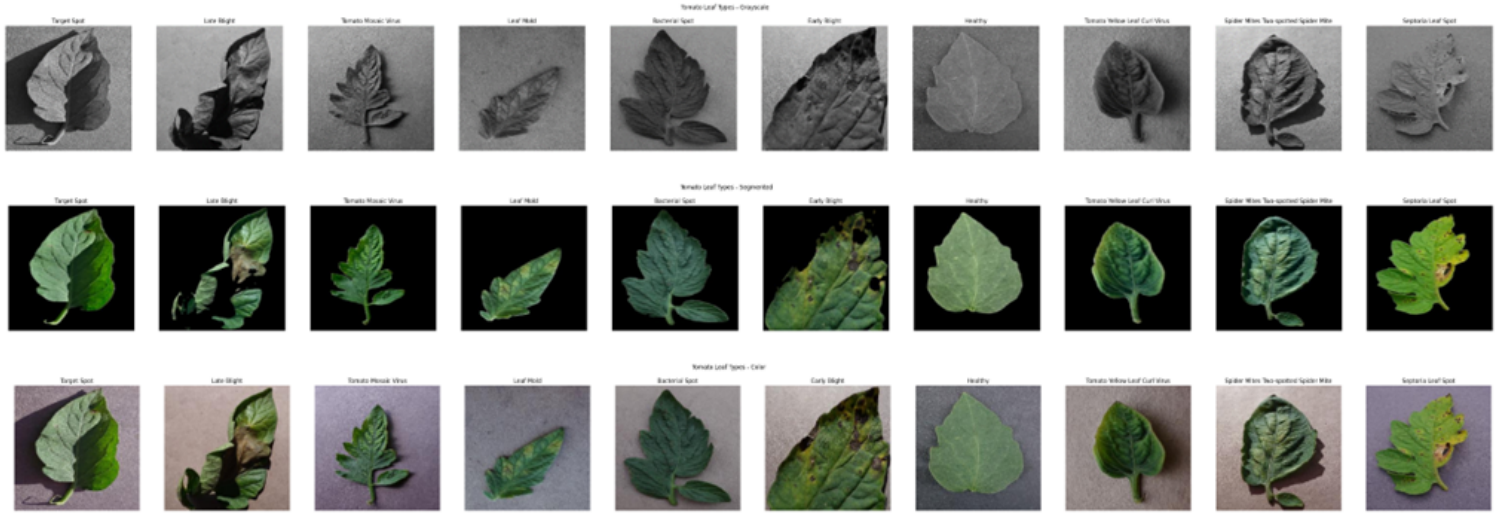

Figure 1: Example images of tomato plant leaves for each of the 10 disease categories. The top row contains sample images from the grayscale data subset, the middle row shows sample images from the segmented subset, and the bottom row contains images from the raw/color subset.

We used the segmented data subset which had some preprocessing already done: the background has already been removed to focus training on the leaf itself and upon visual inspection, there seems to have been some light balancing applied since the leaves appear to be brighter and seem to have more vibrant color tones.

Data Splits for Train, Validation and Testing

In the segmented folder for the healthy and infected tomato leaves, there were a total of 18,160 images spread across the ten different categories. Of those images, all were of shape (256, 256, 3), except for one image. To ensure that all images were of the same shape, we simply removed that one image that had a different shape, leaving us with 18,159 images.

We then performed stratified sampling and split these images into splits of 80% train, 10% validation, and 10% test. This resulted in a train set of 14,523 images, a validation set of 1,812 images, and a test set of 1,824 images as shown in Figures 2 and 3:

Tomato Leaf Category	***Train Dataset***	***Validation Dataset***	***Test Dataset***
Bacterial Spot	1701 images	212 images	214 images
Early Blight	800 images	100 images	100 images
Healthy	1272 images	159 images	160 images
Late Blight	1527 images	190 images	192 images
Leaf Mold	761 images	95 images	96 images
Septoria Leaf Spot	1416 images	177 images	178 images
Spider Mites Two-spotted Spider Mite	1340 images	167 images	168 images
Target Spot	1123 images	140 images	141 images
Tomato Mosaic Virus	298 images	37 images	38 images
Tomato Yellow Leaf Curl Virus	4285 images	535 images	537 images

Figure 2: Table of dataset distribution across 10 categories for train, validation, and test.

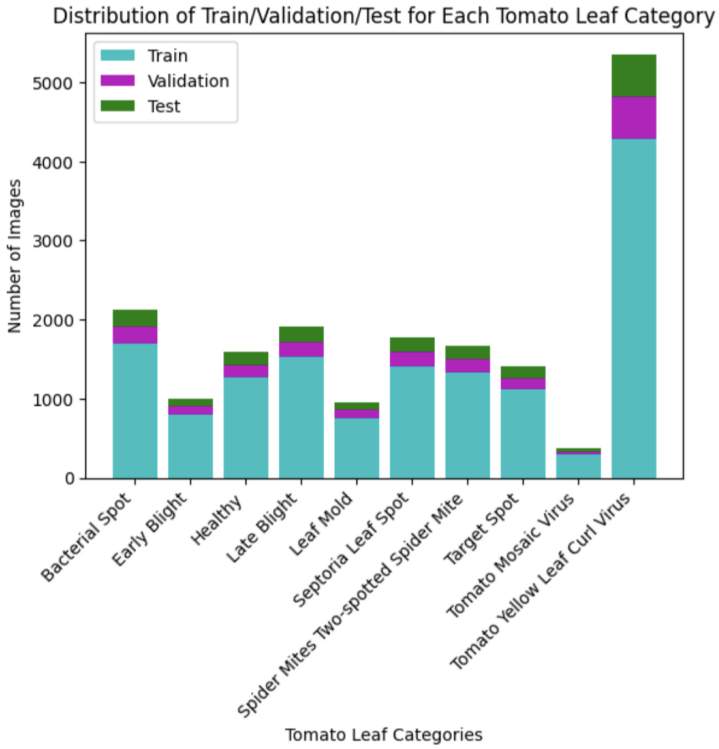

Figure 3: Stacked bar chart of dataset distribution.

Feature Extraction

Color histograms

One of the main effects that the different diseases have on the tomato plants is the decoloration of their leaves. We used color histograms to capture this effect.

We calculated histograms for both RGB and L*a*b color spaces. Histograms with 50 and 100 bins were calculated for each channel of those color spaces and normalized and tested on the classifiers. The ranges of the histogram were set to guarantee that all the images had the same bins. For RGB, the range was set from 0 to 1 given that the images were normalized. For L*a*b, the ranges were defined based on the possible values that each channel could take: *L* range goes from 0 to 100, *a* range goes from -86 to 98, and *b* range goes from -107 to 94. Lastly, each channel is concatenated into 1 vector for each image.

Below we can see the difference in the average L*a*b histogram for a few classes:

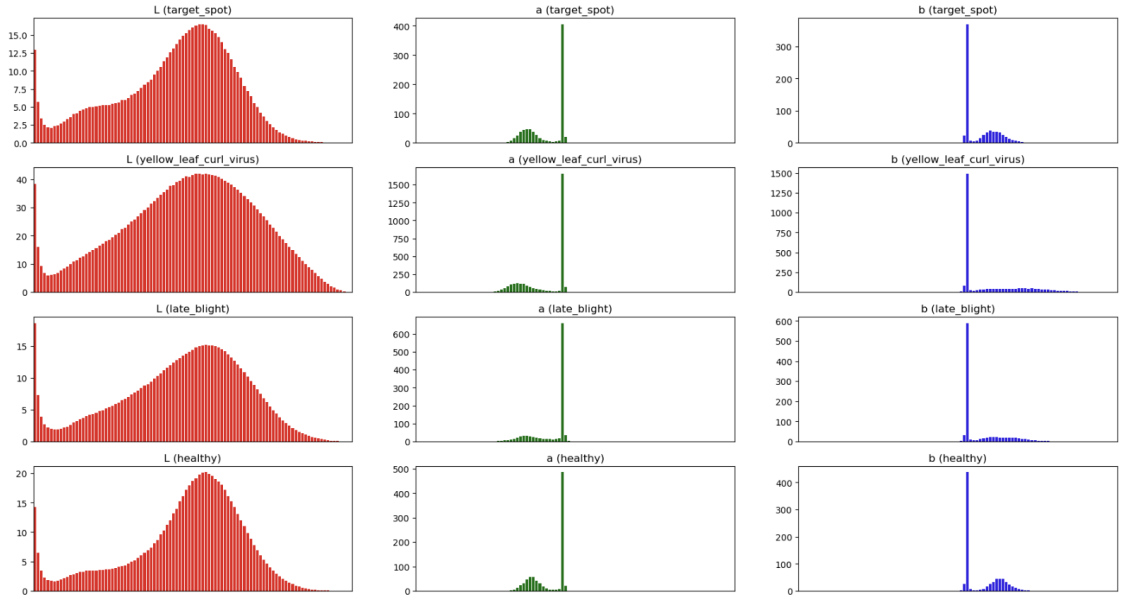

Figure 4: Histogram for the sum of each bin for all training images by channel for L*a*b color space for 4 diseases.

We can see that 0 is the predominant intensity for the color channels (*a* and *b*) given the black background of the leaves. Something similar happens in the RGB space.

When we apply PCA to the L*a*b feature vector, we see that 25 principal components (PCs) explain almost 100% of the variance:

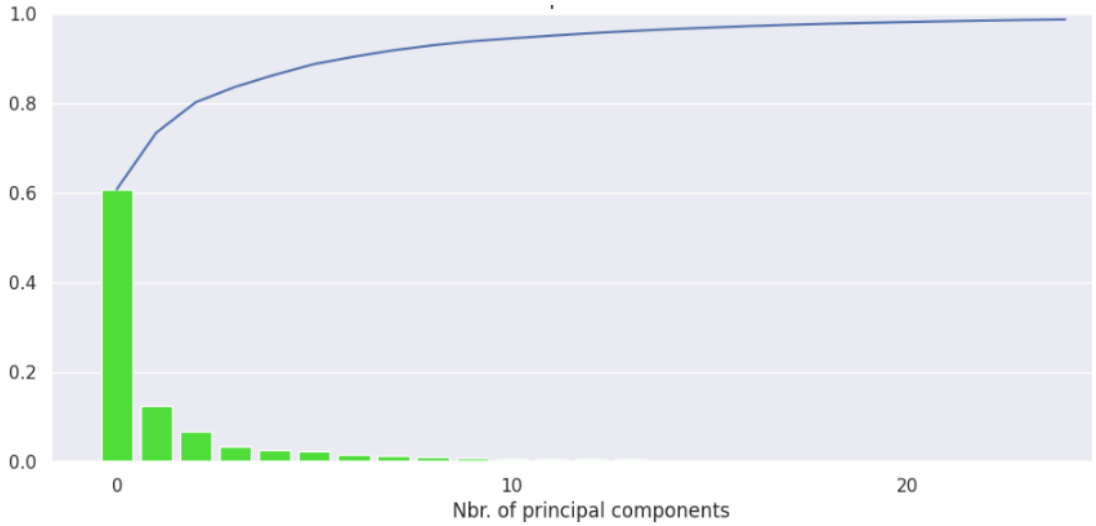

Figure 5: First 25 PCs of the L*a*b histogram features explain 99% of the variance.

Something very similar happens with the RGB space so it's not plotted here.

DAISY

The DAISY descriptor was proposed by Tola et. al[4] in order to efficiently compute a dense histogram of orientations that is robust to different scales and object rotations in the image. At each pixel location, DAISY consists of a vector made of values from the convolved orientation maps located on concentric circles centered on the location, and where the amount of Gaussian smoothing is proportional to the radii of the circles. Given these qualities, we hope that DAISY can capture features of the texture and deformations that each disease may have on the leaf.

The descriptor is parameterized by the radius of the outermost ring *R*, the number of ring layers *Q*, the number of histograms on a layer *T* and the number of bins (i.e. orientations) on each histogram *H*. For each of the *Q* layers, DAISY computes a normalized HOG for the pixels within its radius with *H* bins; but it first applies a Gaussian kernel proportional to the radius given by:

$$\sigma_i = \frac{R(i+1)}{2Q}$$

Where *i* represents the *i* th layer in the circular grid which has *Q* layers. For higher layers, sigma is higher and thus the image is more blurred by the kernel.

Although DAISY was designed to be computationally efficient, calculating it for over 15 thousands images is a time consuming process. A step size *S* was chosen to sample the descriptors every *S* number of pixels and avoid unfeasible running times. Given the time complexity to calculate DAISY, we only tried a few combinations of the parameters and saw best results using *R*=45, *Q*=2, *T*=6, *H*=8 and *S*=80. These returned a descriptor with 936 values representing each of the HOGs calculated for each ring layer, concatenated into one vector. We also tried a more dense representation with R=30, Q=3, T=6, H=8 and S=60, but this 2,437 feature descriptor didn't perform much better than the previously described one.

One other nice quality of DAISY is that it can easily be visualized by plotting the HOG for each layer of rings:

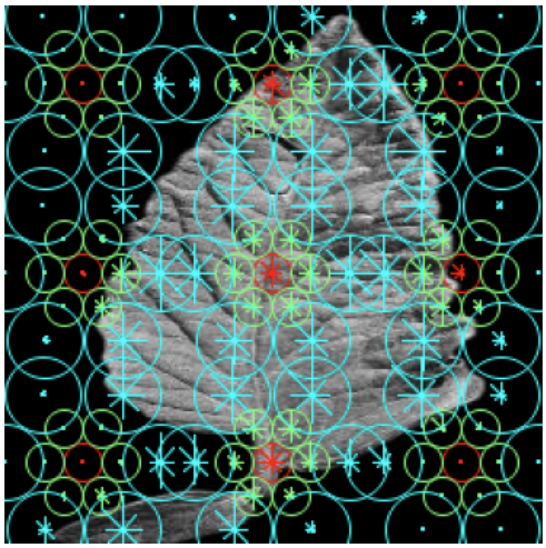

Figure 6: DAISY descriptors visualized with *R*=45, *Q*=2, *T*=6, *H*=8 and *S*=80.

In Figure 6 we can identify the 2 layers of circles (green and cyan circles) with 6 histograms each (6 circles on each layer) and the 8 orientations (lines indicating orientation strength inside each circle) for each descriptor. The radius of the circle is proportional to the sigma used for the Gaussian kernel applied to it. A step of 80 gives even coverage of the image; given the image size of 256x256, we end up with 3 rows with 3 descriptors each. The position at which each descriptor is centered can be identified at the center of the red circles.

The overlap between the ring layers is desired as it makes the descriptor more robust to different light conditions and object rotations according to the authors. One thing to note is that DAISY is calculated over a grayscale version of the images.

When applying PCA to DAISY on the training set, we see that approximately 80 PCs explain 90% of the variance:

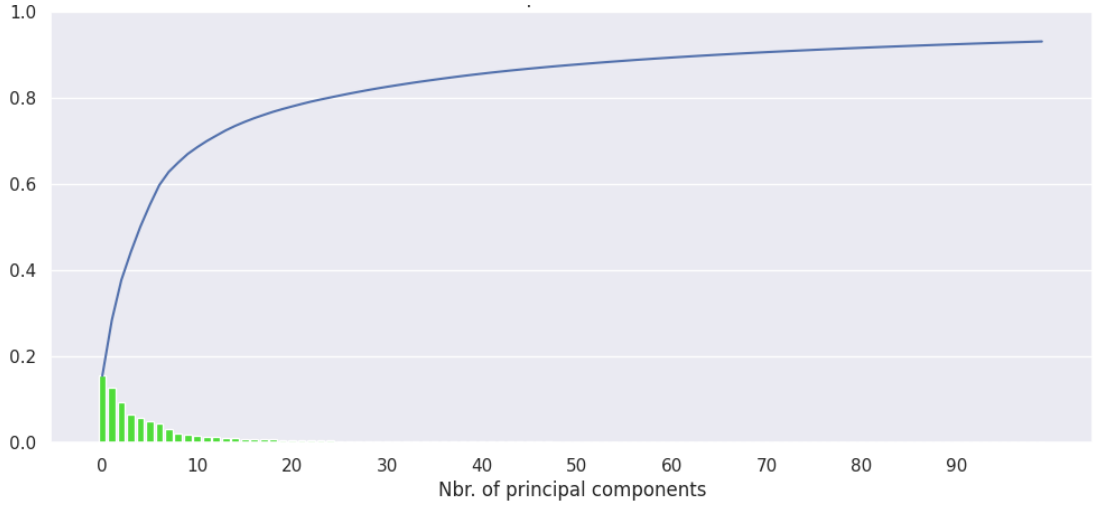

Figure 7: Daisy PCA, first 100 PCs explain 93% of the variance.

Canny edge detection

We also looked into Canny edge detection to see if we could extract more features from each image.

For each image, we first converted it to grayscale and applied a Gaussian kernel to blur the image. By blurring the image, we were able to eliminate some noise by removing some of the finer edges, specifically some of the thinner veins on the tomato leaves, from the image. To apply the Gaussian blur, we used a kernel of size [5, 5], and we let OpenCV automatically calculate the standard deviation of this kernel, to control the amount of blurring applied to the image.

Once we got the blurred image, we leveraged Canny to detect the edges in the image. We experimented with different Canny lower/upper threshold values, which play a role to determine what in the image would be considered as an edge. For different threshold values, we looked at the resulting edges for a few sample images from each of the 10 classes. After rounds of testing different threshold values, we settled on a lower threshold of 50 and upper threshold of 130.

Figure 8 shows an example of one of the tomato leaves from our dataset, and what the image looked like after applying the steps mentioned above.

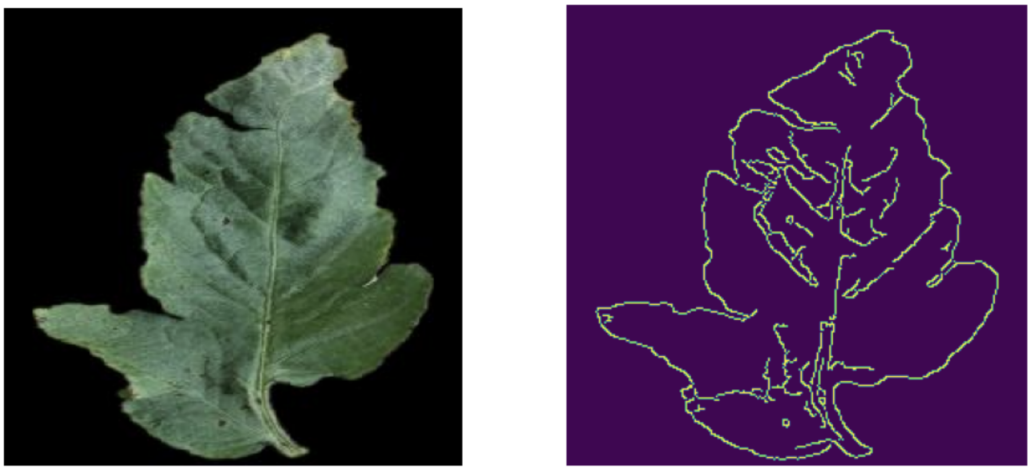

Figure 8: Application of Canny edge detection.

After applying Canny on each image, based on the parameters we defined, we were able to extract a 256 x 256 Numpy array of 0s and 1s. The 0s were attributed to pixels that were not deemed as an edge in the image, whereas the 1s were attributed to pixels that were deemed as an edge in the image. We then gathered the number of 0s and 1s in each image, and set aside those values as a feature vector of two elements. We then normalized this feature vector, such that the value of each element was a decimal value between 0 and 1.

By applying Canny and considering what portion of the image contained edges, we were able to produce a feature vector of two elements to apply for our classification.

Merging all the features

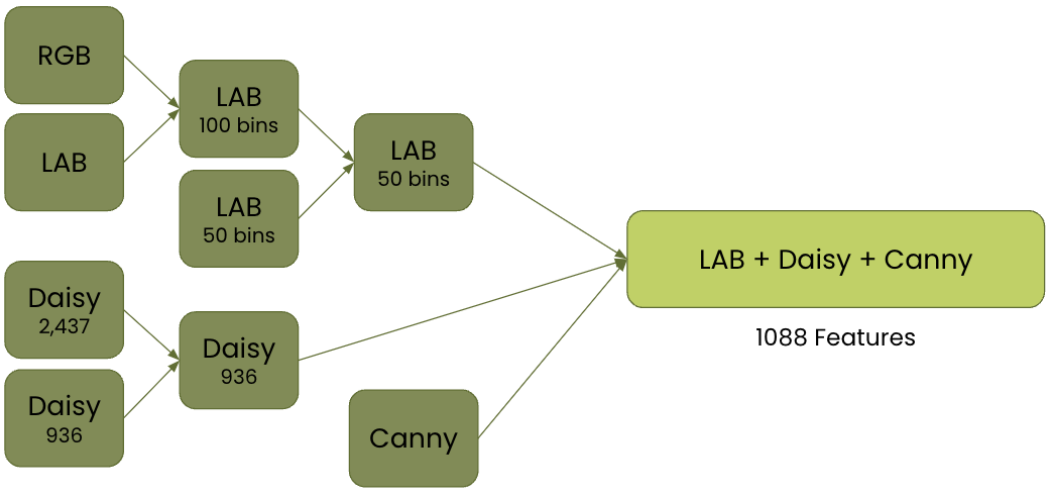

Figure 9: Diagram showcasing the various sets of features we considered before establishing our final feature vector.

After extracting feature vectors with RGB, L*a*b, Daisy, and Canny, we performed a side-by-side analysis by running the different variations of feature vectors through our classifiers. Based on the validation accuracy and time taken to fit the data, we determined whether to use a certain variation of features or not.

To be more specific, we saw that using the feature vector from the L*a*b histogram with 100 bins led to having a higher validation accuracy than using the feature vector from the RGB histogram with 100 bins, and it took a similar amount of time to fit the data for both feature vectors. Then we compared the feature vector from the L*a*b histogram with 100 bins with one from the L*a*b histogram with 50 bins, and saw similar validation accuracies between both feature vectors, but the feature vector associated with the 50 bin L*a*b histogram took considerably less time for fitting the data.

For the Daisy feature vectors, we saw that the feature vector of size 936 had similar / slightly worse validation accuracies than the feature vector of size 2437, but took a lot less time to fit the data. Thus, we determined the feature vector of size 936 to be more efficient for our classifications.

By doing this comparative analysis, we decided to use a final feature vector that contained data from the 50 bin L*a*b histogram, Daisy feature vector of size 936, and the Canny feature vector of size 2. Concatenating these three feature vectors resulted in having a final feature vector of size 1088.

We then proceeded to try applying dimensionality reduction to this final vector. First, we conducted a Principal Component Analysis for 100 principal components, and noticed that with 100 components, it explained for 98% of the variance. We also noticed that even with just 25 or 30 components, it would have explained for 90% of the variance, which we determined to be a good threshold.

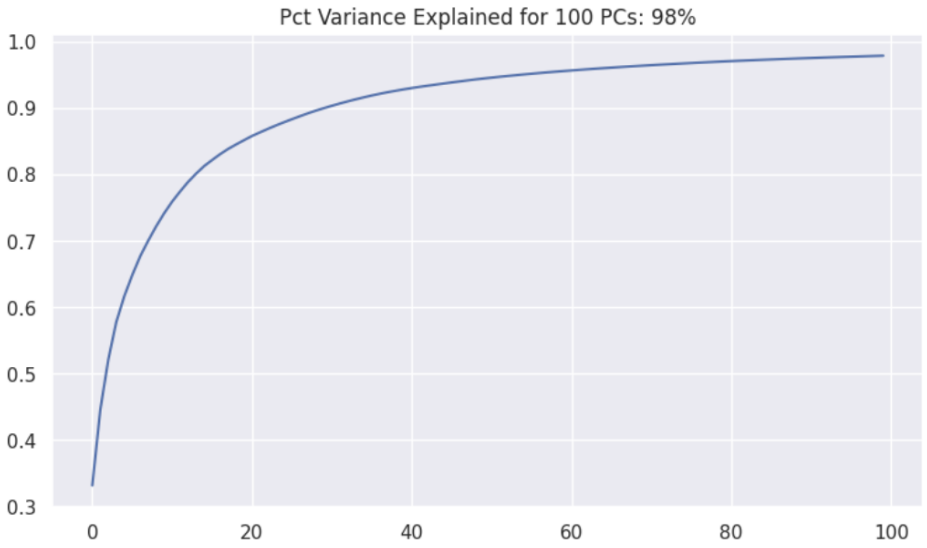

Figure 10: PCA Plot for 100 components, of the final feature vector.

When using the reduced set of features on the classifiers, we saw that the algorithms performed much better with 100 principal components without a significant increase of the runtime.

In addition to PCA, we also applied t-SNE, which computes a non-linear low dimensionality representation of the data. When plotting the first two t-SNE components, we see that some classes are grouped together, but to get a better separation we will need to rely on more dimensions:

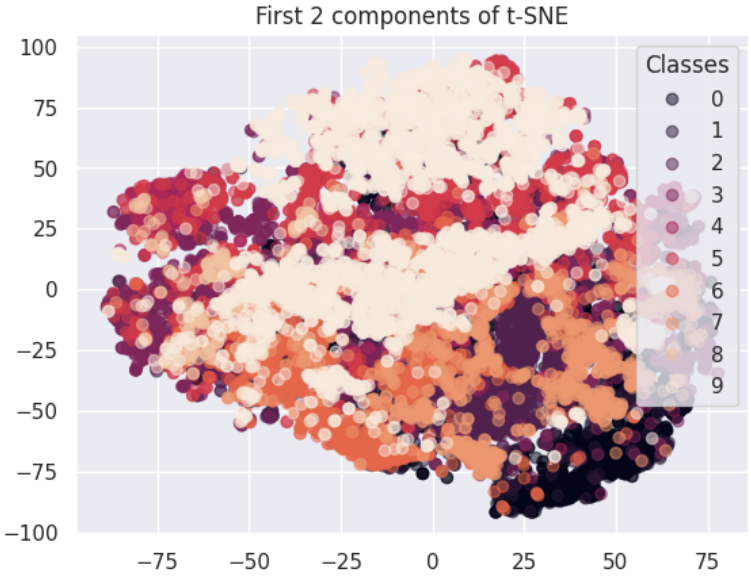

Figure 11: t-SNE Plot of the first two components, of the final feature vector.

Other experiments

In addition to the feature extraction mentioned above, we also attempted other experiments to see if we could leverage other preprocessing techniques and extract other features to consider for our classification models. Due to various reasons (mentioned below), we were not able to leverage these experiments for passing into our models and determining our final feature vector.

“Green World” White Balancing

To account for the varying lighting conditions the images were taken in, we also attempted to preprocess the images with the white balancing technique that follows the gray world assumption (or in this case green world) on a portion of the dataset, by multiplying each image by a scale factor such that the resulting mean of each of the three color channels is equal to the same RGB ratio of the entire dataset portion. However, as shown in Figure 13, this resulted in clipping the color values, which wasn’t desirable because we didn’t want to throw away useful color information.

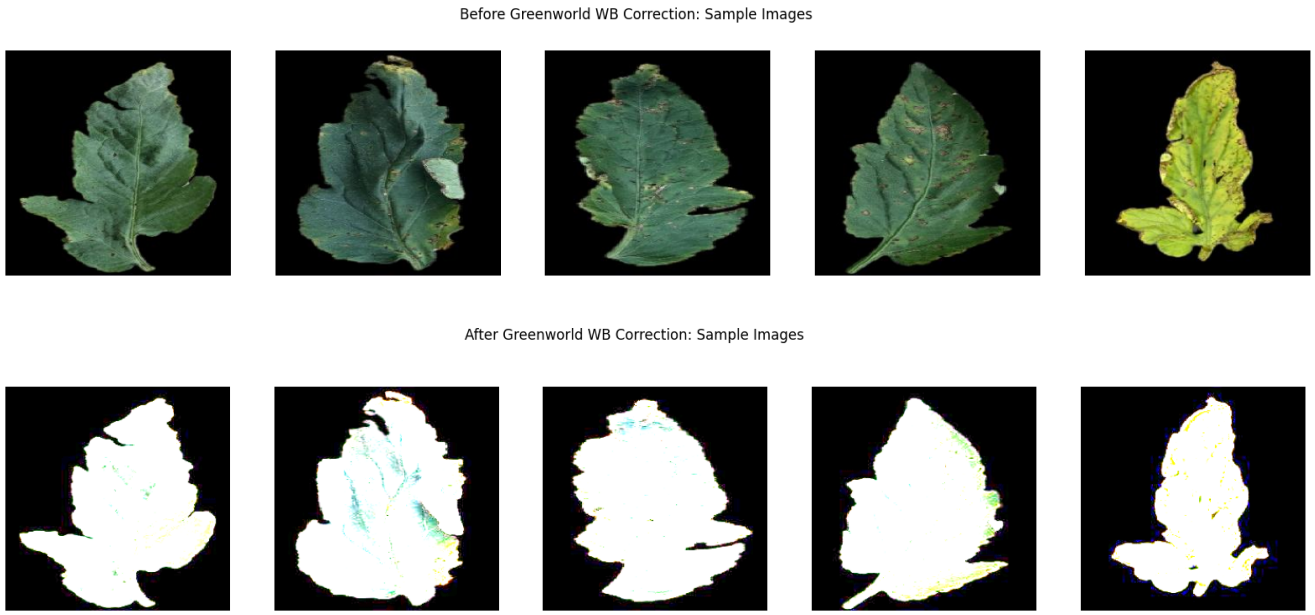

Figure 13: Application of Green World white balancing, which resulted in clipped image values.

Blob Detection

We tried blob detection on our images by plotting histograms of the distribution of blob radii for each image. At first, we were getting too many blobs, so we increased the radii of each blob. But even then, blob detection seemed to capture more noise than signal for disease because the histograms weren't very distinguishable from one category to another. As a result, we did not use blob detection to extract features from the images.

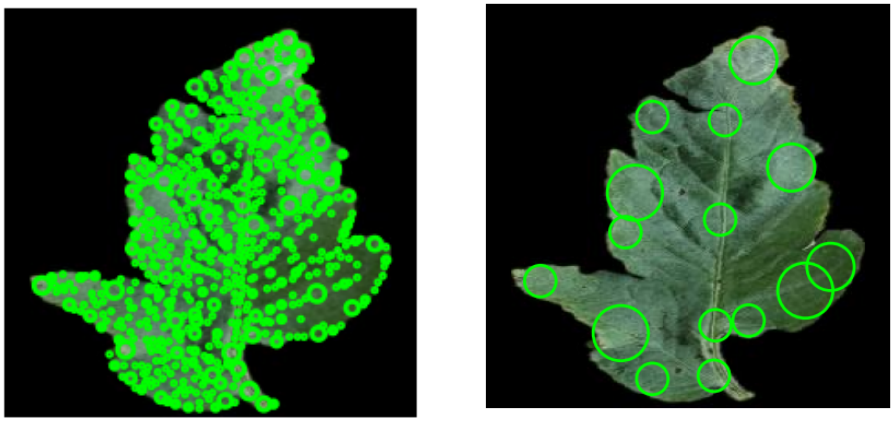

Figure 12: Application of blob detection.

Classifiers Used

For our project, we used three classifiers: Logistic Regression, Feature Net (a deep NN) and a Random Forest.

Logistic Regression is a model that can create a linear boundary to separate the different classes, and is offered by several packages like scikit-learn. Because of this, we were able to leverage the GridSearchCV class (also offered by scikit-learn) and provide it with a set of parameters and various options for each parameter. GridSearchCV would then train a Logistic Regression model for each possible combination of the parameters we provided. From this, we were able to pull the training and validation accuracies from each model that was trained, and handpick the model (and set of parameters applied) that resulted in the best validation accuracy. The LogisticRegression class parameters that we provided to GridSearchCV were the following:

- `penalty` (norm of the penalty): No Penalty, Ridge (l2)
- `class_weight` (weights associated with the classes): No Weights, Balanced Weights
- `max_iter` (maximum number of iterations for the solvers to converge): 100, 250, 500

We also built a NN which we called FeatureNet. The base model had 2 hidden layers of 512 and 256 neurons with a ReLu activation. The output was a softmax layer. We used an Adam optimizer with a learning rate of 0.001 and a standard Cross Entropy loss function. We also tried an extra layer of 64 neurons to compare how the two of them worked. A dropout layer was used to avoid

overfitting on each layer, but even with them, the model slightly overfitted the data. The model with 3 layers performed better than the base one on the validation set.

Last, a Random Forest is an ensemble of decision trees where each has a "vote" toward the final classification and the class with more votes is the one selected. We were able to leverage the scikit-learn implementation of the RandomForestClassifier class and GridSearchCV. The parameters we tested were:

- `n_estimators` (number of trees in the forest): 50, 100
- `max_depth` (maximum depth of the tree): 5, 6, 7

Using these three classifiers, we were able to determine our final feature vector, and ultimately calculate our final test accuracies.

Results

After finalizing our feature vector and also seeing that with just 100 principal components we were able to explain for 98% of the variance, we used the classifiers that were trained for both feature vectors, to generate predictions and gather our test accuracies.

Model	Training accuracy	Validation accuracy	Test Accuracy	Fit time
Performance for L*a*b + DAISY + Canny on full feature vector				
Logistic Regression	88.1%	88.4%	86.0%	97.0s
FeatureNet	89.5%	89.8%	89.0%	78.7s
Random Forest	78.4%	77.5%	76.2%	63.8s
Performance for L*a*b + DAISY + Canny on 100 PCA features				
Logistic Regression	85.1%	85.3%	82.8%	9.8s
FeatureNet	94.8%	92.0%	91.1%	44.4s
Random Forest	67.8%	67.4%	65.3%	5.11s

Table 1: Performance Table for three classifiers and final feature vectors.

As shown on Table 1, for Logistic Regression and Random Forest, the full feature vector performed better than the reduced one. However, it came at the cost of taking a lot more time to fit the training data, for both classifiers.

What we found surprising was that for FeatureNet, we saw both improved performance and less time taken to fit, when working with the 100 components from the PCA of the final feature vector. And it was this test run that presented the best test accuracy.

Efficiency vs Accuracy

The FeatureNet model, when using the 100 components from the PCA of the final feature vector, presented the most accurate results, with a test accuracy of 91.1%.

In the confusion matrix below, for this test run, we see that there are not too many false positives or false negatives, and a vast majority of the results lie on the diagonal of the matrix. The most notable take from this confusion matrix was that the FeatureNet struggled for True Label 1, which maps to Early Blight disease. For this true label, the model was predicting Predicted Label 3, Late Blight disease, instead for a considerable amount of images. It's interesting to note that even though the model predicted a different label, the diseases that mapped to these labels were the same, just at different stages.

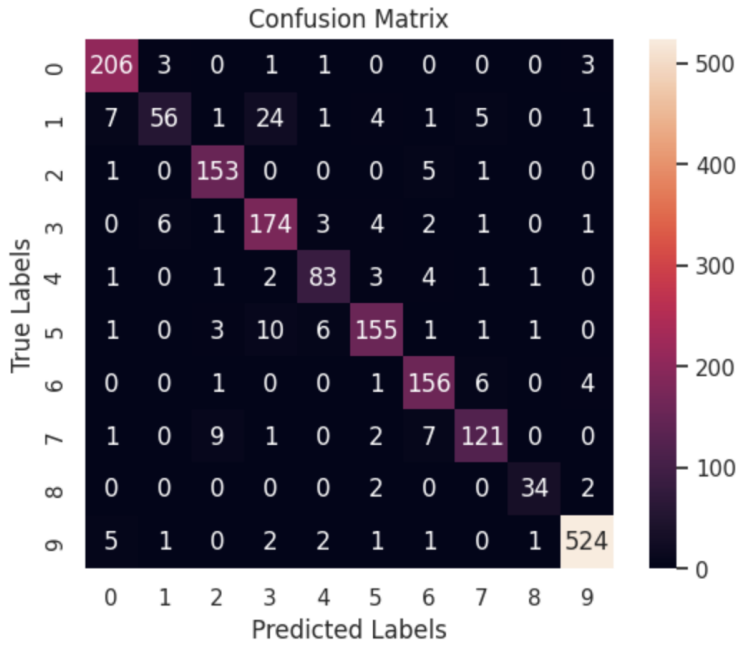

Figure 14: Confusion matrix for FeatureNet model, trained with 100 components from PCA.

Even though the FeatureNet model trained with 100 components from the PCA presented the best test accuracy, it took 44.4 seconds to fit the data during the training.

When looking at which model took the least amount of time to fit the data, it was the RandomForest model trained with 100 components from the PCA. This model took 5.11 seconds to fit the data. However it presented a more disappointing test accuracy of 65.3%.

On the other hand, the Logistic Regression model trained with 100 components from the PCA took only 9.8 seconds to fit the data, and still presented a test accuracy of 82.8%.

When comparing the RandomForest and Logistic Regression models on the reduced feature vector, the Logistic Regression model took roughly 4 more seconds to fit the data, but presented a much better test accuracy (over 17% difference). As a result, we deemed the Logistic Regression model trained with the reduced feature vector to be our most efficient model.

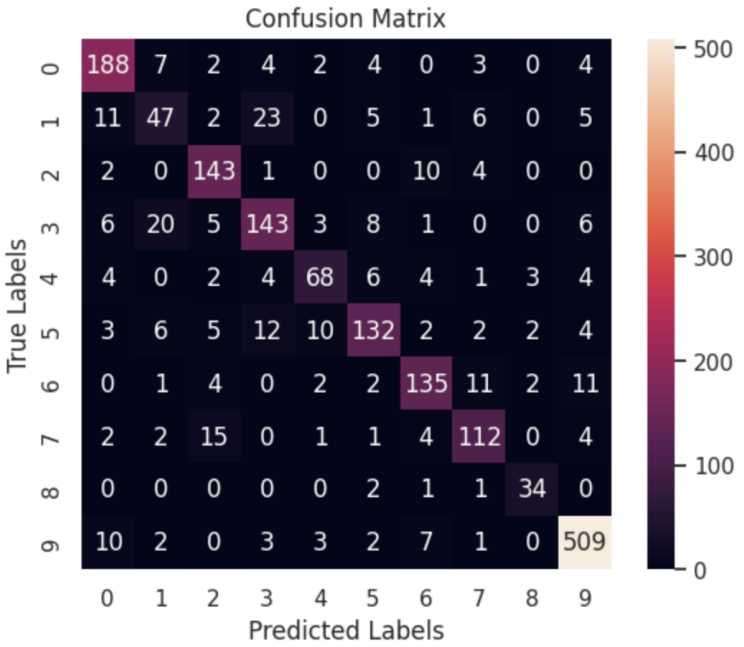

Figure 15: Confusion matrix for Logistic Regression model, trained with 100 components from PCA.

Conclusion

Overall, the feature extraction process produced desirable results for tomato disease classification that were better than we initially anticipated. We found that a simple linear model like Logistic Regression was able to get a very high accuracy given a complex dataset like images. It was also interesting to see the trade-offs that existed within models and how different algorithms performed better or worse than expected.

To improve our models further, we might consider the following for future work:

- We found DAISY to be very useful given its rotation agnostic properties and it seemed to improve model performance significantly. However, we found it took a long time to calculate the descriptor for each image, so we didn't get to experiment with different parameter values as much as we would've liked. To improve the model in the future, we would experiment more with the parameters for DAISY.

- Given that the images were taken in varying lighting conditions, we would further explore preprocessing steps like histogram equalization to adjust and balance the contrast to account for dark shadows and brightness in certain areas within the leaves.
- We might incorporate domain knowledge to help distinguish between different crops by researching topics like plant anatomy and plant pathology, and using that to inform more effective feature extraction specific to plant leaves (e.g researching whether a specific disease type has certain characteristics or patterns in how they affect leaves over time).

Now that we have explored feature extraction and classification of tomato leaf diseases, we might consider exploring how this set of features scales to other plant leaves or other leaf/health based classifications.

References

[1] United Nations Department of Economic and Social Affairs, Population Division (2022). *World Population Prospects 2022: Summary of Results*. UN DESA/POP/2022/TR/NO. 3.

[2] Hughes, D.P. and Salathe, M. (2015) An Open Access Repository of Images on Plant Health to Enable the Development of Mobile Disease Diagnostics. arXiv:1511.08060. http://arxiv.org/abs/1511.08060

[3] Ali, A. (2019, September). PlantVillage Dataset. Retrieved from https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset?resource=download &

[4] Tola, Engin & Lepetit, Vincent & Fua, Pascal. (2010). Daisy: An Efficient Dense Descriptor Applied to Wide Baseline Stereo. IEEE transactions on pattern analysis and machine intelligence. 32. 815-30. 10.1109/TPAMI.2009.77.