Computer Vision Techniques to Diagnose Plant Health

Brian Kazinduka, Ryan Clark, Justin Parsons

Abstract

In this project, we developed and evaluated two machine learning techniques in computer vision to classify leaves as healthy or unhealthy. The project utilized two distinct approaches: classification through clustering and convolutional neural networks (CNN). Our dataset from PlantVillage consists of a collection of tomato leaf images, each labeled as either healthy or unhealthy based on expert botanical assessments. The first approach, classification through clustering, involved placing image pixel colors into a simplified color palette using K-means clustering. The frequency of pixels in each color cluster was used as the features to classify an image as either healthy or unhealthy. The second approach utilized CNNs, a type of deep learning model highly effective for image recognition tasks. We trained a CNN on the labeled images, allowing the model to learn complex patterns and features distinguishing healthy from unhealthy leaves. Both models were evaluated based on various metrics to be discussed with a focus on reducing the amount of misclassified unhealthy leaves. This project not only enhances our understanding of plant health using advanced machine learning techniques but also contributes to agricultural practices by facilitating early detection of plant diseases.

Introduction

In the world of agricultural technology, maintaining plant health is crucial in order to sustain crop production. This project aims to advance the detection of leaf health by classifying images of leaves as either healthy or unhealthy using machine learning techniques. Understanding the distinction between healthy and unhealthy leaves is critical, as it can significantly influence the management of plant diseases and overall crop yield.

The project employs two primary machine learning techniques in order to classify images: clustering and convolutional neural networks (CNNs). Clustering involves grouping a set of objects in such a way that objects in the same group (a cluster) are more similar to each other than to those in other groups. This is a form of unsupervised learning meaning the model can identify structures in the data without any labels. It has been shown that by first running a clustering algorithm described then classifying the features extracted from it with a machine learning algorithm can be quite effective, with results in the past ranging from 88% all the way to 99.6% (Reference 1).

On the other hand, convolutional neural networks (CNNs) are a type of neural network especially effective for image analysis. You can imagine them as automatic feature detectors. They scan through images and learn to identify important patterns, like the differences between healthy and unhealthy leaves, without specific instructions on what to look for. This capability makes them incredibly useful for tasks where visual differences are key, such as determining plant health. This process helps the network capture hierarchical patterns in the data, making it highly suitable for image classification tasks such as this one. Many different types of CNN's such as UNet, AlexNet, VGG16, and more have been proven to show very good results, many of which range from mid to high ninety percent (Reference 3).

The dataset we employed, PlantVillage, contains over 20,000 images of tomato, potato, and bell pepper leaves. The majority of this dataset consists of tomato leaves, which is why we chose to focus on those. Specifically, we chose a healthy subset of 1,596 healthy images and an unhealthy or "late blight" subset of 1,909 images.

The specific goals of this project are to:

1. Develop two types of models capable of accurately classifying images of leaves as healthy or unhealthy. One will use clustering and the other will use a CNN.
2. Implement effective pre-processing techniques to enhance model accuracy, such as image segmentation to isolate the leaf from its background and data augmentation to balance the class distribution.
3. Fine tune and refine the model to prevent overfitting, ensuring that it generalizes well to new, unseen data.

The key challenges include managing the imbalanced data, where the number of images for each class since there are 313 more images of unhealthy leaves. This could potentially bias the model towards the more frequent class (Reference 4). Additionally, ensuring the model does not overfit to the training data is paramount. Overfitting would make the model perform well on training data but poorly on unseen data, which is undesirable for practical applications.

To address these challenges, we utilized data augmentation techniques to balance the dataset and implemented image segmentation to focus the model on relevant features of the leaves. We also employed dropout and L2 regularization amongst other hyperparameters in our CNN to reduce overfitting so that the model would perform well on unseen data. Various machine learning metrics such as precision, recall, F1-score, and the area under the curve (AUC) were used to evaluate model performance.

This project not only aims to classify leaf health accurately but also to enhance our understanding of how machine learning can be applied to real-world problems in agriculture, providing a pathway toward more advanced predictive and monitoring systems for crop management.

Methods

The main aim of this project was to classify images of leaves as either healthy or unhealthy, which falls under the classification type of modeling task in machine learning. Our approach utilized two different machine learning techniques: clustering and convolutional neural networks (CNNs), to address this classification challenge.

Pre Processing:
In regards to preprocessing, the first thing we needed to do was data augmentation. The healthy class had 1,596 images of healthy leaves and 1,909 images of unhealthy leaves. We learned that imbalanced datasets may lead to underfitting or overfitting issues, and the way to remedy this was to perform image augmentation (Reference 4). In order to do this, we wrote a function called generate_extra_images() which randomly selected 313 images from the healthy class and performed transformations on them such as rotation, shifting by height and width, zooming in, and horizontal flip so that the image was similar enough to be classified while being different enough so that the algorithm wouldn't realize it was the same. After that, we stored all the images in 2 separate folders, "Healthy" and "Unhealthy."

Another important point related to preprocessing was image segmentation. This was necessary as before segmentation the experimental models would make inferences based on the images background (Reference 2). Fortunately, the PlantVillage dataset is composed of neutral colored backgrounds that are distinct from leaf colors which were leveraged for a simple segmentation algorithm typically used in plant image processing (Reference 5). The process starts by converting pixel values from the RGB format to LAB. Green is represented in the RGB format when G is sufficiently greater than R and B, as opposed to LAB which denotes the green spectrum with a negative A value. After performing the conversion, the coordinates of each green pixel is recorded in an array. This array is then passed into the scipy.spacial ConvexHull function which generates an equation to determine whether a coordinate pair is within the convex hulls outline. Finally, this equation is applied to the original leaf image to set all pixels outside to black.

After all the pre-processing, we loaded all the images from each of the "Healthy" and "Unhealthy" folders and reshaped them to be 150x150 pixels with 3 channels (red, green, blue) so they could be formatted as our model requires them to be. Using a random 80/20 split from the sklearn function train_test_split(), we split the labeled healthy and unhealthy images into a training generator and testing validation generator using a batch size of 32 through the tensorflow ImageDataGenerator() function. Our data was tested over 5 cross fold validation with some help from sklearn's StratifiedKFold() function, the purpose of which was done to ensure that there was no overfitting and ensure our model's effectiveness.

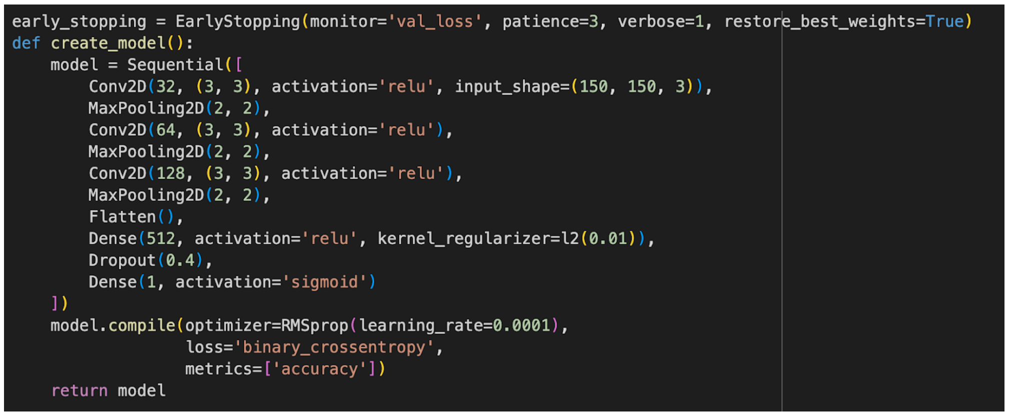
Clustering & Classification:
The goal of clustering and classification technique is to extract color frequencies from a leaf image that can then be put through a classification model to determine if the plant is healthy or unhealthy. The clustering and classification approach was chosen in an attempt to create a ‘simplified’ computer vision model that can accurately classify plants as diseased without the reliance of deep learning. We worked off the assumption that the colors of a leaf would be enough to differentiate a plant as healthy or not. This solution was also ideal as relying on color frequencies removed issues with using models that could not account for positional variance of leaves in the segmented image. To implement our approach, we used K-means clustering for the color frequency extraction and then a K-NN model to classify the image color frequencies as healthy or unhealthy (Reference 1).

The K-means clustering model was used to simplify the entire color spectrum of an image to a reduced color palette. The intuition behind this decision was that using an unsupervised learning algorithm to classify pixel colors would allow for a great reduction in features. The standard RGB color format has 16777216 different color combinations, which in turn means that classification would have to be done on 16777216 features. This is flawed as it would be computationally absurd to process such a large input and the excessive number of features would weaken the ability for most classification models to make an accurate classification (reference 5). Along with this, training a K-means model on the color pixels of the plant dataset meant we would have a color frequency palette with colors most relevant to plant leaf colors.

The clustering model successfully created a color palette that demonstrated a measure difference in color frequency between the healthy and unhealthy class. We reflected on multiple models to perform the next step of classifying based on color cluster frequency. We decided against logistic regression because of the large deviations in color cluster frequencies within each class. We also wanted a model with the flexibility to account for nonlinearity and interactions. Naive Bayes was also rejected due to its assumption of independent features. Testing was also performed with an MLP neural network classifier, but was rejected as using a neural network went against the goal of exploring more primitive models. Ultimately, K-NN was selected as the model as the comparison of color frequencies with neighboring color frequencies seemed intuitive (Reference 1).

K-NN is a non-parametric lazy learner that is optimized by adjusting the number of neighbors an input is compared to, ‘k”. We used the models out-of-sample error rate and area-under-curve as metrics to determine an optimal choice for k. Error rates work as a good baseline metric for the model as it provides the optimal accuracy. However, given the model's use case of diagnosing disease, false negatives are more costly than false positives. This caused us to focus more on the AUC as it can be used to signal how well a model can reduce false negatives without compromising too much accuracy. Much of the time when experimenting, minimum error and maximum AUC both occurred at the same value of k, which ended up being 9.

Convolutional Neural Networks:
The next technique we worked on was building the convolutional neural network.



```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1, restore_best_weights=True)
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.4),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=RMSprop(learning_rate=0.0001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

The layers of the model can be described as follows:

- 3 Conv2D layers. This layer performs a mathematical operation called convolution, which involves sliding a filter of size 3x3 (Reference 3 and 4) in our case over the input image and computing the dot product of the filter and the local regions of the input. Each filter in a Conv2D layer is designed to detect specific features such as edges, textures, or patterns in the input image. The layers have 32, 64, and 128 kernels respectively. It expects images of size 150x150 with 3 channels (red,green,blue).
- MaxPooling2D layers. Each of these layers follows its corresponding Conv2D layer (Reference 4) and reduces the spatial dimensions of the input feature maps. It slides a window over the input feature maps and selects the maximum value from each region covered by the window.
- 1 Flatten layer: Reshapes the multi-dimensional output of the previous layers into a one-dimensional array.
- 2 Dense layers: In a Dense (or fully connected) layer, every neuron in the layer is connected to every neuron in the preceding layer, from which it receives input (Reference 4). Of the 2 layers, there are 512 units and 1 unit respectively. The units receive input from all the units in the previous layer.
- 1 Dropout layer: A type of regularization technique used in neural networks to prevent overfitting. For the 512 unit dense layer, this randomly sets a fraction of the input units (40% in our case) to zero at each update during training time, which helps to make the model more robust and less sensitive to specific weights of neurons. (Reference 4)

The hyperparameter operations of the model can be described as follows:

- ReLU activation: A non-linear function used in Conv2D and the Dense layer with 512 units that outputs the input directly if it is positive, otherwise it outputs zero.
- Sigmoid function: Maps any real-valued number into the (0, 1) range in the Dense layer with 1 unit.
- L2 regularization rate: This adds a penalty equal to the sum of the squared values of the weights multiplied by a small constant (0.01 in our case). It encourages the weights to

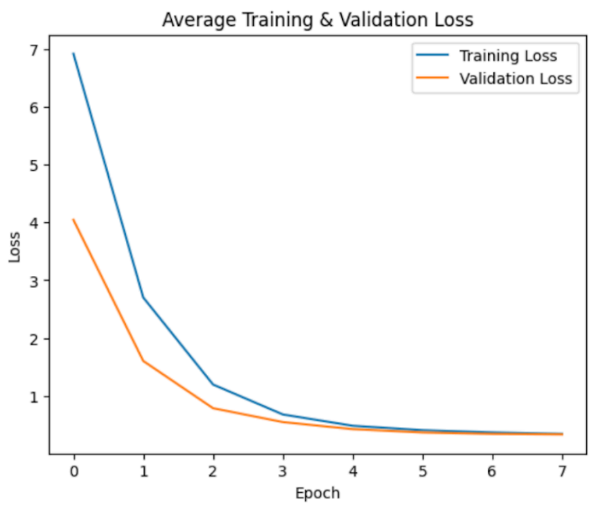
be small (but not necessarily zero), which often results in a more simple and generalizable model (Reference 4).

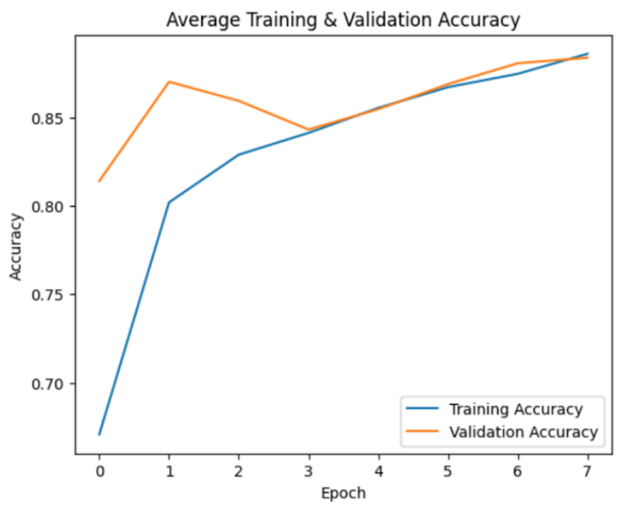
- Optimizer: RMSprop is an optimization algorithm used to train neural networks, where our learning rate of 0.0001 specifies how quickly the model adjusts the weights during training to minimize the loss function. It adapts the learning rate for each parameter by dividing the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight, helping to stabilize the training process.
- Loss: Binary cross-entropy is a loss function commonly used in binary classification tasks, where it measures the difference between the predicted probabilities and the actual binary outcomes (0 or 1). It calculates the loss for each instance by taking the negative log of the predicted probability assigned to the true class, effectively penalizing predictions that are confident but wrong (Reference 3).
- Early stopping: Monitors the validation loss and stops the training process if it does not improve for three epochs, helping to prevent overfitting. It also outputs messages to the console and restores model weights from the epoch with the best performance on the validation loss, ensuring that the model retains the most optimal parameters observed during training (Reference 4).
- Epochs: We chose 8 epochs for training to ensure the model has sufficient opportunity to learn from the data without excessively overfitting, balancing computational efficiency with performance optimization.
- K cross fold validation: Choosing k=5 for k-fold cross-validation provides a good balance between training time and model validation accuracy, ensuring that each fold serves both as training and validation data throughout the process.
- Batch Size: Choosing a batch size of 32 helps to efficiently manage memory usage and computational resources while providing stable gradient estimates during the training of the model (Reference 3 and 4).

Fine tuning these layers and hyperparameters was crucial to our success. Before we chose to write our own model, we first tried some models from pytorch like alexnet that already had their weights trained, but were getting poor in terms of incredibly long training time and underfitting. The model train loss/accuracy would have large fluctuations in between each epoch with no discernable pattern.

In creating our own CNN, we were able to have control over the model complexity. Our training time was cut in half, and the model training/validation accuracy improved significantly. So significantly, that we suspected overfitting. Now, instead of fluctuating loss/accuracies, we were getting near 0 loss and about 99% validation accuracies in each epoch yet poor out of sample performance. Our model was clearly learning the training data too well.

At this point we had to fine tune the model to prevent overfitting. The biggest factors that led to our solution were the segmentation of images, dropout rate, and l2 regularization on the dense layer, all of which have been discussed in detail above. We also tested different amounts in the number of epochs, deciding that 8 was a good balance that we were satisfied with. Running the cross validation was also incredibly useful to ensure that our model was effective on out of sample performance. There were other factors that went into our fine tuning as well, such as learning rate and batch size, but those didn't have as big of an impact on overfitting in particular from our experience.





The provided graphs above illustrate the training and validation loss and accuracy over several epochs of the model. The graph on the left shows the average training and validation loss decreasing significantly over the initial epochs, with the training loss continuing to decrease smoothly, indicating that the model is learning from the training data. The validation loss also decreases but tends to stabilize, suggesting that the model is not overfitting significantly to the training data. The second graph displays the training and validation accuracy, where both metrics increase sharply during the initial epochs. Training accuracy continues to climb slightly, nearing 85%, while validation accuracy plateaus around 83% after the first few epochs. This pattern suggests that the model is generalizing well to new data, maintaining a stable accuracy level after the initial learning period. Together, these graphs are indicative of a well-tuned model that balances learning from training data without overly fitting to it, as evident by the convergence of validation metrics.

The performance metrics we looked to find were area under the curve (AUC), precision, recall, and F1 score (Reference 4).

- AUC: The area under the ROC curve (a graphical representation of a classifier's performance). It is valuable because it provides a single measure of performance across all classification thresholds, making it useful for evaluating models where the optimal threshold is not known in advance or varies by context. For example, different thresholds might be needed for different plant diseases, and a high AUC indicates the model's robustness across these scenarios.
- Precision: The ratio of true positive predictions to the total number of positive predictions made. Precision is particularly important in situations where the cost of a false positive is high. For instance, if a false positive results in unnecessary and costly treatments or the

disposal of healthy plants, maintaining high precision would minimize these unnecessary costs and interventions

- Recall: The ratio of true positives to the actual total number of true positive cases in the data. It measures the model's ability to capture all relevant instances. High recall is important when we need to detect every possible case of an unhealthy plant, such as in high-value crops or in preventing the spread of a contagious plant disease. Missing an unhealthy plant (a false negative) could lead to wider spread of the disease, affecting more of the crop.
- F1 Score: The F1-score is the harmonic mean of precision and recall, providing a single metric that balances the two. It's useful when both false positives and false negatives have significant but comparable consequences. It can be very helpful in cases where both identifying every unhealthy plant and avoiding the misclassification of healthy plants are equally important. For example, in a tightly controlled agricultural experiment, where each plant's treatment is expensive and the risk of disease spread is high, optimizing both precision and recall through the F1-score would be essential.

After all this, each model certainly had its advantages and drawbacks.

Classification through clustering:

- Pros
    - Clustering doesn't require labeled data, meaning it is useful when you don't have predefined categories or when labeling data is impractical.
    - It can identify unforeseen or hidden patterns in the data, which might not be evident or predefined.
    - Many clustering algorithms are straightforward to implement and understand, such as K-means or hierarchical clustering (Reference 2).
- Cons
    - Determining the number of clusters can be subjective and often requires domain knowledge or lots of testing.
    - Since it's working with pixel values, images could easily be distorted given certain conditions, such as something as simple as a cloudy day making the image darker.
    - Does not provide explicit class labels. This could lead to somewhat ambiguous results.

Classification through CNN:

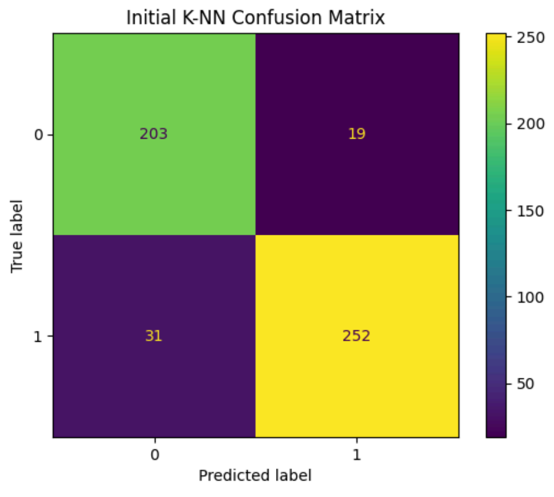
- Pros
    - Generally provide high accuracy in tasks due to their ability to capture spatial hierarchies in data.
    - Automatically detect important features without any human intervention.
    - Robust to variations in the input, such as translation, scaling, and other forms of distortion (Reference 3).
- Cons

- Require a large amount of labeled data to train effectively, which can be a significant limitation.
- Computationally intensive, requiring powerful GPUs and significant resources for training and inference (Reference 3).
- Requires regularization techniques since they can easily overfit on small or insufficiently diverse datasets.

Both these techniques can be considered a success. In a controlled environment in which the pixel values will not be distorted, classification through clustering might make more sense. In a real-world environment, classification through CNN might make more sense. The purpose of our decision to work with these two types of techniques is that we knew they would each have their individual advantages and drawbacks, allowing us to properly compare the two.

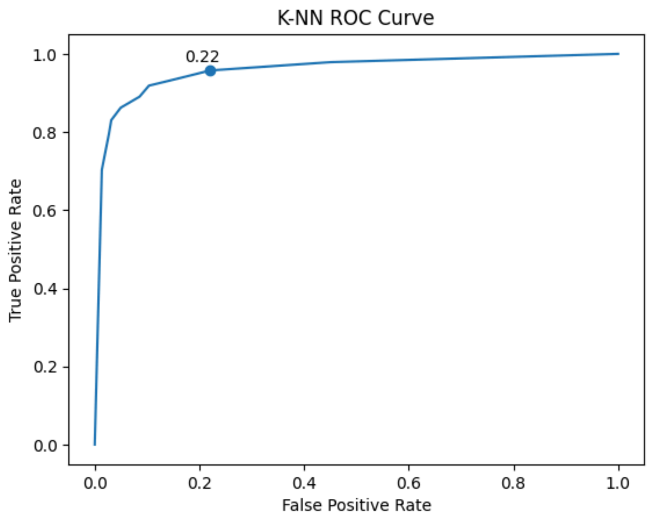
Results and Discussion:

We initially evaluated the k-means model with traditional clustering metrics. The first metric we used was the silhouette score, which measures the quality of clustering as a function that compares inter-cluster and intra-cluster distances. Scores range from -1 to 1, where 1 indicates clusters as well defined, 0 as overlapping, and -1 when samples are assigned to the wrong cluster. The model generally performed poorly with a score of 0.45 being the highest attained with two clusters and a score around 0.27 by 16 clusters. For reference, a silhouette score of 0.5 or greater is considered to be a sign of good clustering. The Davies-Bouldin score was also used, which is the average similarity measure (the ratio of within-cluster distances to between-cluster distances) of each cluster with its most similar cluster. Clusters with scores closer to 0 are considered well separated and compact while those with a value greater than 1 are considered poorer. The trained model scored 0.86 for two clusters and had values close to 1 for all other k. This would usually be a cause for concern, however, we were using clusters for the purpose of creating a color palette to reduce the dimensions of a narrow spectrum of colors. A better metric for this goal is the explained variance ratio (EVR), which helps understand how much variance of the original dataset is preserved after a transformation. For the value of k=2, the EVR was already 85%, which went up to 97% by k=16. The value of 16 was determined to be the optimal value of k as a compromise of capturing a high amount of variance without adding too many dimensions.

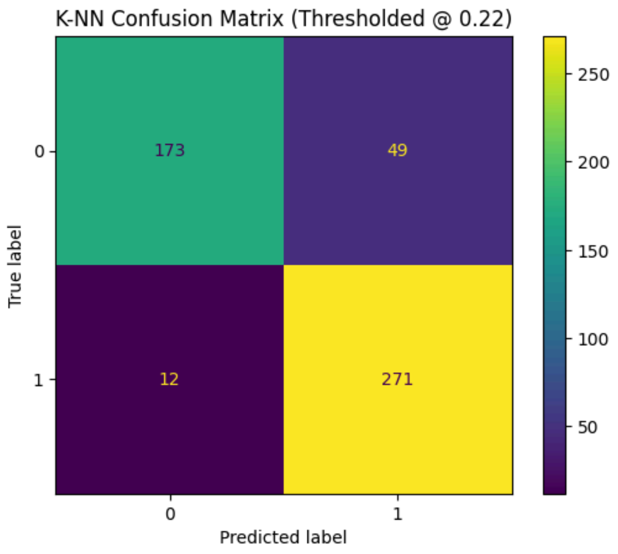


For the k-nn clustering algorithm, we initially attained an accuracy of 0.9010 for k=9. The confusion matrix for the model is shown to the right. Breaking down its confusion matrix into rates corresponds to an FPR of 0.0856, TPR of 0.8904, FNR of 0.1095, and TNR of 0.9144.

As mentioned previously, due to our model being trained for the purpose of disease classification it is important to minimize false negatives (shown in the bottom left quadrant of the matrix). To reduce the false negative rate, the threshold in which an example is classified as diseased is reduced. For k-nn algorithms, this is equivalent to reducing the minimum number of neighbors that have to be diseased.

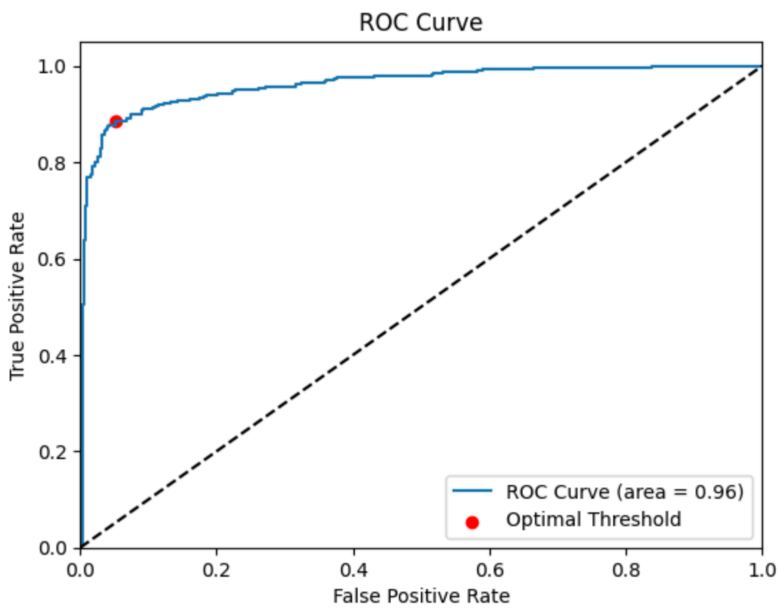


The roc curve shown above is used to find a suitable threshold. Each point on the graph corresponds to the TPR and FPR of a threshold. The FNR is equivalent to 1-TPR, so selecting a point that is higher on the TPR axis will effectively reduce the false negative rate. The point corresponding to 0.22 selected was as a fair compromise to increase the TPR without drastically affecting the FPR. The new confusion matrix is displayed below.



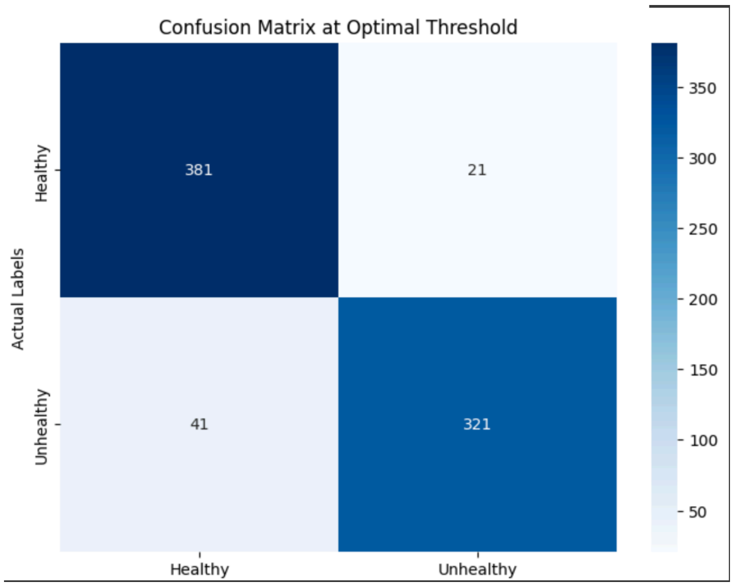
As shown, the number of false negatives was drastically reduced at the expense of an increase in false positives. The new confusion matrix has an FPR of 0.2207, a TPR of 0.9576, a FNR of 0.0424, and a TNR of 0.7793. The overall accuracy is 0.8792.

Next, we evaluated the convolutional neural network.



After training, we obtained the true positive rate, false positive rate, and thresholds using sklearn’s roc_curve() function using the testing labels and prediction labels. The auc() function finds the area under the curve of 0.96 given the true positive and false positive rates as parameters. The optimal point is where you minimize the difference between the true positive and false positive rate, which we found at threshold 0.15 to plot the red dot.

As mentioned, the area under the curve is 0.96, which is very close to 1. This suggests that the model has excellent discrimination ability between the positive and negative classes. A high AUC value implies that the model can correctly classify the positives and negatives with high confidence. The shape of the curve (far above the diagonal line which represents a random guess) confirms the model's strong capability in distinguishing between classes across different thresholds.



Based on the same parameters used for roc_curve(), we can use sklearn's confusion_matrix() function to achieve the results shown above. We can use the confusion matrix to find the metrics we seeked evaluate.

Precision = TP / (TP+FP)
= 381 / (381+21)
= 0.94

This means that 94% of the examples predicted by the model as unhealthy were actually unhealthy. This means that we are reducing the amount of false positives very well which we expected since for most of the leaves it is easy to tell that they are healthy by taking a quick glance.

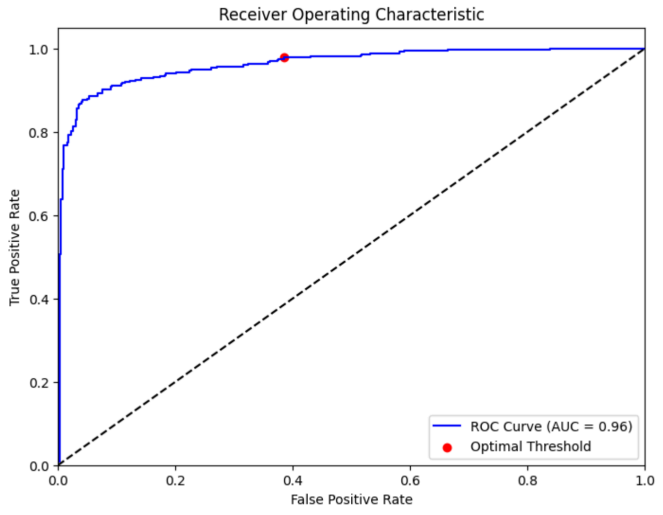
Recall= TP / (TP+FN)
= 381 / (381+41)
= 0.89

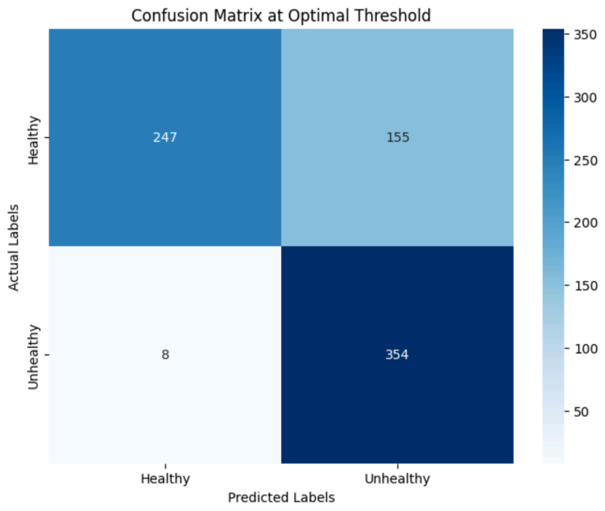
This means that 89% of all the actual unhealthy cases were successfully identified.This means that we are doing a very good job of reducing the amount of false negatives, but not as good as the false positives. This is what we expected as some of the leaves are a bit more difficult to tell that they are unhealthy, however our goal was to prioritize reducing the number of false negatives as they can be more dangerous in terms of disease spreading. This issue will be addressed in a moment.

F1 Score = 2×(Precision+Recall / Precision×Recall)
= 2×(0.94+0.89 / 0.94×0.89)
= 0.91

We were very happy with a F1 score of 0.91 is high, indicating a strong balance between the model's accuracy and its completeness in capturing positive cases. This suggests that the model is both reliable and comprehensive, making it effective for scenarios where it is important to both correctly identify as many true cases as possible and to maintain a low rate of false positives.

Overall, all these metrics demonstrate a very good out of sample performance, meaning that our many efforts to reduce overfitting while maintaining a good classification learning algorithm were successful.





Going back to the point where we discussed wanting to lower false negative rates, we experimented with weighting the false positive rate less in order to move up along the curve in order to reduce the amount of false negatives. As shown in the roc and confusion matrix above, when we move the red dot to the right we get a significant amount less false negatives, although it comes at the cost of way more false positives.

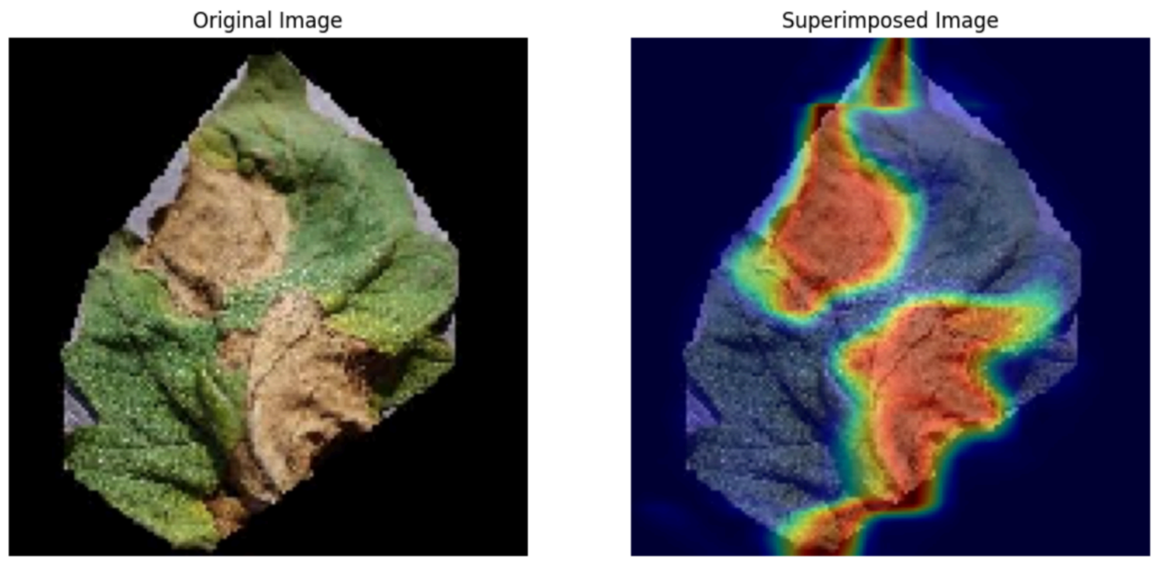
$$\begin{aligned} \text{Precision} &= TP / (TP+FP) \\ &= 247 / (247+155) \\ &= 0.70 \end{aligned}$$

$$\begin{aligned} \text{Recall} &= TP / (TP+FN) \\ &= 247 / (247+8) \\ &= 0.98 \end{aligned}$$

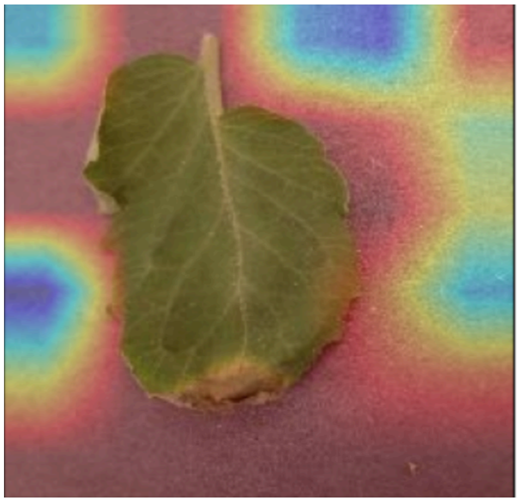
$$\begin{aligned} \text{F1 Score} &= 2\times(\text{Precision}+\text{Recall} / \text{Precision}\times\text{Recall}) \\ &= 2\times(0.70+0.98 / 0.70\times0.98) \\ &= 0.81 \end{aligned}$$

Obviously such a shift leads to worse F1 score despite an improved recall, but the important thing to take away is that using this model and an roc curve, you can adjust the threshold depending on your specific need. Since we need less false negatives to classify plant leaves, we might want to move up the curve, but the point at which the increasing amount of false positives becomes too detrimental would be a decision for an agricultural or botany expert.

Lastly, we implemented Grad-CAM (Gradient-weighted class activation map). This is a technique for making CNNs more transparent by visualizing the regions of input that are important for predictions from these models. This method aids in understanding and interpreting the model's behavior, making the neural network's predictions more transparent and trustworthy (Reference 3).



This picture further demonstrates that the model is working well, because the spots on the leaf that would obviously make it an unhealthy one are highlighted on the heatmap. This wasn't always the case. Before segmentation, when we were getting much worse results the Grad-CAM looked like the image below:



In this example, the model is taking the background too heavily into account, which is how we knew we needed to implement segmentation.

Conclusion:

In this project, our main objective was to classify images of leaves into healthy and unhealthy categories using machine learning techniques, specifically clustering and convolutional neural networks. We successfully achieved our goal, demonstrating the effectiveness of both techniques in different scenarios, which is a significant achievement for agricultural and botanical applications.

The most important factors in determining the results included the handling of an imbalanced dataset, the segmentation of images to focus on the leaves, and the fine-tuning of the model to prevent overfitting. The use of dropout and L2 regularization played critical roles in enhancing the model's generalization to new, unseen data.

Reflecting on our approach, the experience has illuminated several aspects that could be optimized. For instance, better initial handling of the data imbalance and more sophisticated image preprocessing could potentially improve model accuracy and efficiency. Additionally, while our models performed well, exploring hybrid approaches or more advanced neural network architectures could yield even better results.

For someone who would be interested in extending our work, one natural next step would be to apply the developed models to a broader range of plant diseases rather than binary healthy vs unhealthy or to integrate these models into a real-time monitoring system for crops. Another avenue could be to explore the integration of environmental data, such as weather conditions, to predict leaf health proactively.

In closing, this project not only advanced our understanding of machine learning applications in plant health but also provided a robust framework that can be adapted and expanded for broader agricultural technology uses. The integration of such technologies could revolutionize how we detect and manage plant health, leading to more sustainable and productive agricultural practices.

References

1. **Title**: *Plant leaf disease detection using computer vision and machine learning algorithms*
    a. **Author(s)**: Sunil S. Harakannanavar, Jayashri M. Rudagi, Veena I Puranikmath , Ayesha Siddiqua, R Pramodhini
    b. **Publication Venue**: Global Transitions Proceedings
    c. **Volume and Page Numbers**: Volume 3, Issue 1, June 2022, Pages 305-310
    d. **Date**: June 2022
    e. **Publisher**: 2024 KeAi Communications Co. Ltd
2. **Title**: *Plant Leaf Disease Detection Using Computer Vision Techniques and Machine Learning*
    a. **Author(s)**: Kalpesh Joshi, Rohan Awale, Sara Ahmad, Sanmit Patil, and Vipul Pisa
    b. **Publication Venue**: ITM Web of Conferences
    c. **Volume and Page Numbers**: Volume 44, 2022/01/01, Pages 1-6
    d. **Date**: January 2022
    e. **Publisher**: EDP Sciences
3. **Title**: *Deep learning-based segmentation and classification of leaf images for detection of tomato plant disease*
    a. **Author(s)**: Muhammad Shoaib, Tariq Hussain, Babar Shah, Ihsan Ullah, Sayyed Mudassar Shah, Farman Ali, and Sang Hyun Park
    b. **Publication Venue**: Frontiers in Plant Science
    c. **Date**: 07 October 2022
    d. **Publisher**: Copyright © 2022 Shoaib, Hussain, Shah, Ullah, Shah, Ali and Park
4. **Title**: *Real-time plant health assessment via implementing cloud-based scalable transfer learning on AWS DeepLens*
    a. **Author(s)**: Asim Khan,Umair Nawaz,Anwaar Ulhaq,Randall W. Robinson
    b. **Publication Venue**: PLoS ONE
    c. **Date**: December 17, 2020
    d. **Publisher**: © 2020 Khan et al.
5. **Title**: Diagnosis of grape leaf diseases using automatic *K*-means clustering and machine learning
    a. **Author(s)**: Seyed Mohamad Javidan, Ahmad Banakar, Keyvan Asefpour Vaklian, Yiannis Ampatzidis
    b. **Publication Venue**: Smart Agricultural Technology
    c. **Date**: February 2023
    d. **Publisher**: © 2024 Elsevier B.V.