Image-Based Fruit Classification

Team: Joker
Team Members: Jiacheng Zhou, Ruitao Shen, Shihang Liu, Siwei Guo, Yuanchi Guo

1. Introduction

In real life, there are many similar-looking fruits or vegetables, and the inability to clearly distinguish between two similar vegetables often causes confusion. For example, two fruits that are too similar can lead to mislabeling in grocery stores. In order to solve such problems, we hope to design and implement a machine learning model to classify various fruits efficiently and accurately by analyzing their visual features in images, thereby directly offering practical solutions for enhancing the accuracy and efficiency of fruit identification.

Our original fruit image dataset is sourced from Kaggle, created by Mihai Oltean and last updated two years ago. The dataset comprises 90,483 images, each depicting a single fruit or vegetable. It encompasses a total of 131 classes representing various fruits and vegetables. The reason why we selected this dataset is its unique characteristic of having a noise-free background.

2. Exploratory Data Analysis

2.1 Class Distribution

There are a large number of different categories of fruits and vegetables in our data set. We first performed a class distribution on this dataset (Figure 1).

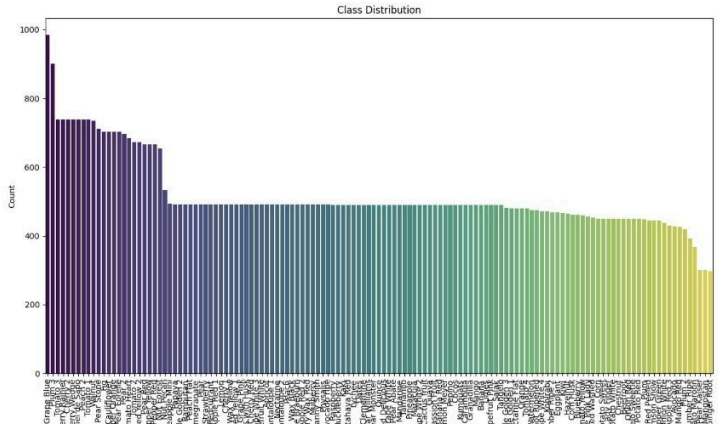

Figure 1: Class Distribution of Images

2.2 Greyscale

Figure 2 is a histogram representing the frequency distribution of pixel intensity values across the grayscale images (Figure 2). It provides a quick visual summary of the grayscale levels present in the images. knowing the distribution of pixel intensity could help in choosing the right thresholding techniques to separate the objects (fruits) from the background.

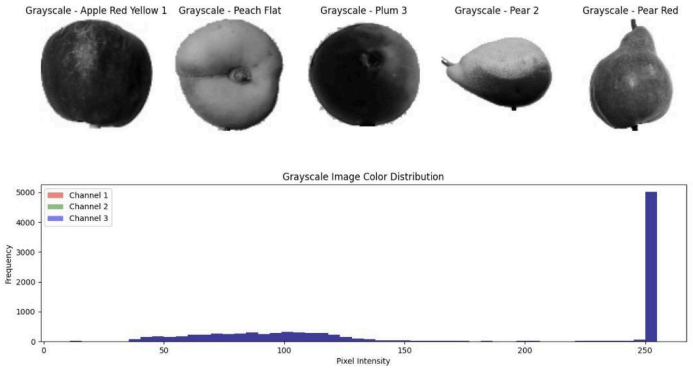

Figure 2: Greyscale Visualization of Images

2.3 Reshape and Check Sizes

In this data set, there are different fruits and vegetables. Although the backgrounds are all white boards, the sizes are still very different. In order to help us with subsequent analysis, we reshaped all the pictures here so that all the pictures are in the same position size (Figure 3).

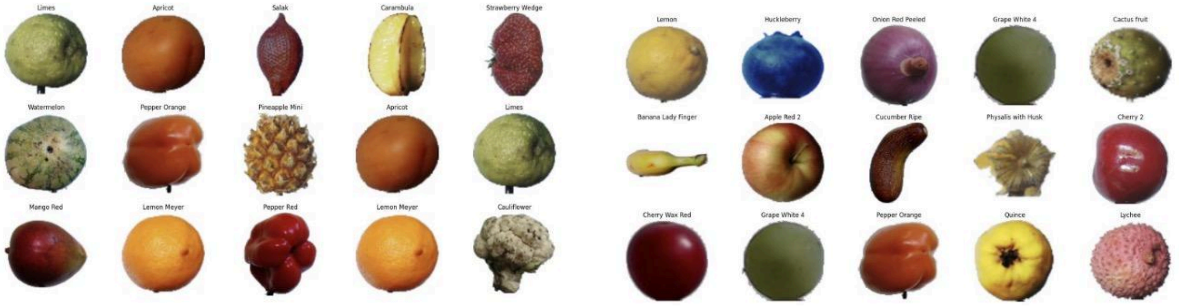

Figure 3:Example Size after Reshaping

At the same time, after reshaping, we checked and confirmed all images are of the same size (Figure 4).

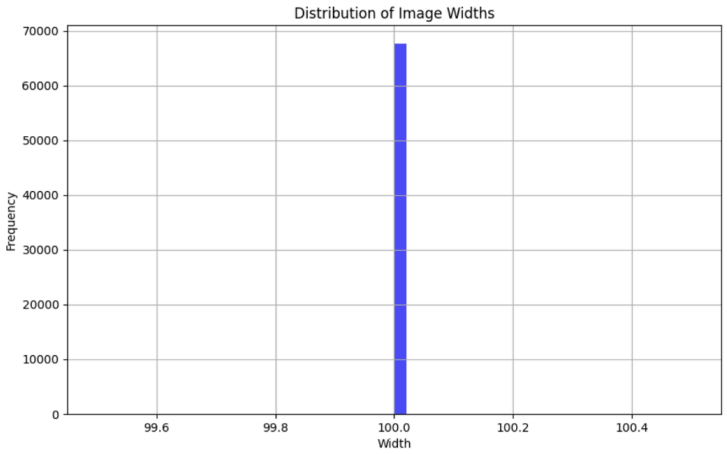

Figure 4: Image Sizes Confirmation

3. Dimension Reduction

3.1 Principal Component Analysis

Our dataset comprises images of various fruit families. To perform dimension reduction, we initially scaled all images and then flattened each one and transformed all image data into a matrix, where each array represents the information of an individual image.

Based on a literature review, we embarked on utilizing PCA for dimension reduction. However, PCA did not yield distinct clusters of fruits from different families when applied to the entire dataset encompassing all families. The top 10 principal components explain 74.23% of the total variance, and achieving over 90% variance explanation necessitates the utilization of 60 principal components (Figure 5).

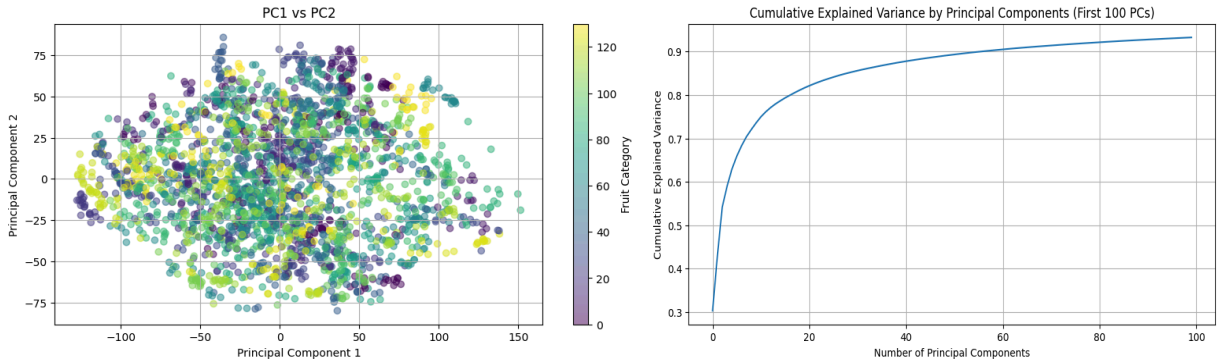

Figure 5: Left: Scatter plot of principal component 1 (PC1) versus principal component 2 (PC2) for all families of fruits in the dataset. Right: Plot showing the cumulative variance explained by the first 100 principal components.

Nonetheless, when PCA was applied to limited fruit families, such as those intended for potential binary classification, the PC1 versus PC2 plot reveals two distinct clusters, which demonstrates the effectiveness of PCA in capturing underlying patterns or structures within our dataset. The top 4 principal components can account for over 90% of the variance. We reconstructed the images based on PCA information with the first four principal components (Figure 6). The overall shape of fruits and main features are preserved.

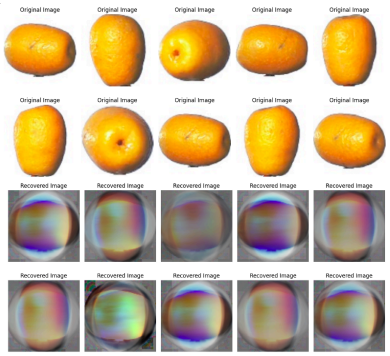

Figure 6: Comparison between ten original images and their corresponding recovered images using PCA with 4 principal components.

3.2 Isomap

Since PCA is a linear dimension reduction method, we also explored Isomap as a nonlinear alternative to better suit our image dataset (Tenenbaum et al., 2000). The Pearson correlation coefficient quantifies the strength and direction of the linear association between these two sets of distances, which is 0.69 in our project and indicates a moderately strong positive linear association between the two variables.

4. Binary Classification

4.1 Support Vector Machine

Binary classification involves classifying two types of fruits. In our project, there are three types of binary classification scenarios: distinguishing between significantly different families of fruits, distinguishing between similar families of fruits, and distinguishing between fruits within the same family but different species with high similarity. For each classification scenario, images are divided into training and testing datasets in an 8:2 ratio.

Support Vector Machine (SVM) was introduced by Cortes & Vapnik (1995) for binary classification. Both nonlinear and linear SVM achieved 100% accuracy with the first 2 principal components when classifying significantly different families of fruits, such as pineapple and peach (Figure 7a). Figure 5b depicts the SVM decision boundaries for classifying kumquats and lemons, two similar families of fruits, with very few misclassifications. The nonlinear SVM test accuracy is 96.65%, while the linear SVM test accuracy is 99.85%, demonstrating considerable great performance (Figure 7b). However, both linear and nonlinear SVM perform poorly when classifying different species within the same family of fruits due to fruits' high similarities. For instance, linear SVM achieved a test classification accuracy of 62.65%, while nonlinear SVM achieved 82.91% (Figure 7c).

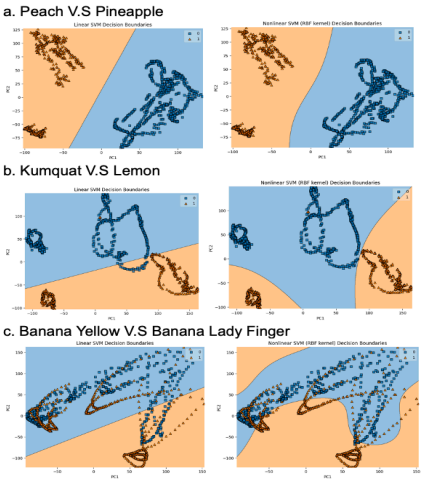

Figure 7: Comparison of linear and nonlinear SVM performance and decision boundaries for three classification scenarios involving (a) significantly different fruit families, (b) similar fruit families, and (c) different species within the same fruit family.

After testing SVM binary classification models on all combinations of fruit classes available to us, we observed that as the similarity between two families of fruits increases, the

classification accuracy of both linear and nonlinear SVM generally decreases. Interestingly, our test accuracy across different cases reveals that as similarity increases, nonlinear SVM consistently outperforms linear SVM, as evidenced by the classification between kumquats and lemons, cocos and pineapples, and persimmon and tomato. However, both linear and nonlinear SVM struggle when classifying two species of fruits from the same family with high similarity.

4.2 K-Nearest Neighbors

To enhance the classification between two species of fruits from the same family with high similarities, we further employed K-Nearest Neighbors (K-NN) (Zhang, 2016). An essential step in applying K-NN is selecting an appropriate value for the number of neighbors to consider. In the case of binary classification between banana yellow and banana lady finger, we observed a significant reduction in classification accuracy when K exceeded 5. Therefore, we fixed our K value at 5.

Upon applying the K-NN model to the same training and testing datasets of banana yellow and lady finger used for SVM model, we observed a substantial increase in test accuracy, reaching 99.76%. To identify misclassifications, we utilized a confusion matrix, revealing that three images of lady fingers were mistakenly classified as banana yellow (Figure 8).

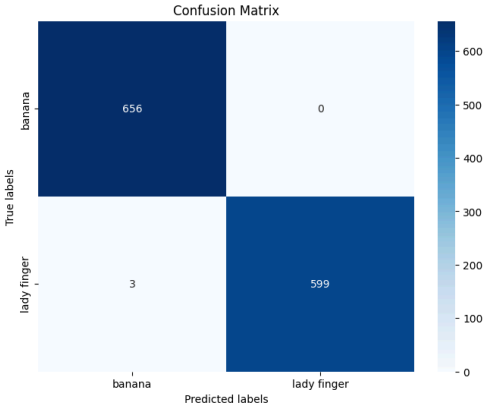

Figure 8: Confusion matrix depicting the classification results between banana yellow and banana lady finger.

To further boost the classification accuracy of our K-NN model, we employed cross-fold validation. We selected 100 folds and recorded the test accuracy after each iteration. While the overall mean accuracy with 100-fold cross-validation stood at 97.92% (+/- 4.04%), many high-fold cross-validation iterations achieved 100% accuracy (Figure 9).

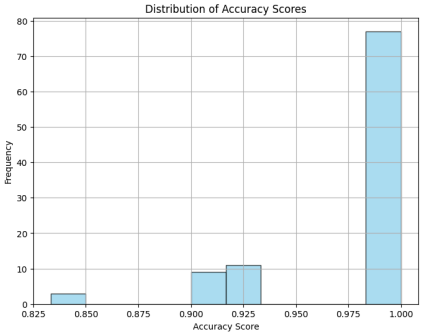

Figure 9: Distribution of accuracy scores obtained from 100-fold cross-validation on binary classification between banana yellow and banana lady finger.

Apart from banana yellow and banana lady finger, we applied K-NN to other binary classification cases. However, K-NN did not consistently yield significant increases in classification accuracy, especially when compared to the nonlinear SVM model. Hence, in binary classification scenarios, there is no absolute superior model, and the selection of K-NN or SVM, the choice of K, as well as the choice of cross-validation fold numbers, depends on the specific families of fruits under consideration.

5. Multi-Class Classification

5.1 Decision Tree

DecisionTreeClassifier from sklearn.tree is initialized with a random state of 42. This is done for reproducibility. For model training, we considered a fit method to train the decision tree classifier (clf) on the training data X_train. The model learned to classify based on features from X_train and the labels from y_train. The load_data function is called with the test_data_directory as an argument to load the test dataset, return the features of the test set X_test, the true labels y_test, and another value which is not used (hence the underscore _). Then, the predict method of the trained classifier (clf) is used to predict the labels for the test set X_test. The predictions are stored in y_pred. In the end, accuracy_score from sklearn.metrics is used to calculate the accuracy of the predictions by comparing y_pred (predicted labels) to y_test (true labels).

By the tree graph, we observed that the decision tree is quite large and complex, as evidenced by the overlapping nodes in the visualization (Figure 10). The classifier used a large number of features to make decisions, and the splits were based on these pixel values. Despite the complexity, the tree has a reasonable accuracy level, 72.28%, though not exceptionally high, suggesting there is room for improvement. The result means that approximately 72 out of every 100 images were correctly classified by the decision tree.

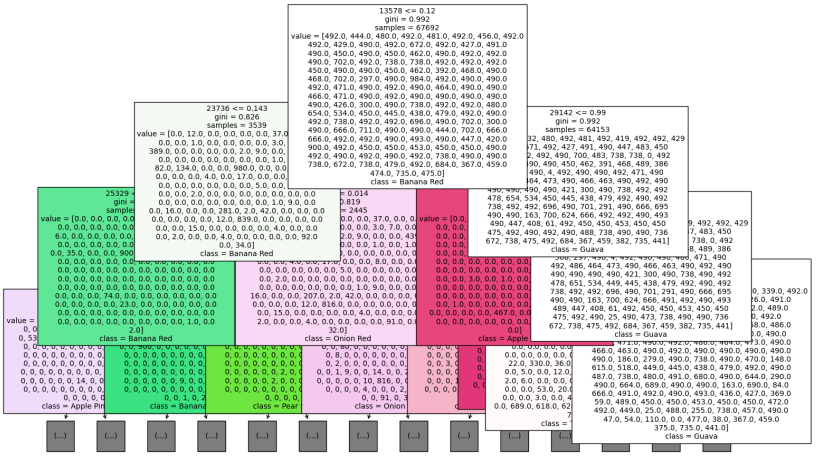

Figure 10: Decision Tree

5.2 Random Forest

To train and evaluate a Random Forest model, we started by importing the RandomForestClassifier and accuracy_score from sklearn. We initialized the classifier with 100 trees and set the random state to 42 for reproducibility. The model was fitted on the training dataset (X_train, y_train) and used to predict the labels of the test set (X_test). The accuracy of the model is computed by comparing the predicted labels (y_pred_rf) with the true labels (y_test). Next, for the evaluation of the model's performance, we imported visualization libraries, matplotlib.pyplot and seaborn, and calculated the confusion matrix. The matrix below is a table that describes how well the classification model has performed against the test data with known true values (Figure 11).

In order to better handle and analyze the confusion matrix data, we converted the numpy array of the confusion matrix into a pandas DataFrame. This conversion facilitates easier manipulation and data access. From this DataFrame, we computed the sum of the predictions (row_sums) and the actual labels (col_sums) across each class to identify which classes are most frequently confused by the model. Using the computed sums, we created a series called most_confused by adding the row_sums and col_sums together. This series provides a measure of the total number of times each class was involved in a misclassification. We then sorted this series in descending order and identified the top 10 indices (class names) that represent the most confused classes. In the end, we filtered both the rows and columns of the original confusion matrix based on the most confused classes, which was then visualized using seaborn to provide a clearer view of where the model struggles the most in classifying the data accurately, thereby offering insights into how to improve the model or focus further analysis.

Random forest classifier builds new trees introducing randomness into the tree construction process to ensure diversity among the trees. The random forest method achieved a high accuracy level of 94.90% on the test dataset. Given the large number of classes and the size of the dataset, a full-size confusion matrix might not be the best way due to readability.

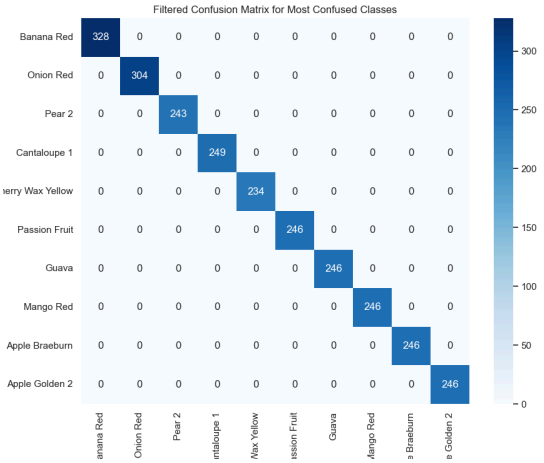

Figure 11: Confusion Matrix

The classes represented, have been perfectly classified, with the numbers along the diagonal indicating the count of true positives for each class. There is no off-diagonal number, which means there were no false positives or false negatives for these classes. This filtered view suggests that the Random Forest model performs exceptionally well for these classes. It implies that each class had a significant number of samples, enough to confirm that the model's performance is robust - at least for these classes.

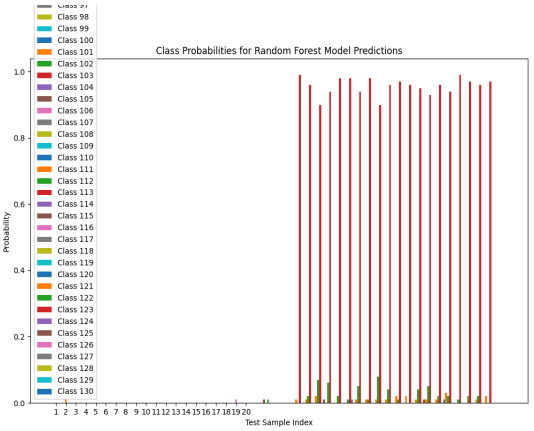

Figure 12: Class Probabilities

Class probabilities for Random Forest model predictions show the class probabilities for 20 different test samples (Figure 12). For each of the 20 test samples, there is one class that has a probability close to 100%, while the probabilities for all other classes are at 0. This indicates that the Random Forest model is very confident in its predictions for each sample.

5.3 CNN

The other Multi-Class Classification we employed was Convolutional Neural Network (CNN). The Convolutional Neural Network is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks (Saha, 2022). In order to

improve the accuracy of our CNN model and avoid overfitting, the first step we did was cross-validation. Overfitting can lead to a model that has difficulty in making accurate predictions from any data other than the training data (Twin, 2021). As a result, we split our dataset into two parts – the training set and the test set. The training set was used to train the model, and the test set was only used to test the model.

Our CNN model consists of three basic types of layers – convolutional layers, pooling layers, and fully connected layers. After running our CNN model, we got the result with a 96.76% accuracy on the training set and 94.90% accuracy on the test set (Figure 13).

```
Epoch 1/10
2116/2116 [==============================] - 866s 409ms/step - loss: 1.9783 - accuracy: 0.4583 - val_loss: 0.4943 - val_accuracy: 0.8543
Epoch 2/10
2116/2116 [==============================] - 836s 395ms/step - loss: 0.4230 - accuracy: 0.8568 - val_loss: 0.2722 - val_accuracy: 0.9197
Epoch 3/10
2116/2116 [==============================] - 857s 405ms/step - loss: 0.2613 - accuracy: 0.9123 - val_loss: 0.2888 - val_accuracy: 0.9174
Epoch 4/10
2116/2116 [==============================] - 847s 400ms/step - loss: 0.2014 - accuracy: 0.9317 - val_loss: 0.1811 - val_accuracy: 0.9510
Epoch 5/10
2116/2116 [==============================] - 851s 402ms/step - loss: 0.1671 - accuracy: 0.9445 - val_loss: 0.1734 - val_accuracy: 0.9517
Epoch 6/10
2116/2116 [==============================] - 851s 402ms/step - loss: 0.1403 - accuracy: 0.9533 - val_loss: 0.1819 - val_accuracy: 0.9550
Epoch 7/10
2116/2116 [==============================] - 832s 393ms/step - loss: 0.1301 - accuracy: 0.9580 - val_loss: 0.2919 - val_accuracy: 0.9249
Epoch 8/10
2116/2116 [==============================] - 843s 398ms/step - loss: 0.1212 - accuracy: 0.9614 - val_loss: 0.1529 - val_accuracy: 0.9576
Epoch 9/10
2116/2116 [==============================] - 842s 398ms/step - loss: 0.1005 - accuracy: 0.9681 - val_loss: 0.1797 - val_accuracy: 0.9523
Epoch 10/10
2116/2116 [==============================] - 1038s 490ms/step - loss: 0.1006 - accuracy: 0.9676 - val_loss: 0.1315 - val_accuracy: 0.9646

709/709 [==============================] - 115s 162ms/step - loss: 0.2289 - accuracy: 0.9490
```

Figure 13: Train set accuracy and test set accuracy of first CNN model

Although the result shows that our CNN model is not overfitting, we still believed that the model accuracy can be boosted. In this case, we came up with several methods to improve this model. The first method we utilized was to increase the image size. Changing the image size from 100 to 150 can make our model capture more details of each fruit image and thus increase the accuracy. In addition, changing the model's learning rate was also a way to increase the accuracy. Applying a high or low learning rate may result in model inaccuracy, so we slightly changed the rate between 0 and 1 to see if there was an optimized result. Data augmentation, such as rotating and shifting images, was another consideration.

With the combination of these methods, our new CNN model achieved a 97.28% accuracy on the training set and 96.27% accuracy on the test set (Figure 14).

```
Epoch 1/10
2116/2116 [==============================] - 1891s 893ms/step - loss: 1.4241 - accuracy: 0.5869 - val_loss: 0.4612 - val_accuracy: 0.8585
Epoch 2/10
2116/2116 [==============================] - 1866s 882ms/step - loss: 0.3134 - accuracy: 0.8942 - val_loss: 0.3390 - val_accuracy: 0.8941
Epoch 3/10
2116/2116 [==============================] - 1875s 886ms/step - loss: 0.2177 - accuracy: 0.9264 - val_loss: 0.1553 - val_accuracy: 0.9518
Epoch 4/10
2116/2116 [==============================] - 1886s 891ms/step - loss: 0.1799 - accuracy: 0.9410 - val_loss: 0.2030 - val_accuracy: 0.9414
Epoch 5/10
2116/2116 [==============================] - 1886s 891ms/step - loss: 0.1485 - accuracy: 0.9517 - val_loss: 0.1347 - val_accuracy: 0.9565
Epoch 6/10
2116/2116 [==============================] - 1927s 911ms/step - loss: 0.1297 - accuracy: 0.9591 - val_loss: 0.1075 - val_accuracy: 0.9708
Epoch 7/10
2116/2116 [==============================] - 1910s 903ms/step - loss: 0.1247 - accuracy: 0.9602 - val_loss: 0.1222 - val_accuracy: 0.9693
Epoch 8/10
2116/2116 [==============================] - 1830s 865ms/step - loss: 0.1054 - accuracy: 0.9658 - val_loss: 0.1393 - val_accuracy: 0.9674
Epoch 9/10
2116/2116 [==============================] - 1857s 878ms/step - loss: 0.0949 - accuracy: 0.9698 - val_loss: 0.1727 - val_accuracy: 0.9622
Epoch 10/10
2116/2116 [==============================] - 1848s 873ms/step - loss: 0.0877 - accuracy: 0.9728 - val_loss: 0.1743 - val_accuracy: 0.9606
```

```
709/709 [==============================] - 228s 322ms/step - loss: 0.1674 - accuracy: 0.9627
```

Figure 14: Train set accuracy and test set accuracy of the second CNN model

6. Conclusion and Future Scope

Overall, our project demonstrated the efficient application of machine learning models in image-based fruit classification. The goal of efficiently and accurately classifying various fruits by analyzing their main features in images was achieved. Integrating multiple fruit classification models into our daily lives can help improve and detect market mislabeling issues.

In binary classification, we succeeded in classifying fruits from highly similar families that are hardly distinguished by the naked eye, using SVM or K-NN. In some cases, K-NN with cross-validation achieved a classification accuracy of 100%, demonstrating significant improvement over both linear and non-linear SVM models. However, the choice of models still depends on which two species of fruits are being classified.

In multiclass classification, the first CNN model we set up only achieved a 96.76% accuracy on the training model. After making some adjustments, our new model achieved a 97.28% accuracy on the training model. Different from the binary classification, the CNN model can reach the goal of classifying multiple fruit types while maintaining the overall accuracy in an acceptable high level.

In the future, we will continue to improve our model through different approaches. For binary classification, we will increase the size of the image dataset by including more families of fruits and then apply additional machine learning models, or even integrate multiple models together to identify the best-performing model for the majority of binary classification cases. In addition, adjusting the batch size is another improvement we may apply to the CNN model to reach higher accuracy. It took a long time for us to process the training set through the CNN model to see its accuracy result. So we will also modify the model and the size of the training dataset to shorten the potential processing time.

Reference:

Cortes, C. & Vapnik, V. (1995). Support-vector network. Machine Learning, 20, 1–25.

Miao, Z., Gaynor, K.M., Wang, J. et al. Insights and approaches using deep learning to classify wildlife. Sci Rep 9, 8137 (2019). https://doi.org/10.1038/s41598-019-44565-w

M. S. Hossain, M. Al-Hammadi and G. Muhammad, "Automatic Fruit Classification Using Deep Learning for Industrial Applications," in IEEE Transactions on Industrial Informatics, vol. 15, no. 2, pp. 1027-1034, Feb. 2019, doi: 10.1109/TII.2018.2875149.

Naik, Sapan & Patel, Bankim. (2017). Machine Vision based Fruit Classification and Grading - A Review. International Journal of Computer Applications. 170. 22-34. 10.5120/ijca2017914937.

Saha, S. (2022). *A comprehensive guide to Convolutional Neural Networks - the eli5 way*. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Tenenbaum, J. B., Silva, V. de, & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*(5500), 2319–2323. https://doi.org/10.1126/science.290.5500.2319

Twin, A. (2021). *Understanding overfitting and how to prevent it*. Investopedia. https://www.investopedia.com/terms/o/overfitting.asp

Zhang, Z. (2016). Introduction to machine learning: K-Nearest Neighbors. *Annals of Translational Medicine*, *4*(11), 218–218. https://doi.org/10.21037/atm.2016.03.37