



Classification of Fruits and Their Decay State

Prepared for

Dr. Erchin Serpedin and
Teaching Assistant Moin Ahammed

By

Sayed Mohammad Kameil, Abdallah Kafoud, and Yaroub Hussein

ECEN 429: Machine Learning
Texas A&M University at Qatar

April 11, 2022

Abstract

This report highlights the concept of machine learning by incorporating machine learning algorithms and relevant tools for image classification. Utilizing machine learning algorithms and relevant tools, images of fruit are provided as input to then be classified into types of fruit and whether the fruit is fresh or rotten. The algorithms used are convolutional neural network (CNN), support vector machine (SVM), and a combination of the two. Performance of each algorithm is evaluated in comparison to establish the best-fit algorithm. This is a real-life application of machine learning, and the methodologies discussed can be generalized to any type of image classification.

Keywords: machine learning, convolutional neural network, support vector machine, fruit, rotten, images

Table of Contents

	Page
Introduction.....	2
Methods.....	2
Convolutional Neural Network.....	2
Support Vector Machine.....	6
Convolution Neural Network + Support Vector Machine.....	8
Results.....	10
Conclusion.....	16
References.....	17
Code.....	19
Contributions	29

Introduction

The problem of focus is to reduce distribution of rotten fruit as much as possible. This prevents grocery stores from wasting inventory space and reduces produce waste in general as rotten fruits can cause other fruits to rotten quicker through mold.

Over a billion tons of food produced for humans is wasted globally every year. This equates to around a trillion US dollars' worth. The food wasted yearly would be sufficient to feed around two billion people. Current solutions utilize human beings to manually sort out rotten fruit which tends to only be extremely rotten fruit. With humans being inefficient and extremely costly, machine learning is a cost-effective, highly efficient, and reliable escape from this global issue.

The proposed solution that is analyzed and highlighted in this report is to apply machine learning through the utilization of image classification where a classifying algorithm scans images of fruit and determines the type of fruit and its freshness. The fruits are sorted to one of the following six categories: fresh apple, fresh orange, fresh banana, rotten apple, rotten orange, and rotten banana. The algorithms that are tested are convolutional neural network (CNN), support vector machine (SVM), and a combination of the two (CNN + SVM). The algorithms, along with the relevant tools for optimization and logistics, will be implemented through the coding language Python.

Methods

The methodology of this project composes of pre-processing, feature extraction, and the algorithm model that executes the classification. The method of implementation for convolutional neural network (CNN), support vector machine (SVM), and the combination of the two are elaborated next, respectively, along with their corresponding approaches to solve the problem statement.

Convolutional Neural Network (CNN)

The convolutional neural network (CNN) is a deep learning algorithm that takes in an input image, assigns weights and biases to various objects/aspects in the image and uses such to differentiate between the objects/aspects ultimately classifying to respective classes. CNN consists of an input layer, an output layer, and hidden layers that aid in processing and classifying. The hidden layers are composed of convolutional layers, ReLU layers, pooling layers for feature extraction, and fully connected layers where the actual classification takes place before routing to output layer. Figure 1 provides a visual representation of the implementation of the CNN model.

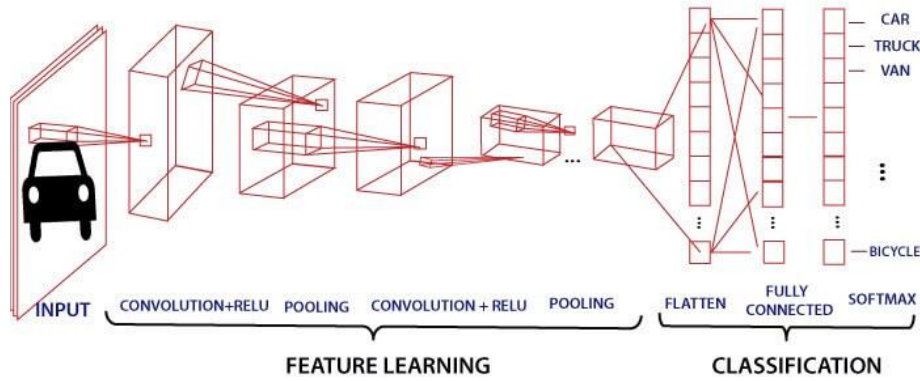


Figure 1. CNN implementation design

Setting up CNN started with preprocessing and preparation of the data. A dataset of different fruit types with different degrees of freshness was obtained from Kaggle (<https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification>). The six classes mentioned previously were used, and the number of images per class for both the training and testing split were modified to be the same. The Train/Test split was 80/20. There were 7800 images used for training, and 1950 images for testing.

The images consisted of different aspect ratios. As a result, all images were converted to a 1:1 aspect ratio by adding horizontal and vertical bars using an online website (<https://bulkresizphotos.com/>). The resolution of all images was also set to 140x140. Next, using the website (<https://ediker.com/>), all the images were cropped in bulk to only retain the center (50x50) pixels. The purpose of retaining only the center 50x50 pixels of the images is to remove any redundant information concerning the background of the fruit image, the texture of the background, its color, etc., and make sure the classifier is trained only on the fruit's texture. The data from the Kaggle repository was already augmented, by rotating, translating, modifying, and intentionally adding noise to the images to better represent real life images and make the classifier more resilient.



Figure 2. Image of fresh apples reflected, rotated, and edited with noise.



Figure 3. Examples of 50x50 pixel images used for training all six classes.

In a CNN, feature selection is done by the CNN, and they are arbitrarily chosen. Features are based on the uniformity and surface texture of the fruit to aid in the classification of freshness, the amount of energy concentrated in each of the RGB channels to aid in the classification of fruit type, and many other features that are arbitrarily chosen to train the neurons. Figure 3 and figure 4 show two images belonging to the fresh apple and rotten apple classes, respectively. Their FFT is also displayed (<https://www.ejectamenta.com/Fourifier-fullscreen/>). For the fresh apple, most of the energy is concentrated around the middle (strong DC component) due to the lack of variation along the spacial coordinates of the image. The rotten apple however has a much larger amount of its energy surrounding the DC component due to the strong presence of variation in the image space.

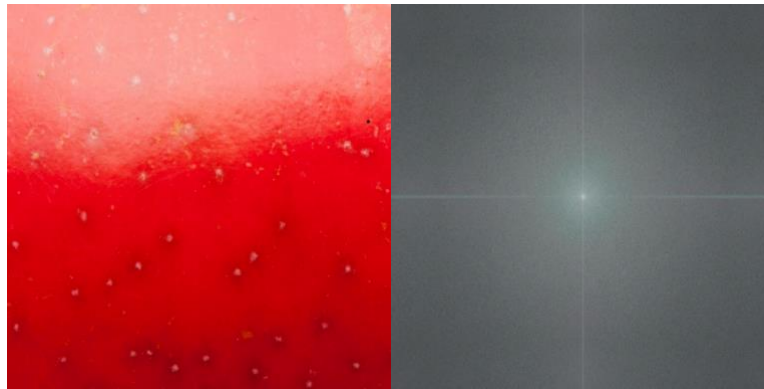


Figure 4. Image of surface of a fresh apple with its FFT representation.

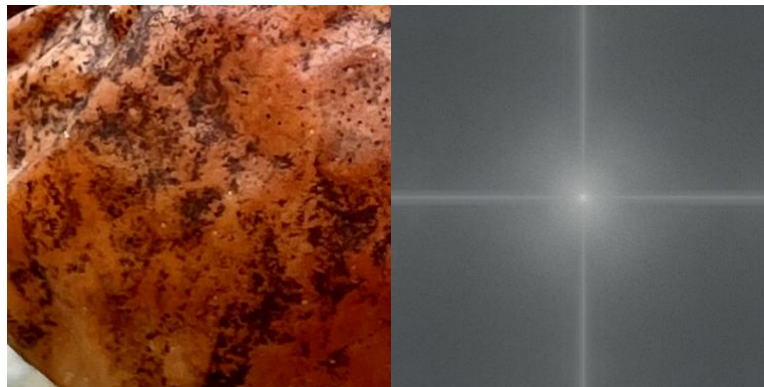


Figure 5. Image of surface of a rotten apple with its FFT representation.

All the classes were split and saved in their respective training/testing directories to be picked up by a Python script. The Python program consists of multiple blocks for preprocessing, training, testing, etc. The code can be found in the Code section of this report.

Coding design was composed of two blocks. In the first block of code, libraries such as NumPy and matplotlib were imported to be used for linear algebra and plotting graphs. OS was imported to reach out to directories where training and testing data were located in. Pandas, Sklearn, Keras, and TensorFlow were used for training of the ML algorithms used. The second block of code defined the training and testing data path and set the training and testing data generators, to be used later. All images were rescaled to ease computational burden by setting the pixel values as non-integers from 0 to 1. All three RGB channels were used for training since it helps encode valuable information about the object.

The CNN model consists of 3 layers: an input layer, the hidden layers, and the output layer. The input layer is the image signal itself, which is a matrix, index of which corresponds to the numerical value of the pixel. Elements in the image matrix have values between 0-255.

The hidden layer consists of two stacks of layers: feature extraction and classification. In the first stack, each feature extraction layer consists of a 2-D matrix of neurons with adjustable weights, which is more ideal for images since images are 2-D matrices as well. Three feature extraction 2D layers were used.

For the second stack, three 1D layers of NN were used for classification. Both the 2D convolutional layers for feature extraction and 1D layers for classification combine to classify input images. Transitioning from 2D convolutional layers to 1D neural network requires the `flatten()` operation to reduce 2D signals from the convolutional layers in the form of matrices into 1D signals in the form of arrays.

Within the convolutional layers, `Maxpooling2D()` is used. The advantage of using `MaxPooling` is that it helps reduce the computational burden on the CPU by decreasing the number of elements within the input matrices through replacing a group of square elements by a single element whose value is the maximum value of the original group. The illustration of `MaxPooling` can be seen in Figure 6.



Figure 6. Visual Aid for `MaxPooling`

For activation functions, `ReLU` was used for every convolutional layer and the first two 1D layers all within the hidden layers, and `Softmax` was used in the final layer since there are multiple output classes. The architecture consists of 6 neurons in the output layer, each of which provide a probability score of the input image belonging to that corresponding class. The `Softmax` equation can be seen in Equation 1.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{Equation 1}$$

The `Softmax` σ of an input vector z is calculated from the standard exponential function for the input vector e^{z_i} being divided by the summation of the standard exponential function for output vector e^{z_j} for each class K in the multi-class classifier.

The shape of the input signal (matrices of pixel values of The RGB channels of the image) was specified after defining the first convolutional layer. The loss function selected was the `categorical_crossentropy` function to handle multi-class classification due to the problem at focus having six classes. The loss function can be seen in Equation 2.

$$-\sum^N y_i * \log \hat{y}_i \quad \text{Equation 2}$$

The negative attribute is to ease the computational complexity. When the optimizer is used it can be processed such that when the gradient the function is used, the negative minimum of it is the same as the positive maximum.

The optimizer function was standard gradient descent because it overcomes the high cost of running back propagation on the entire training set while still converging quickly. The metric to be displayed was the accuracy. The model was trained for 400 epochs with a batch size of 250. Epoch count was determined from trying to find the best fitting for the model and the batch size was determined from an estimation of reasonable merit.

Support Vector Machine (SVM) with Principal Component Analysis (PCA)

The support vector machine algorithm can be used for both classification and regression. For the problem analyzed by this report, it is classification of multiple fruits. Naturally SVM acts as a binary classifier and therefore, to extend it to multiple classes, several techniques were used.

Firstly, for the preprocessing stage, it was necessary to load the training image data set, and then create a path such that the python program can access the image and read it. After this was done, it was then necessary to transform the raw images into 2D Matrix, with each column representing the information contained in one picture as well as its respective class label of which ranged from zero to five, which is for the six classes we would like to classify. Another preprocessing technique that had to be employed was to split the data set into training and testing for the 2D Matrix and the class labels into a 70% training and 30% testing.

After the preprocessing stage, the PCA unsupervised learning algorithm was employed. The purpose of PCA is to reduce the dimensionality of a dataset, this is especially helpful when dealing with large datasets. The way it can do this is by first employing what is called feature scaling was then employed, and its purpose is to take the data and scale it such that the magnitude of one datapoint is not heavily emphasized over a datapoint of a small magnitude. The standard scaler function removes the mean from a dataset and scales it to the unit variance. Where the standard score of a sample x is calculated as:

$$z = \frac{x-u}{s} \quad \text{Equation 3}$$

where z is the score, u is the mean, s is the standard deviation, and x is the datapoint. Using the standard scaler function, this makes the data much more reliable.

Next was to compute the covariance of a matrix, determining the Eigen decomposition of the covariance matrix. The eigen values are then sorted by decreasing fashion to determine which features hold the most variance. Then to project the data into a reduced PCA space, it is done by computing the dot product of the scaled dataset by the most leading eigen vectors of the covariance matrix; in this case they are called the principal components. In the case of the model used, the number of PCA's used ranged from 20, 50 and 100, with 100 providing the best result as it provided much more accuracy for the SVM model.

After the data was completely preprocessed, it was ready to be inserted into the GridSearchCV function. This function was used to obtain the best hyperparameters inputted into it. The inputs were for the C penalty term, the Gamma penalty term, and the kernel function employed as well as fitting the grid search to the training data. The way in which it works is using an exhaustive search. It takes the parameters given to it and tries all possible combinations of the hyperparameters and then computes its performance for each combination, of which the best hyperparameters values are then decided. This is excellent as opposed to doing it manually, however it is computationally intensive and takes an extremely long time.

When hyperparameters were determined, they were fed to an SVM model provided by the scikit library that allows for the input hyperparameters to be selected. The hyperparameters for the model are the C linear penalty term with set value of 100, the radial basis function (RBF) for the kernel function, and the gamma RBF penalty term with set value of 0.01.

The hyperparameters above are namely the C linear penalty term, which is the penalty term given to a datapoint on the wrong side of the hyperplane. Furthermore, another hyperparameter chosen was the radial basis function (RBF) for the kernel function. The kernel function's purpose is to take non-linearly separable data and projects that data into a higher dimension, such that the data can be linearly separable so that a hyperplane can be constructed. The RBF can be referenced by Equation 4.

$$\text{Radial Basis Function} \quad \phi_j = \phi(\|x - c_j\|) \quad \text{Equation 4}$$

The radial basis function (RBF) depends on the distance between on some fixed-point x and some other fixed-point c (typically the center). This function specifically administers the transformation from original feature space dimension to a higher dimensional feature space which allows for efficient formation of the hyperplane. In addition is the gamma parameter which defines how far the influence of one single training example reaches. Such that high values are close and low values are far. This parameter is needed as it can be intuitively interpreted as the inverse of the radius of influence of the support vectors chosen by the hyperplane.

In the SVM model, one issue to explore is the fact that it can only be used for binary classification, or in other words it can only draw a distinction between only two classes. For SVM to be able to perform multiclassification, the one versus all approach was utilized. This approach draws a hyperplane between one category of data against the remainder. Figure 7 illustrates the one versus all approach as a visual aid.

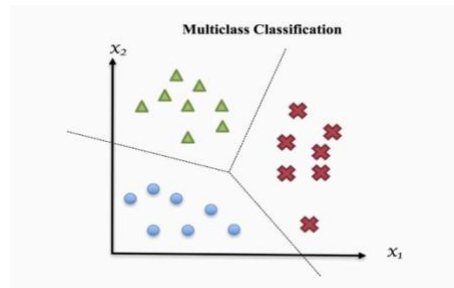


Figure 7. One v. All approach for SVM multiclassification.

For our SVM model, the SVC function (short for support vector classification) can be used to fit our data to this model. It is a function from the scikit-learn library. To give more detail, it uses a one vs one approach which classifies each category against every other category and an illustration of the one vs one SVM model is shown in Figure 8.

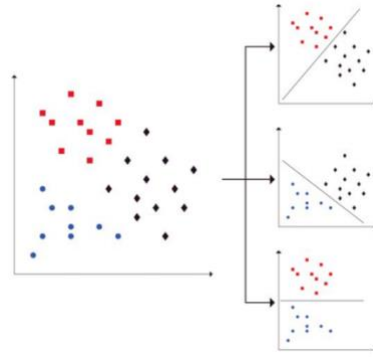


Figure 8. Another One v. All approach for SVM multiclassification.

Convolutional Neural Network with Support Vector Machine (CNN + SVM)

The convolutional neural network (CNN) with a support vector machine (SVM) classifier utilizes the CNN to extract features and the SVM to work as a multi-level classifier with the help of the kernel trick. This hybrid re-utilizes main aspects of the CNN and SVM with PCA models previously mentioned in this report with a few changes that allow the combination of the two algorithms. Figure 9 provides a visual representation of the implementation of this hybrid model.

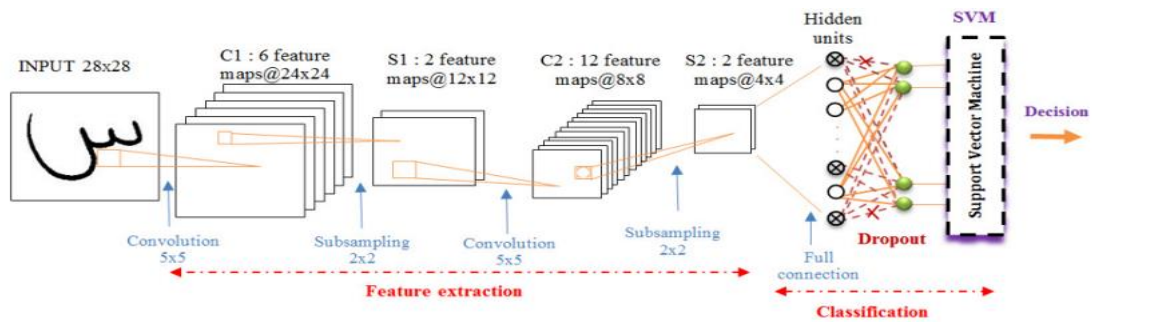


Figure 9. CNN with SVM hybrid model implementation design.

For data preparation, the process remains the same as the CNN model previously, and firstly, mentioned. To briefly revisit the procedure, a dataset of different fruit types with varying levels of freshness (sourced from Kaggle) is inserted as the input. The six classifications of fresh for the three tested fruits and rotten for the three tested fruits are used. Number of images per class for training and testing remained the same as well as the ratio of 80% training and 20% testing. Conversion of images to a uniform aspect ratio, resolution, and focal area of 50x50 pixels was

done using the same method mentioned in CNN. Visual representation of the image preprocessing can be referenced by Figure 1 and Figure 2.

Feature extraction in this hybrid model of CNN and SVM is handled by the CNN alone. The feature extraction process remains the same as elaborated in the CNN-only model. As previously mentioned, CNN arbitrarily distinguishes the features recognized from training of which include features such as uniformity of surface texture and RGB color concentration, along with other arbitrarily decided features. Figures 3 and 4 reflect the differences between an example of a fresh fruit and rotten fruit where their respective Fourier transforms illustrate differing energy concentrations based on the arbitrary features.

Coding design and implementation for feature extraction in this hybrid model directly mirrors architecture of feature extraction used for the CNN only model. The method of creating respective directories for the classes based on training and testing remained the same in which Python was utilized for preprocessing, training, testing, and relevant tools to aid in the process. Libraries utilized for the CNN remained the same as well.

Classification for this hybrid model utilized SVM with the assistance of the kernel trick, a particular activation function, a particular loss function, and a particular optimizer.

The activation function that was utilized for this hybrid model was the Softmax activation function which is also used in the CNN-only model.

The loss function utilized for this hybrid model is the squared hinge loss function shown in Figure 10. This loss function performs a binary classification upon each neuron in the neural network such that the output was either a zero or one instead of probabilities. The SVM used for this hybrid is a “Soft-margin” SVM with the squared hinge-loss and L2 regularization. The “soft-margin” description allows the SVM to make certain number of mistakes while keeping the margin as wide as reasonable to allow other points to be classified correctly. The L2 regularization’s purpose is to reduce the chance of model overfitting and is best fit with this squared hinge-loss function due to the nature of it penalizing the sum of squares of the weights.

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^N \max(1 - \mathbf{y}^{(i)} \mathbf{w}^T \mathbf{x}^{(i)}, 0)^2 + \lambda \|\mathbf{w}\|^2$$

Figure 10. Squared Hinge Loss Function with L2 Regularization

Optimization for the hybrid model differed from the CNN model due to the nature of replacing the classification portion with a SVM. With this hybrid design, the most appropriate optimizer is the Adam optimizer. Compared to the standard gradient descent (SGD) utilized with the CNN-only model, the Adam optimizer converged quicker than the SGD optimizer which yielded overall better optimization therefore performance.

Results

The results of running the CNN, SVM with PCA, and CNN+SVM hybrid models are outlined in the following section. Each model was provided the same datasets. These datasets were images of six classifications which included images of fresh and rotten apples, fresh and rotten oranges, and fresh and rotten bananas. The results will include training/validation accuracies and losses, receiver operating characteristic (ROC) curves, confusion matrices, and overall accuracy ratings along with respective analysis for each performance representation.

Convolutional Neural Network (CNN)

For the CNN model, an overall accuracy of 94.5% was assessed to the model's success and performance.

Figure 11 illustrates the confusion matrix corresponding to the performance of the CNN+SVM model. The confusion matrix provides a summary of prediction results where the number of a correct and incorrect predictions are summarized with count values and broken down by class.

The overall accuracy was calculated utilizing Figure 11 by the following technique. Each blue square represents a class as observed from the axes. The calculation began with taking the value in the blue square and dividing it by the summation of the values in each square within the entire respective row. This is done for each classification therefore each of the six rows. After six accuracy values are calculated for each respective classification, the six accuracy values are added together and divided by six to get the average overall accuracy of 94.5%.

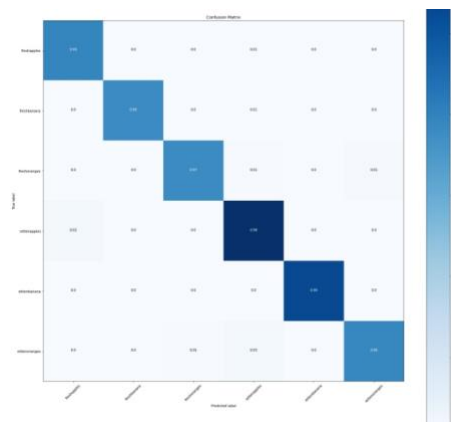


Figure 11. Confusion matrix for CNN in heatmap view with the axes representing the six classes

The heatmap view of the confusion matrix shows that the diagonals are deeply colored, which indicates that the algorithm is running well.

Success in how well the algorithm modeled and classified the provided dataset is represented by training and validation loss/accuracy plots. Referencing Figure 12, the training and validation displayed a negative slope on loss while needing around 250 epochs to converge. The negative slope for the loss on both training and validation portrays that per iteration there is a reduction in misclassifications. This is ideal since low loss rates reflect successful classification.

Referencing Figure 13, the training and validation displayed a positive slope on accuracy while also needing around 250 epochs as well to converge to the overall accuracy of the model. The positive slope for the accuracy on both training and validation portrays that per iteration the algorithm was increasing in accuracy. This is ideal since high accuracy rates reflect successful classification.

Three hours elapsed until training was over for the 400 epochs. However, two hours and 250 epochs would have been sufficient since training and validation loss and accuracy would have stabilized anyway. The model tends to start overfitting if trained for more than 400 epochs, since the training curves seem to exceed the performance of the validation curves. 400 epochs seem like a good compromise between underfitting and overfitting.

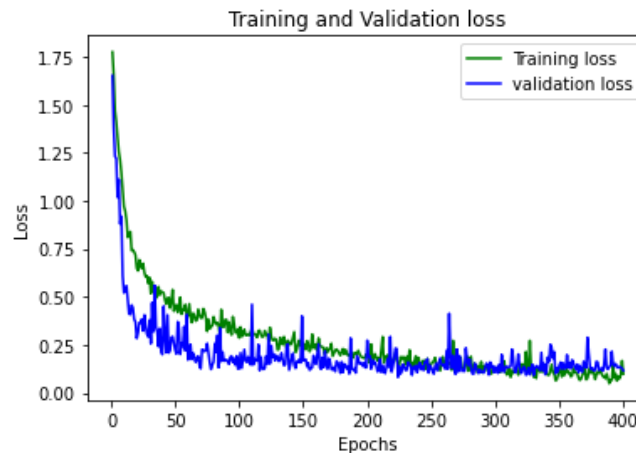


Figure 12. Training and Validation Loss Plot

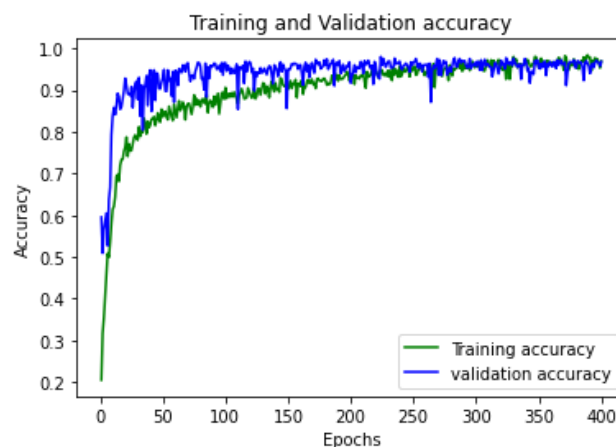


Figure 13. Training and Validation Accuracy Plot

The receiver operating characteristic (ROC) curve is portrayed on a graph and displays the performance of a classification model at all available classification thresholds. The metric that determines whether an ROC portrays a successful classification is from the area under the curve (AUC) which gives a score from 0 to 1. The ROC for the CNN model is shown in Figure 14. The AUC was based on the averages from each six curves from each six classes. The AUC was 0.98 for this CNN model which indicates that the results from the confusion matrix and overall performance is extremely well.

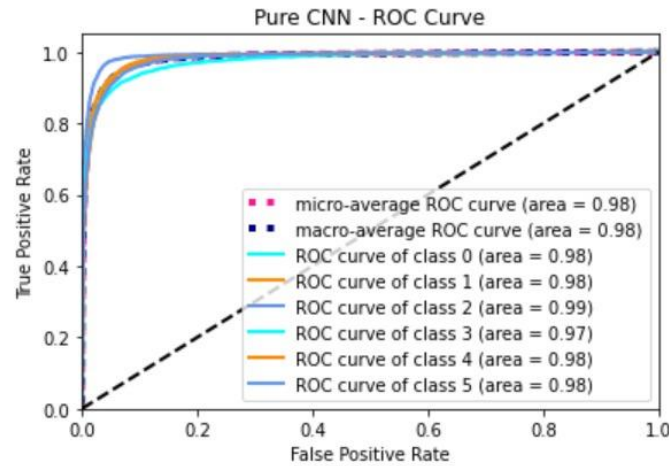








Figure 14. ROC Curve for CNN model.

Numerical thresholds to determine if the fruit is fresh or rotten is portrayed by Table 1. Only one fruit was considered, to determine the model's sensitivity to fruits with different degrees of freshness. The leftmost image in Table 1 is a superposition of the images of two apples: one fresh, and the other rotten. Classification was performed for the superposition of the two images with varying degrees of freshness. As it can be seen, 95% contribution from the fresh apple image was required to classify the image as fresh.

Table 1. Freshness thresholds of two stacked classes with varying contributions from each stack.

Image						
% Of freshness	50%	75%	85%	90%	92.5%	95%
Classified as	Rotten	Rotten	Rotten	Rotten	Rotten	Fresh

Support Vector Machine (SVM) with PCA

For the SVM model, the overall accuracy of 79.2% was assessed to the model's success and performance.

Figure 15 is the confusion matrix for the SVM with PCA model and was used to calculate the overall accuracy of 79.2% using the same technique outlined in the CNN model results.

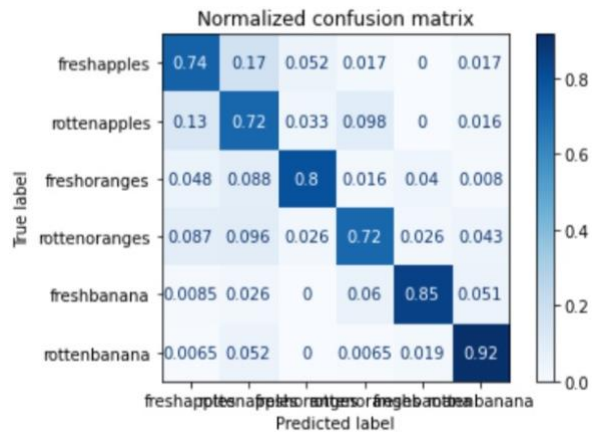


Figure 15. Confusion matrix for SVM with PCA in heatmap view with the axes representing the six classes.

Figure 16 represents the ROC curve with corresponding AUC values for the SVM with PCA model. Referencing Figure 16, the average AUC for the ROC curves was 0.94. This AUC score reflects an overall successful and reasonably well-modeled classification performance.

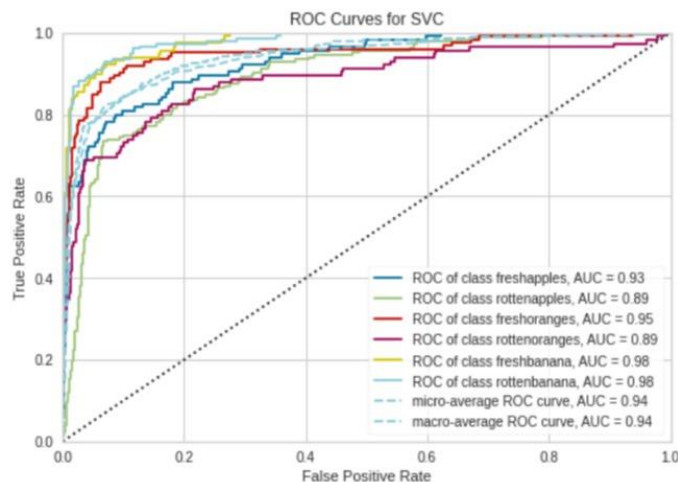


Figure 16. ROC curve for SVM with PCA model.

Convolutional Neural Network with Support Vector Machine (CNN + SVM)

For the CNN+SVM hybrid model, an overall accuracy of 88% was assessed to the model's success and performance.

The confusion matrix for the CNN+SVM hybrid model is represented in Figure 17. The confusion matrix was used to calculate the overall accuracy of 88% using the same technique outlined in the CNN model results and also used for the overall accuracy of the SVM with PCA model.

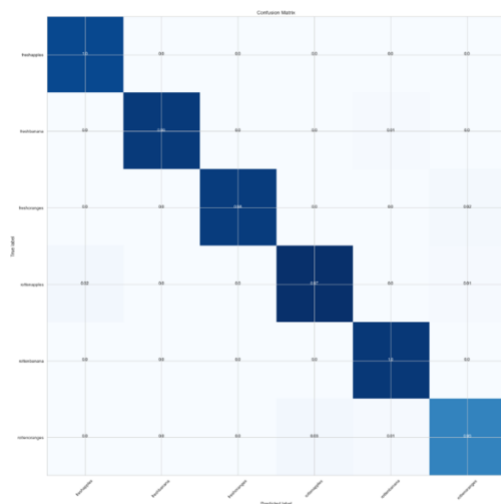


Figure 17. Confusion matrix for CNN+SVM in heatmap view with the axes representing the six classes.

Referencing Figure 18, the training and validation displayed a negative slope on loss while needing around 20 epochs to converge. The negative slope for the loss on both training and validation portrays that per iteration there is a reduction in misclassifications. This is ideal since low loss rates reflect successful classification.

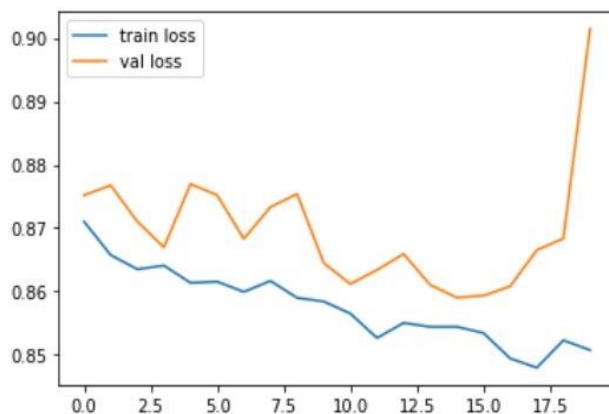


Figure 18. Training and Validation Loss Plot

Referencing Figure 19, the training and validation displayed a positive slope on accuracy while also needing around 20 epochs to converge to the overall accuracy of the model. The positive slope for the accuracy on both training and validation portrays that per iteration the algorithm was increasing in accuracy. This is ideal since high accuracy rates reflect successful classification.

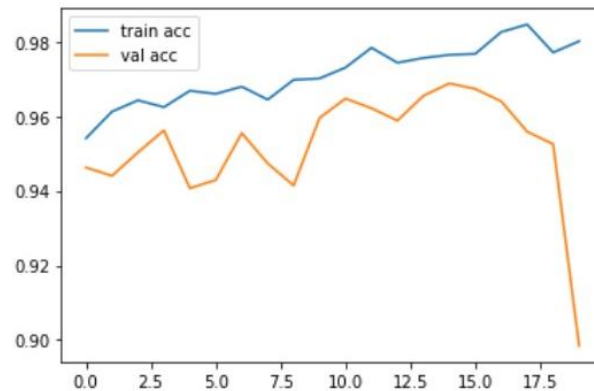


Figure 19. Training and Validation Accuracy Plot

Figure 20 represents the ROC curve with corresponding AUC values for the CNN+SVM. Referencing Figure 20, the average AUC for the ROC curves was 0.99. This AUC score reflects an overall successful and extremely well-modeled classification performance as a perfect performance is a 1.

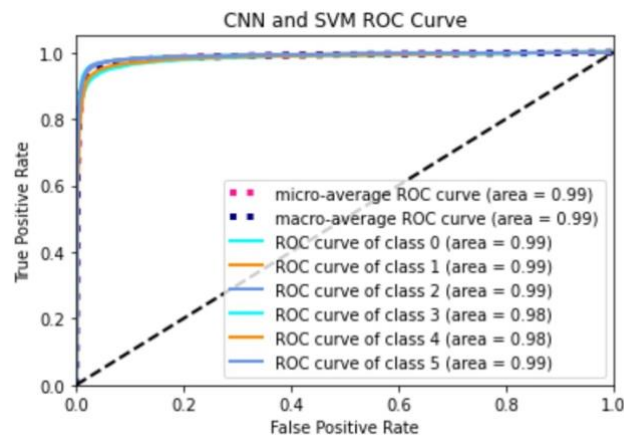


Figure 20. ROC curve for CNN+SVM model.

Conclusion

In conclusion, after running and comparing performances between CNN, SVM with PCA, and the CNN+SVM hybrid models, the model with the best performance for image classification, with the better overall accuracy and ROC AUC paring, was the CNN model.

To elaborate, the performance was determined by evaluating both the overall accuracy and ROC AUC together, not individually, per algorithm provided by the output of running the code for each model. Training and validation plots were not part of comparison due to the generalized trend being consistent for the models. The CNN model had an overall accuracy of 94.5% with an ROC AUC of 0.98. The SVM with PCA model had an overall accuracy of 79.2% with an ROC AUC of 0.94. The CNN+SVM hybrid model had an overall accuracy of 88% with an ROC AUC of 0.99.

CNN was 15.3% more accurate than SVM with PCA and 6.5% more accurate than CNN+SVM hybrid. CNN+SVM hybrid had a better AUC score of 0.01 more than CNN and had a better AUC score of 0.05 more than SVM with PCA. Although CNN did not have the largest AUC score, the difference in overall accuracy weighs far more than the 0.01 difference in AUC scores between the CNN and CNN+SVM hybrid models. With this analysis of differences between the models' performances, the best performing model was CNN with CNN+SVM model being second best performing, and the SVM with PCA model being the worst performing model with the lowest overall accuracy and lowest AUC score.

This conclusion confirms the common conclusion of the convolutional neural network algorithm being the better performing algorithm over support vector machine when being utilized for image classification. The hybrid model was necessary to see the performance from taking the strong feature extraction of CNN and combining it with the strong feature of classification of SVM. It can be concluded that the hybrid model performed significantly well and close to the performance of the pure CNN model.

A strong characteristic of this report and approach of testing two algorithms and a hybrid of the two with image classification helps contribute to exploration of the best performing image classification algorithm for problems across the world. This is valuable as it continues the motivation of finding the most efficient method. This report offers general explanations of the top image classification algorithms to help create the most efficient image classification problems. Other problems could include medical imaging, traffic control systems, and autopilot for vehicles to name a few. At this reports core, the input must be coherent images with proper training and that is what makes this approach so compatible.

References

- Sharma, Devnash. “CNN for Image Classification: Image Classification Using CNN.” *Analytics Vidhya*, 14 Jan. 2021, <https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/>.
- “Big Six Produce Companies - Rural Migration News: Migration Dialogue.” *Big Six Produce Companies - Rural Migration News / Migration Dialogue*, <https://migration.ucdavis.edu/rmn/more.php?id=348>.
- Anna Biju Kuriakose, Feba Siju, Merin K Aji, Thasli Rahim, Chithra Rani P R, 2021, Automatic Fruit Classification and Freshness Detection, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCREIS – 2021 (Volume 09 – Issue 13),
- Brownlee, Jason. “Overfitting and Underfitting with Machine Learning Algorithms.” *Machine Learning Mastery*, 12 Aug. 2019, <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>.
- Rufai, Aminah Mardiyah. “The Support Vector Machine: Basic Concept.” *Medium*, The Startup, 9 Aug. 2020, [LINK](#).
- Elleuch, Mohamed, et al. “A New Design Based-SVM of the CNN Classifier Architecture with Dropout for Offline Arabic Handwritten Recognition.” *Procedia Computer Science*, Elsevier, 1 June 2016, <https://www.sciencedirect.com/science/article/pii/S1877050916309991>.
- Baeldung. “Multiclass Classification Using Support Vector Machines.” *Baeldung on Computer Science*, 25 Aug. 2021, <https://www.baeldung.com/cs/svm-multiclass-classification>.
- Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks-the eli5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- Valentino, Febrian & Cenggoro, Tjeng Wawan & Pardamean, Bens. (2020). A Design of Deep Learning Experimentation for Fruit Freshness Detection.
- Harsawardana, et al. *AI-Based Ripeness Grading for Oil Palm Fresh Fruit Bunch in Smart Crane Grabber*. <https://iopscience.iop.org/article/10.1088/1755-1315/426/1/012147/pdf>.
- Samal, Ashis, et al. “Understanding Regularization for Image Classification and Machine Learning.” *PyImageSearch*, 22 Feb. 2022, [LINK](#).
- World Food Programme. “5 Facts about Food Waste and Hunger: World Food Programme.” *UN World Food Programme*, <https://www.wfp.org/stories/5-facts-about-food-waste-and-hunger>.

-
- Ahlawat, Savita, and Amit Choudhary. “Hybrid CNN-SVM Classifier for Handwritten Digit Recognition.” *Procedia Computer Science*, Elsevier, 16 Apr. 2020, <https://www.sciencedirect.com/science/article/pii/S1877050920307754>.
 - Vadapalli, Pavan. “Using Convolutional Neural Network for Image Classification.” *UpGrad Blog*, 5 Jan. 2022, <https://www.upgrad.com/blog/using-convolutional-neural-network-for-image-classification/>.
 - “Categorical Crossentropy Loss Function: Peltarion Platform.” *Peltarion*, <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy>.
 - “Sklearn.preprocessing.StandardScaler.” *Scikit*, <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>.
 - Brownlee, Jason. “A Gentle Introduction to the Rectified Linear Unit (ReLU).” *Machine Learning Mastery*, 20 Aug. 2020, <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>.
 - Loukas, Serafeim. “PCA Clearly Explained-How, When, Why to Use It and Feature Importance: A Guide in Python.” *Medium*, Towards Data Science, 8 Oct. 2021, <https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e>.
 - “Sklearn.model_selection.GRIDSEARCHCV.” *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.
 - “RBF SVM Parameters.” *Scikit*, https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html.
 - Band, Amey. “Multi-Class Classification-One-vs-All & One-vs-One.” *Medium*, Towards Data Science, 3 Mar. 2022, <https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b>.
 - Kalluri, Sriram Reddy. “Fruits Fresh and Rotten for Classification.” *Kaggle*, 24 Aug. 2018, <https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification>.

Code

The software used to create and conduct this project was Python. Coding the machine learning aspects was done purely through Python. Online tools (see methodology and references) were used for preprocessing the images. All the libraries used in Jupyter Notebook were installed using the pip install command and then imported into their respective model files.

Convolutional Neural Network (CNN)

```
#Import libraries for tensors, image processing, linear algebra, accessing os directories, plotting, and Neural Networks
import numpy as np
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
import os
```

Figure 21. Importing all the libraries.

```
#Rescaling intensity of each subpixel to [0,1] instead of [0,255], to make computation easier
#Constructing training and testing dataset:
train_path = "/Users/black/Documents/Spring 2022/429/Project/Machine learning project/TRAIN/"
test_path = "/Users/black/Documents/Spring 2022/429/Project/Machine learning project/TESTTEST/Class"
train_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size=(50,50),
                                                    batch_size = 20,
                                                    color_mode= "rgb",
                                                    class_mode = "categorical")
test_generator = test_datagen.flow_from_directory(test_path,
                                                  target_size=(50,50),
                                                  batch_size = 20,
                                                  color_mode= "rgb",
                                                  shuffle= False,
                                                  class_mode = "categorical")

#Directory of training data
#Resolution of images
#Number of images used per step within an epoch
#RGB images, Multiple classes

#Directory of testing data
#Resolution of images
#Number of images used per step within an epoch
#RGB images, Multiple classes
#Predictions are not shuffled

Found 7800 images belonging to 6 classes.
Found 325 images belonging to 1 classes.
```

Figure 22. Setting up the training and testing data

```

#Using RELU activation to deal with negative numbers (nonlinearity) in the matrix
#Using MaxPooling on the convolutional matrix to reduce it's size
#Defining 3 feature extraction layers
model = Sequential()
model.add(Conv2D(128, 3, activation="relu", input_shape=(50,50,3))) #filters #Kernelsize #RELU
model.add(MaxPooling2D())
model.add(Conv2D(64, 3, activation="relu"))
model.add(Conv2D(32, 3, activation="relu"))
model.add(MaxPooling2D())
model.add(Dropout(0.50)) #Helps with overfitting
model.add(Flatten()) #Use flatten to turn matrices into arrays and reduce computational burden
model.add(Dense(5000, activation = "relu")) #5000 neurons in this layer with RELU activation
model.add(Dense(500, activation = "relu")) #500 neurons in this layer with RELU activation
model.add(Dense(6, activation = "softmax")) #6 o/p probabilities with softmax activation
model.compile(loss="categorical_crossentropy", optimizer = "SGD", metrics = ["accuracy"]) #Loss fn, Gradient Descend, accuracy
#Start training the NN to fit to the given dataset for 50 training cycles, and save the model to be called later.
hist = model.fit(train_generator,
                  steps_per_epoch = 50,
                  epochs = 400,
                  validation_data = test_generator,
                  validation_steps = 50)
0.9540
Epoch 395/400
50/50 [=====] - 15s 307ms/step - loss: 0.0628 - accuracy: 0.9770 - val_loss: 0.1362 - val_accuracy:
0.9660
Epoch 396/400
50/50 [=====] - 16s 311ms/step - loss: 0.0850 - accuracy: 0.9730 - val_loss: 0.1301 - val_accuracy:

```

Figure 23. Constructing architecture of CNN model and training it with training dataset

```

#Plotting the evolution of loss and accuracy for training and validation data for different epochs
loss_train = hist.history['loss']
loss_val = hist.history['val_loss']
epochs = range(1,401)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

loss_train = hist.history['accuracy']
loss_val = hist.history['val_accuracy']
epochs = range(1,401)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

Figure 24. Plotting the training and validation loss and the training and validation accuracy

```

from keras.models import load_model
import tensorflow as tf
import os
model.save("Fruitmodel.h5") #The model can be called from this
probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
test_generator.classes[6]
predictions = probability_model.predict(test_generator)

```

Figure 25. Utilizing model to create predictions for testing data

```

import numpy as np
from scipy import interp
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.metrics import roc_curve, auc

## Code for Plotting the ROC Curve
## Source: scikitLearn

# Plot linewidth.
lw = 2

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

```

Figure 26. Part 1 of ROC construction for CNN model

```

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(1)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
         .format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
         .format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             .format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('CNN and SVM ROC Curve')
plt.legend(loc="lower right")
plt.show()

```

Figure 27. Part 2 of ROC construction for CNN model

Support Vector Machine (SVM) with PCA

For the PCA and SVM Code, Google collab was used to access the P100 GPU for higher computational power such that the execution time of the code is drastically reduced.

```
[ ] ## Importing Necessary Libraries for PCA, SVM, ROC, and Confusion Matrix
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage.io import imread
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import pickle
```

Figure 28. Importing libraries for SVM with PCA

```
[ ] ## Listing Categories
Categories=['freshapples','rottenapples','freshoranges','rottenoranges','freshbanana','rottenbanana']

[ ] ## Creating Empty Data Array
flat_data_arr=[]

## Creating Target Array for the LAbsel
target_arr=[]

## Data Director (this time using Google Collab )
datadir='/content/drive/MyDrive/ECEN 429/train'

## For Loop to Access the files on Google Drive
for i in Categories:
    print(f'loading... category : {i}')
    ## creates path that combines path name to one path
    path=os.path.join(datadir,i)
    ## Image Pre-Processing
    for img in os.listdir(path):
        img_array=imread(os.path.join(path,img)) ##Reads image from file
        img_resized=resize(img_array,(150,150,3)) ##Resizing Image to 150,150,3
        flat_data_arr.append(img_resized.flatten()) ##Flattens the Image into one array
        target_arr.append(Categories.index(i)) ## sets labels to index variable
    print(f'loaded category:{i} successfully')
```

Figure 29. Creating file path and inputting it into empty array

```
## Setting Data to numpy array from flat_data_array
flat_data=np.array(flat_data_arr)

## Setting Data to numpy array from target_arr
target=np.array(target_arr)

## Creates 2D tabular data
df=pd.DataFrame(flat_data)
## sets label value to each image's information
df['target']=target
df
```

Figure 10. Creation of 2D tabular data

```
[ ] ## Sets df to be for x until last column
x=df.iloc[:, :-1]

## Sets target of df for y until last column
y=df.iloc[:, -1]
```

Figure 11. Setting x and y for data and labels respectively for all values

```

▶ ## Importing Libraries for Feature Scalling
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler

## removes the target column from the dataset
x = df.drop('target', axis=1)
y = df.target

## Prints Shape of the data and the labels
print(f"x' shape: {x.shape}")
print(f"y' shape: {y.shape}")

## pipelining is sequencing multiple preprocessing steps together
pipeline = Pipeline([
    ('min_max_scaler', MinMaxScaler()), ##1) for each datapoint, it subtracts min value in feature and divides by range
    ('std_scaler', StandardScaler()) ##2) Standardize features by removing the mean and scaling to unit variance
])

❏ 'x' shape: (2698, 67500)
  'y' shape: (2698,)

[ ]
scaler = StandardScaler()

## Splitting the data for labels and for the 2D array like data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

## Transforming 2D Tabular Data using Standard Scaler
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

Figure 12. Feature scaling and splitting of data into training and testing

```

[ ] ## Importing PCS Library
from sklearn.decomposition import PCA

## Setting PCA to choose 100 mos energetic components
pca = PCA(n_components=100)

scaler = StandardScaler()

## Transforming the PCA - 2D Tabular Data using Standard Scaler
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

Figure 33. Applying PCA for 100 most energetic components and applying feature scaling afterwards

```

[ ] ## Importing the SVM and Grid Search Libraries
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

## Defining type of parameters, and range of parameters, and types of kernel functions
param_grid = {'C': [0.01, 0.1, 0.5, 1, 10, 100], 'gamma': [0.1, 0.01, 0.001], 'kernel': ['rbf']}

## Conducting the GridSearch: it is an exhaustive search that pins each combination to find the best one
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=1, cv=5)
grid.fit(X_train, y_train)

## defining the best parameters obtained from the grid search
best_params = grid.best_params_
print(f"Best params: {best_params}")

## Fitting the Parameter search SVM Model on training data
svm_clf = SVC(**best_params)
svm_clf.fit(X_train, y_train)

```

Figure 34. Applying GridSearchCV and then utilizing SVM


```

▶ ## importing required dataset for built in confusion matrix.
## Source: scikit learn Confusion Matrix
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay

class_names= ['freshapples','rottenapples','freshoranges','rottenoranges','freshbanana','rottenbanana']

classifier = svm_clf.fit(X_train, y_train)

np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    disp = ConfusionMatrixDisplay.from_estimator(
        classifier,
        X_test,
        y_test,
        display_labels=class_names,
        cmap=plt.cm.Blues,
        normalize=normalize,
    )
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()

```

Figure 35. Confusion matrix construction for SVM with PCA

```

▶ ## importing libraries for ROC and SVM
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
from yellowbrick.classifier import ROCAUC

[ ] ## Defining the Model, with input provided by result given by GridSearchCV output
model = SVC(kernel='rbf', C=100, gamma=0.01)

#using visualizer function from Yellobrick Library
## defining the Classes
visualizer = ROCAUC(model, classes=['freshapples','rottenapples','freshoranges','rottenoranges','freshbanana','rottenbanana'])
## fitting visualizer on training data
visualizer.fit(X_train, y_train) # Fit the training data to the visualizer
## fitting visualizer on testing data
visualizer.score(X_test, y_test) # Evaluate the model on the test data
## Display the figure
visualizer.show()

```

Figure 36. ROC Curve construction for SVM with PCA utilizing Yellow Brick Library

Convolutional Neural Network with Support Vector Machine (CNN + SVM)

```

# Import libraries for tensors, image processing, linear algebra, accessing os directories, plotting, and Neural Networks
import os
import numpy as np
import pandas as pd
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns

# Importing the libraries from keras and tensorflow
import tensorflow as tf
import tensorflow.keras
# Importing layers for convolutional operations for CNN
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import BatchNormalization, Activation

## Importing Libraries for Confusion Matrix and ROC
import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage.io import imread
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import pickle

```

Python

Figure 37. Importing appropriate libraries for CNN+SVM model

```

## File Path for Training and Testing Datasets
train_path = '/Users/Abdulla Kafoud/Documents/SPRING 2022/ECEN 429/Project/dataset/dataset/train'
test_path = '/Users/Abdulla Kafoud/Documents/SPRING 2022/ECEN 429/Project/dataset/dataset/test'

## Using the ImageDataGenerator function to generate batches of tensor image data with real-time data augmentation
## we rescale the image to it being divided by 255, because of the greyscale range
train_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)

##flow_from_directory function used to take path to given directory and generates batches of augmented data
train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size=(50,50),
                                                    batch_size = 20,
                                                    color_mode= "rgb",
                                                    class_mode = "categorical")
test_generator = test_datagen.flow_from_directory(test_path,
                                                  target_size=(50,50),
                                                  batch_size = 20,
                                                  color_mode= "rgb",
                                                  shuffle= False,
                                                  class_mode = "categorical")

```

Python

Figure 38. Creation of directories for classes based on training and testing

```

#CNN + SVM Architecture

# Using Sequential Layering
model = Sequential()

#Using RELU activation to deal with negative numbers (nonlinearity) in the matrix

# Input Layer, input shape specified same as target_size, 3 is for RGB
# Input Layer is conv2d which uses kernel, convolves with layer input, produces tensor output
# Conv2D is used to reduce the dimensionality, here it is 128 by 3
# Relu used as Activation Function
model.add(Conv2D(128, 3, activation="relu", input_shape=(50,50,3))) #filters #Kernelsize #RELU

# MaxPooling Layer: used to downsample input, to condense the data
# Takes Max of input window which is the pool size, standard being 2X2
model.add(MaxPooling2D())

## Additional Convolutional Layers to condense dimensionality
## filter is of size 64 by 3
# Relu used as Activation Function
model.add(Conv2D(64, 3, activation="relu"))

## Additional Conv2D with filter reduced to size 32 by 3
## Enhances reduction of dimensions of tensor input
## Relu used as Activation Function
model.add(Conv2D(32, 3, activation="relu"))

## Another reduction of tensor inputs using MaxPooling2D
model.add(MaxPooling2D())

## Dropout Layer, fixes overfitting of Data
## Reduces amount of unwanted data with fixed constant 0.5
# constant represents fraction of input unit to be dropped
model.add(Dropout(0.50)) #Helps with overfitting

## Use flatten to turn matrices into arrays, making it digestible for an ANN
model.add(Flatten())

## Creates a layer of 5000 neurons with input coming from Flattening Layer
## Relu Function for activation
model.add(Dense(5000, activation = "relu")) #5000 neurons in this layer with RELU activation

```

Figure 39. The architecture and main code for the CNN+SVM model (Part 1)

```

## Creates another smaller layer of 500 neurons to condense data from previous layer,
## Output of Dense Layer layer fed to final layer
## Relu Function for activation
model.add(Dense(500, activation = "relu"))

## MultiClassification SVM Layer

## Final Layer has Dense Layer 6, for 6 classes
## activation function softmax to display multiple probabilities
## kernel regularizer of l2 norm acting as a penalty term and for SVM layer
model.add(Dense(6, kernel_regularizer=tf.keras.regularizers.l2(0.01), activation = 'softmax'))

## Optimizer Adam, loss function squared hinge for SVM
model.compile(optimizer = 'adam', loss = 'squared_hinge', metrics = ['accuracy'])

```

Figure 40. The architecture and main code for the CNN+SVM model (Part 2)

```

## Plotting the evolution of loss and accuracy for training
## and validation data for different epochs
loss_train = hist.history['loss']
loss_val = hist.history['val_loss']
epochs = range(1,401)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

loss_train = hist.history['accuracy']
loss_val = hist.history['val_accuracy']
epochs = range(1,401)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```

Figure 41. Training and Validation Loss/Accuracy Plotting

```

## Saving the CNN and SVM model for it to be loaded in confusion matrix
batch_size=250
model.save("CNN+SVM.h5")
model = keras.models.load_model("CNN+SVM1.h5")

## Confusion Matrix Function
## Obtained from source: SciKit Learn Library

def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):

    ## Plotting Size, Color Map, title
    plt.figure(figsize=(20,20))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    ## Defining labels of confusion matrix
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    ## Normalizing Values for Confusion Matrix
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)] = 0.0
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    thresh = cm.max() / 2.

    ## for loop to iterate through Confusion Matrix
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

Figure 42. Creation of confusion matrix for CNN+SVM model (Part 1)

```

# print(target_names)
target_names = []
for key in train_generator.class_indices:
    target_names.append(key)

#input to be for testing dataset
Y_pred = model.predict_generator(test_generator)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
cm = confusion_matrix(test_generator.classes, y_pred)
plot_confusion_matrix(cm, target_names, title='Confusion Matrix')

#Print Classification Report
print('Classification Report')
print(classification_report(test_generator.classes, y_pred, target_names=target_names))

```

Python

C:\Users\Abdulla Kafoud\AppData\Local\Temp\ipykernel_6152\315908801.py:47: UserWarning: 'Model.predict_generator' is deprecated and will be removed in a future version. Please use 'Model.predict', which supports generators.

Y_pred = model.predict_generator(test_generator)

Confusion Matrix

Normalized confusion matrix

Classification Report

	precision	recall	f1-score	support
freshapples	0.97	1.00	0.99	112
freshbanana	1.00	0.99	1.00	120
freshoranges	1.00	0.98	0.99	120
rottenapples	0.98	0.97	0.97	128
rottenbanana	0.98	1.00	0.99	120
rottenoranges	0.97	0.95	0.96	88
accuracy			0.98	688
macro avg	0.98	0.98	0.98	688
weighted avg	0.98	0.98	0.98	688

Figure 43. Creation of confusion matrix for CNN+SVM with results (Part 2)

```

## Importing Necessary Libraries from sklearn
from sklearn.datasets import make_classification
from sklearn.preprocessing import label_binarize

## Number of Classes
n_classes = 6

## creates cluster of points with N(0,1)
X, y = make_classification(n_samples=80000, n_features=20, n_informative=6, n_redundant=0, n_classes=n_classes,
                           n_clusters_per_class=2)

# Binarize the output, which is Boolean thresholding of given array
y = label_binarize(y, classes=[0, 1, 2, 3, 4, 5])
n_classes = y.shape[1]

## Split labels and data into half
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
print(y[:2])

```

Python

```

## Defining the ANN+SVM Model,
def build_ANN_SVM_model():
    model = Sequential()
    ## Input Layer has 20 inputs
    model.add(Dense(20, input_dim=20, activation='relu'))

    ## Added more layers, with RELU Activation Function
    model.add(Dense(100, activation = 'relu'))
    model.add(Dense(500, activation = "relu"))
    model.add(Dense(100, activation = "relu"))

    ## Output layer with 6 neurons and L2-Norm, softmax for multiple probs
    model.add(Dense(6, kernel_regularizer=tf.keras.regularizers.l2(0.01), activation
                    = 'softmax'))
    # Compile model
    model.compile(optimizer = 'adam', loss = 'squared_hinge', metrics = ['accuracy'])
    return model

keras_model2 = build_ANN_SVM_model()
keras_model2.fit(X_train, y_train, epochs=100, batch_size=100, verbose=1)

```

Python

Figure 44. Creating synthetic data for and compatible model for ROC of CNN+SVM model (Part 1)

```

##predicting the model on the test data
y_score = keras_model2.predict(X_test)

import numpy as np
from scipy import interp
import matplotlib.pyplot as plt
from itertools import cycle
from sklearn.metrics import roc_curve, auc

## Code for Plotting the ROC Curve
## Source: scikitLearn

# Plot linewidth.
lw = 2

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

```

Figure 45. Creating ROC of CNN+SVM (part 2)

```

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(1)
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
         ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
         ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('CNN and SVM ROC Curve')
plt.legend(loc='lower right')
plt.show()

```

Figure 46. Creating plot for ROC of CNN+SVM (part 3)

Contributions

The contributions for the project in its entirety are as follows.

Sayed Kameli contributed by designing, coding, training, and validating the CNN algorithm. He also contributed to the presentation power points and report by implementing all required information on the CNN algorithm per the provided rubric.

Abd-allah Kafoud contributed by designing, coding, training, and validating the SVM with PCA algorithm. He also contributed to the presentation power points and report by implementing all required information on the SVM with PCA algorithm per the provided rubric. He came up with ideas for this project and coordinated the execution.

Yaroub Hussein contributed by designing, coding, training, validating the CNN+SVM algorithm. He also contributed to the presentation power points and report by implementing all required information on the CNN+SVM algorithm per the provided rubric.

Overall, each member was responsible for one of the three tested algorithms in all aspects related to their own algorithm.