Final Project Report

DATASCI 281: Foundations of Computer Vision

Authors: Michael Liston, Michael Tay, Varun Venkatesh

Introduction

Image classification of fruits and vegetables has a wide range of applications in nutrition, cooking, farming, and produce wholesale. Being able to identify the type (fruit or vegetable) and class (which particular fruit or vegetable) is the foundational layer upon which one can build useful techniques related to produce such as quality evaluation, insect infestation, ripeness evaluation, sorting, recipe generation, and a myriad of others. Creating a well-performing baseline fruit and vegetable classifier opens up a world of possibilities for computer vision applications within the produce industry.

Dataset

Our dataset consists of 3861 images of 36 fruits and vegetables sourced from Kaggle (https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition). The images are in JPEG form and originated from a Bing image search. As such, the dimensions of the images vary with the following range of dimensions: (480 ~ 2000) x (360 ~ 1000) pixels with a DPI of 72 ppi. To standardize our input images, we resized our image dimensions down to 512x512 for training & testing. To alleviate computational load further, we resized images to 64x64 when performing PCA on our extracted features.

This dataset contains images of the following food items:

- Fruits - banana, apple, pear, grapes, orange, kiwi, watermelon, pomegranate, pineapple, mango.

- Vegetables - cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soybean, cauliflower, bell pepper, chili pepper, turnip, corn, sweetcorn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant.

Image Examples

Vegetable: Carrot

Fruit: Kiwi

Estimation of Class Numbers

The following estimations are based on the provided folder structure of the dataset:

We will choose 10 Fruits & 10 Vegetable types to detect:

- Fruits (10)
- From Vegetables (26) -> Choose 10 Vegetables
    - Bell Pepper, Cauliflower, Chilli Pepper, Peas, Corn, Spinach, Turnip, Garlic, Ginger, Cabbage

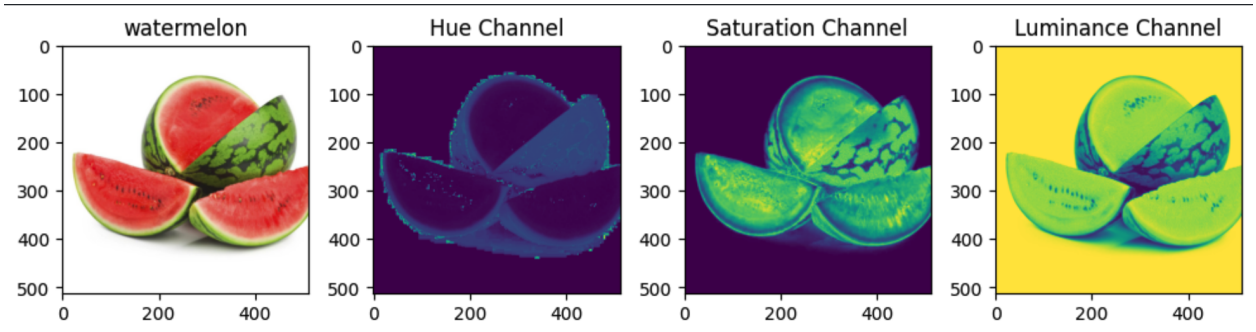
We will have a total of 20 classes

For each of the 20 classes, the distribution of the data is as follows:
Training Images: 100 images for each kind
Test Images: 10 images for each kind
Validation Images: 10 images for each kind

Feature Detection

In order to better capture features relevant and useful to our dataset, we categorized 3 important characteristics that we will be using prior to our classification. We looked at color features through HSV channel output, Laplacian & Sobel for edge output, HOG, and DAISY features.

Color Features



Color features stood out to us as being extremely important for our classification problem, as fruits and vegetables tend to have distinct color properties that help us differentiate between them.

Hue

In image processing and color theory, "hue" refers to one of the three primary attributes of color perception, along with saturation and value (brightness). Hue represents the dominant wavelength of light that gives a color its distinctive appearance on the color wheel. It is essentially what we commonly refer to as the "color" of an object or pixel in an image.

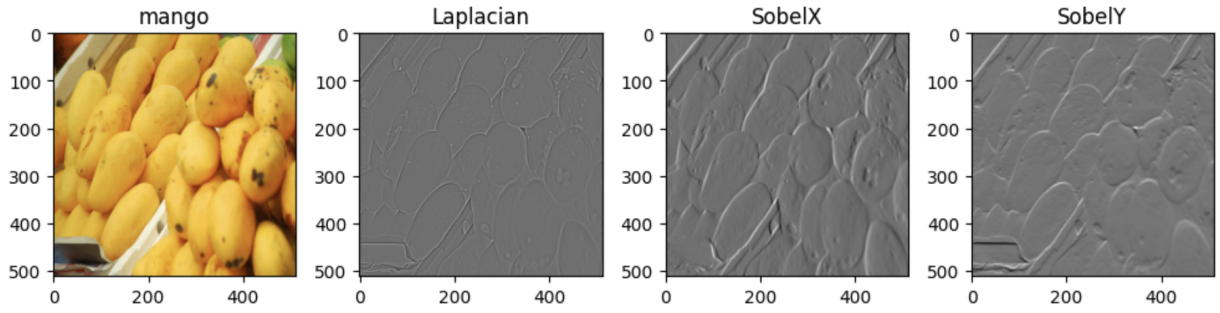
Saturation

In image processing and color theory, "saturation" refers to the intensity or purity of a color. It represents the vividness or strength of a color, ranging from a fully saturated color (pure and vivid) to a desaturated color (more muted or grayish).

Luminance

In image processing and color theory, "luminance" refers to the brightness or intensity of a color, often described in terms of the perceived brightness of a pixel or an area in an image. Luminance is a crucial factor in how we perceive the overall brightness of an image, regardless of its color.

Edge Features



Sobel

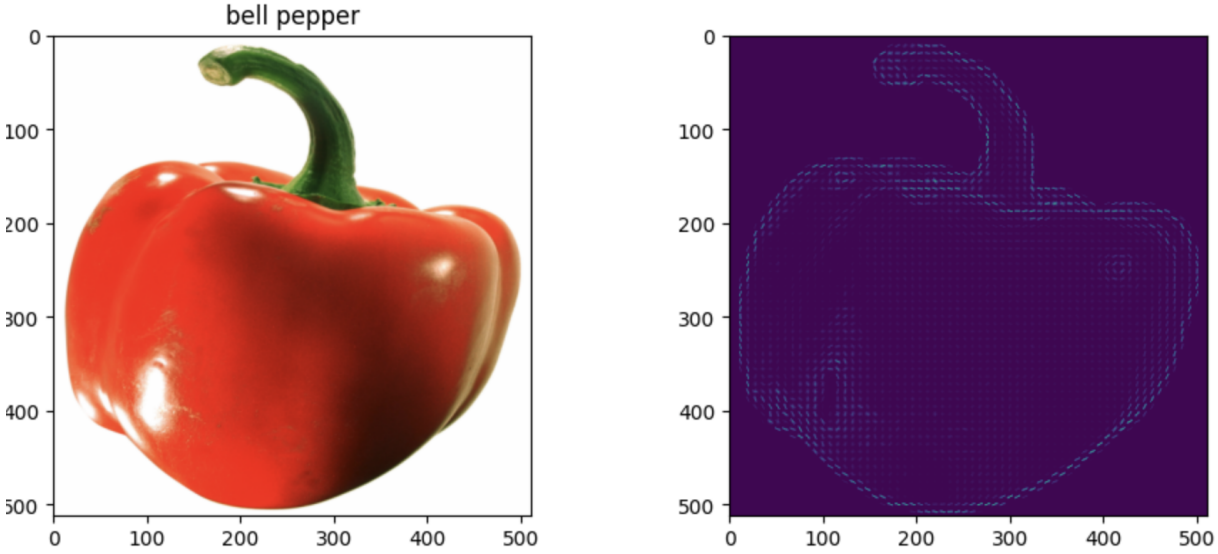
In image processing and computer vision, the Sobel operator is a widely used edge detection filter that helps identify edges and boundaries within an image. It calculates the gradient magnitude and direction of intensity changes in an image, highlighting regions of rapid intensity variation.

Laplacian

In image processing and computer vision, the Laplacian operator is a filter used for detecting areas of rapid intensity changes, such as edges and corners, within an image. It calculates the second derivative of the image's intensity values, highlighting regions where the intensity changes quickly.

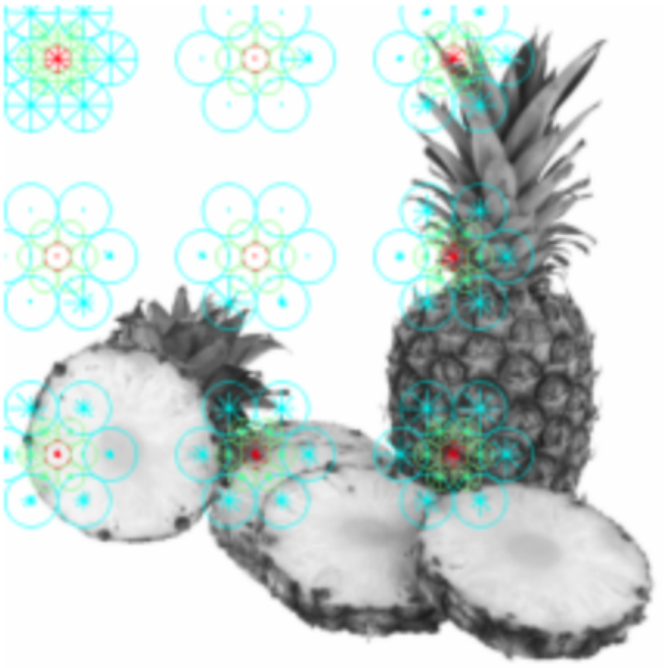
Feature descriptors

Histogram of Gradients (HOG)



The Histogram of Gradients (HOG) is a feature descriptor used in computer vision and image processing to represent the local texture and shape information within an image. It is often employed for object detection and recognition tasks, such as identifying pedestrians, vehicles, or other objects in images. HOG works by analyzing the distribution of gradient orientations within small regions of an image.

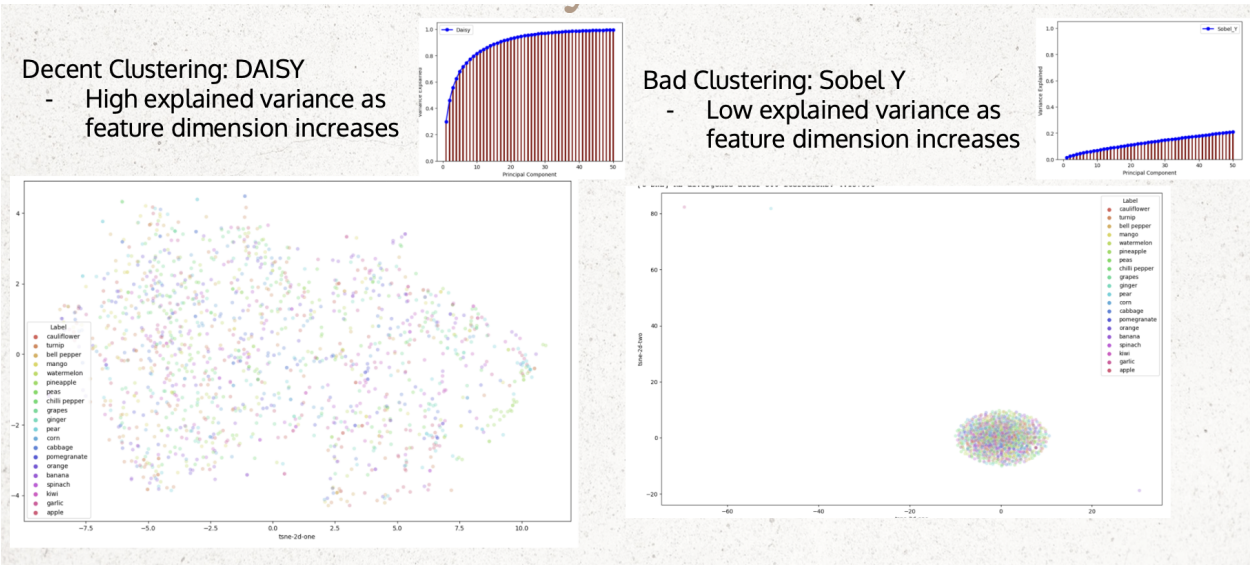
DAISY



The DAISY descriptor is an algorithm that converts regions of interest in an input image into scale invariant descriptors that can be used for keypoint matching and image classification. DAISY features capture the distribution of gradient orientations and magnitudes within circular concentric regions around points of interest in the image. The DAISY descriptor is particularly useful for images with distinctive local structures, as is the case for many of the fruits and vegetables included in our study.

For our classification project, we created DAISY features using the implementation provided in the scikit-image features library. The function takes in parameters such as step size, radius, number of rings, histograms per ring, and orientations. When running the function on our images, we noticed a spike in processing time. To improve runtime performance while minimizing information lost, we lowered the step size from 180 to 150 and the ring radius size from 58 to 40. Decreasing the step size actually increased computational load, but this was offset by the decrease in radius size.

Feature Extraction



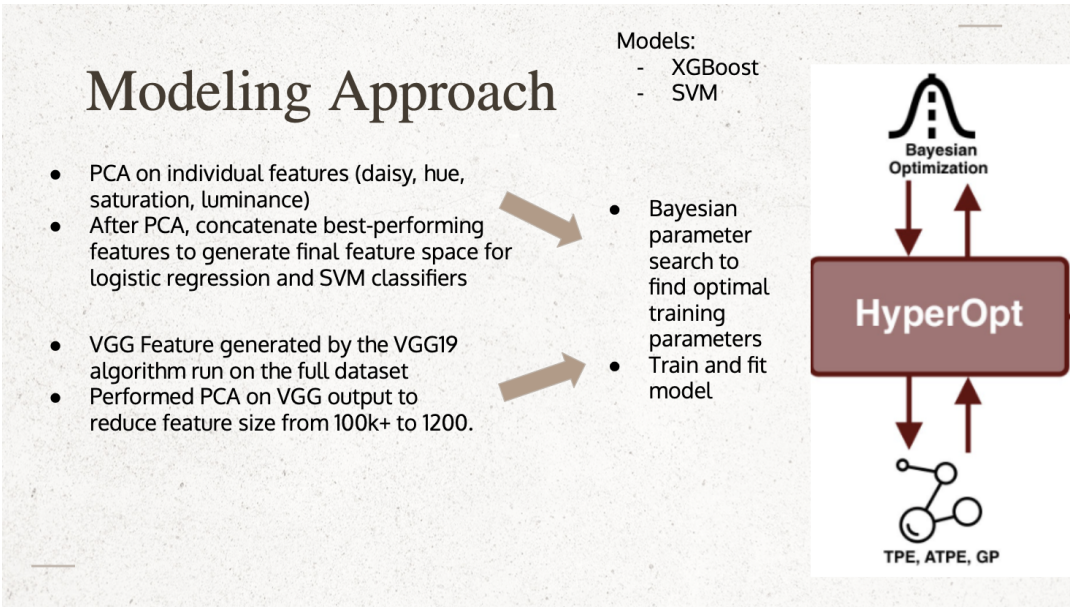
For feature extraction, we started with the full complete set of features and did an initial evaluation on the performance using a simple logistic regression for each feature as a baseline. We also did initial PCA analysis & t-SNE visualization of the extracted features to deem which is impactful in terms of explained variance. Afterwards, we combined each feature vector that we think categorized as important and standardized the values before running PCA on the total vector as a complex feature. We also did PCA on each vector and concatenated them as an alternative variant for testing and comparison.

Based on PCA explained variance plot & t-SNE visualization on the primary two vectors (this is out of 50 features), we can see that the DAISY features perform better than Sobel mainly for two reasons.

1. The primary vector on the X-Y axis is able to separate most of the classes out. Do note we have 50+ feature vectors and we are selecting the top two just for visualization
2. The Sobel explained variance plot shows that as the number of feature vectors grows, the explained variance picks up slowly and linearly compared to the explained variance for DAISY which picks up exponentially before it tapers off after 10 feature vectors.

We also used pretrained VGG feature embeddings and ran PCA on them to create another complex feature for the purpose of training an alternative model.

Classification Models

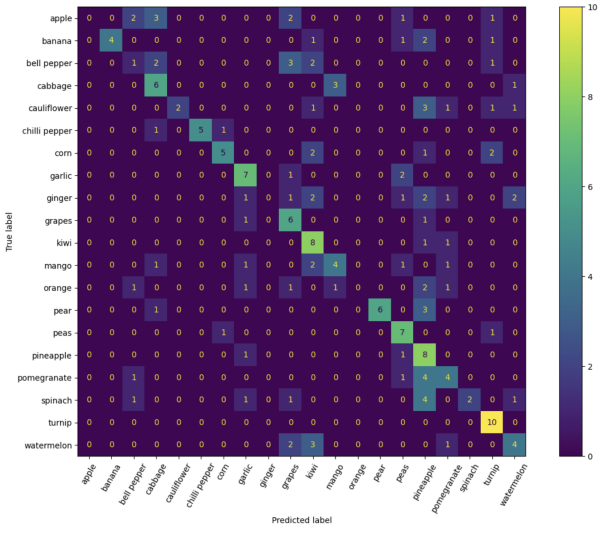


Our classification approach is to run a baseline via Logistic Regression on the extracted features followed by more extensive models like SVM and XGBoost. All models will undergo a Bayesian parameter search by using subsets of the validation data to get the optimal parameters for training and fitting the model.

Logistic Regression

The logistic regression model is trained by adjusting the weights of the input features in a way that minimizes the difference between the predicted probabilities and the actual class labels in the training data. This is usually done through optimization algorithms like gradient descent. The model's parameters (weights) learned during training are used to make predictions on new, unseen data.

Logistic regression is simple, interpretable, and often serves as a baseline algorithm for binary classification tasks.



Run on combined feature set (color features + DAISY) - 200 features

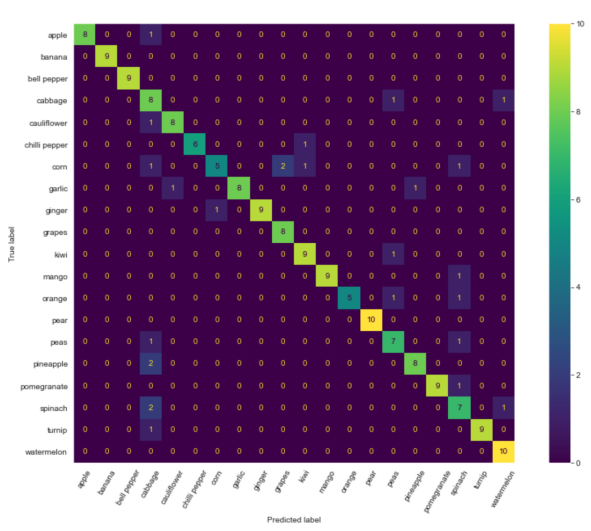
- Binary classifier adapted for multiclass classification

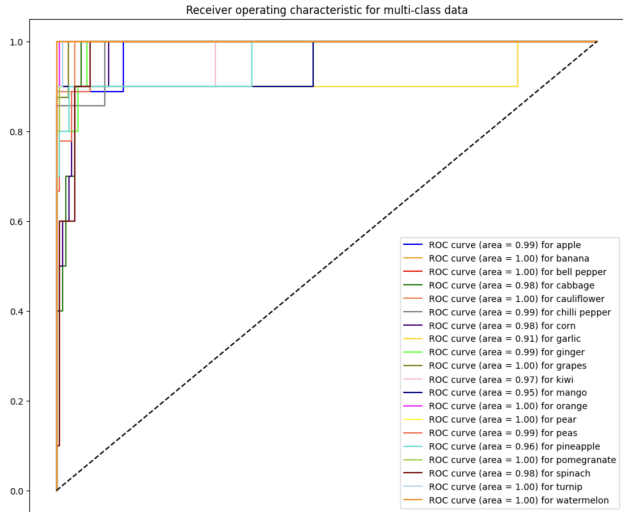
- Accuracy: 47.59%
- Parameters:
    - C: 2.3592
    - Max Iterations: 596
    - Tolerance: 3.148e-05
    - Kernel: lbfgs

SVM

In SVM, the algorithm identifies a subset of the training data points, called support vectors, that are crucial in determining the position and orientation of the optimal hyperplane. These support vectors lie closest to the separating hyperplane and have the greatest influence on its placement.

SVM can handle both linearly separable data (where a straight line can perfectly separate the classes) and non-linearly separable data. For non-linear data, SVM uses a technique called the kernel trick. This involves mapping the original input space into a higher-dimensional feature space where the data points might become linearly separable. Common kernel functions include polynomial kernels, radial basis function (RBF) kernels, and sigmoid kernels.





We ran the SVM model on our combined feature set (color features + DAISY) - 200 features

- Binary classifier adapted for multiclass classification
- Accuracy: 86.1%
- Parameters:
    - C: 2.853
    - Max Iterations: 168
    - Tolerance: 4.26e-5
    - Kernel: rbf

Our model that performed the best on the combined feature set data (color + DAISY features)

was the SVM model. Since SVM was designed for binary classification, we had to adapt the classifier to handle multi-class data. This involved a one-to-one approach where we split the problem into multiple binary classification problems. Boundaries were drawn between each class of data and accuracy results represent the average accuracy for each individual class. This is in contrast to a one-to-all approach that seeks to find the ideal boundary between one class of data vs the rest of the data (ignoring the class clustering of the rest of the data) – we also briefly tried this approach and saw significantly worse results.

The accuracy of the SVM model was very close to 90% with only a handful of misclassifications.

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Given that a ROC curve is also designed for single-class classification, we had to design code that would generate a ROC curve for each class in our model. This involved dummifying the categorical variables, calculating the ROC and area under the ROC curve values, then plotting. As we can see from the plot, the SVM classifier performed very well across all the different classes.

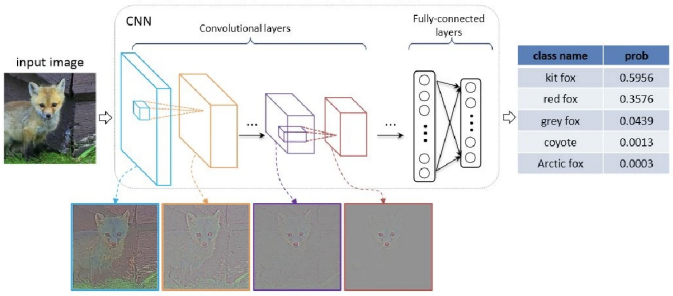
XGBoost on VGG-19

XGBoost (Extreme Gradient Boosting) is a powerful and widely-used machine learning algorithm that belongs to the category of boosting algorithms. It's designed for both regression and classification tasks and is known for its exceptional performance and flexibility in handling a variety of data types and complexities.

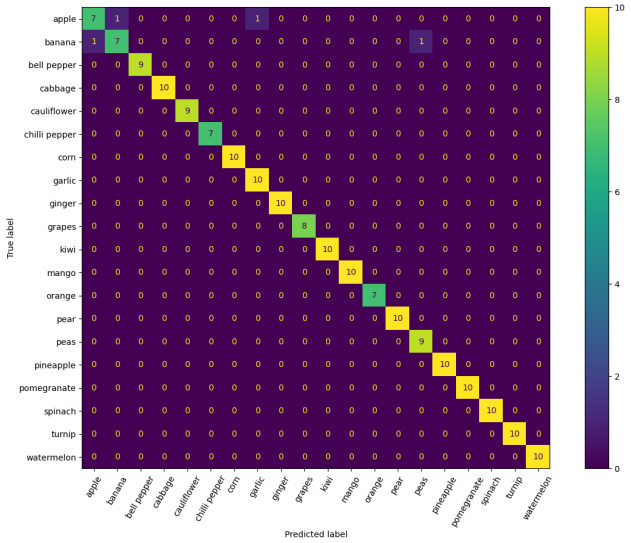
XGBoost builds a predictive model by combining the predictions of multiple individual models, typically decision trees. The algorithm does this through an iterative process that sequentially adds new trees to the model while focusing on the instances that the previous trees misclassified. This process aims to improve the overall predictive performance.

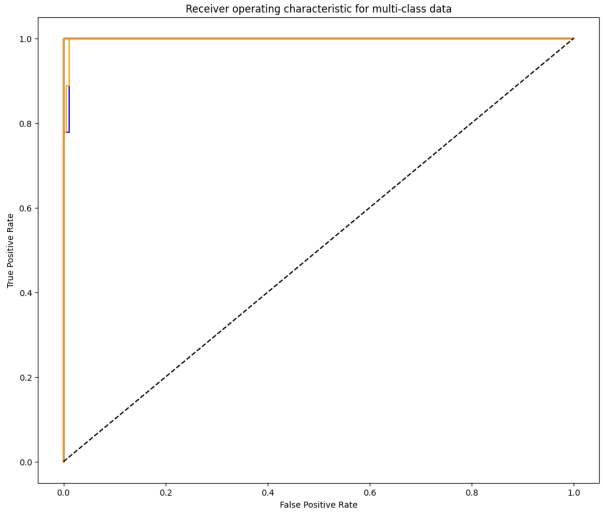
VGG Architecture

VGGNets are based on the most essential features of convolutional neural networks (CNN). The following graphic shows the basic concept of how a CNN works:



VGG (Visual Geometry Group) is a convolutional neural network (CNN) architecture that was developed by researchers at the University of Oxford's Visual Geometry Group. It gained prominence as one of the pioneering architectures in deep learning, particularly for image classification tasks. VGG is known for its simplicity and effectiveness, and it has served as a foundation for many subsequent CNN architectures





Run VGG features (1200 features)

- Accuracy: 97.86%
- Parameters tuned:
  - eta: 0.2104541
  - n_estimators: 68

	precision	recall	f1-score	support
apple	0.88	0.78	0.82	9
banana	0.88	0.78	0.82	9
bell pepper	1.00	1.00	1.00	9
cabbage	1.00	1.00	1.00	10
cauliflower	1.00	1.00	1.00	9
chilli pepper	1.00	1.00	1.00	7
corn	1.00	1.00	1.00	10
garlic	0.91	1.00	0.95	10
ginger	1.00	1.00	1.00	10
grapes	1.00	1.00	1.00	8
kiwi	1.00	1.00	1.00	10
mango	1.00	1.00	1.00	10
orange	1.00	1.00	1.00	7
pear	1.00	1.00	1.00	10
peas	0.90	1.00	0.95	9
pineapple	1.00	1.00	1.00	10
pomegranate	1.00	1.00	1.00	10
spinach	1.00	1.00	1.00	10
turnip	1.00	1.00	1.00	10
watermelon	1.00	1.00	1.00	10
accuracy			0.98	187
macro avg	0.98	0.98	0.98	187
weighted avg	0.98	0.98	0.98	187

Results

- Logistic regression: 47.59%
- Support Vector Classifier
  (on combined daisy, luminance, hue, and saturation): 86.1%
- XGBoost (on VGG features): **97.86%**

Conclusion

Even though running feature detection & extraction to manually generate filters in preparation for training results in an adequate classifier (Logistic, SVM), the embeddings from a conventional deep CNN network (VGG) seem to outperform the more traditional or primitive approaches. This is probably due to the lack of dimensionality or inadequate feature space that manual feature extraction does not provide compared to a well trained network with rich prior data.