



# Deep Learning Techniques for Image-Based Fruit Recognition

---

ITSC 5156 - Applied Machine Learning  
Adam Nur



# ABOUT THE PROJECT

The main goal of this project is to evaluate and compare the results of different neural network architectures for image recognition on the same dataset.



# Fruits 360 dataset

Dataset properties:

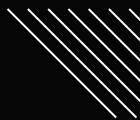
- Total number of images: 90,483
- Number of classes: 131 (fruits and vegetables)
- Image size: 100x100 pixel

# Fruits 360 dataset

The dataset off kaggle contains two directories both are virtually the same except that one of the directories already has all the images sized to 100x100 pixels

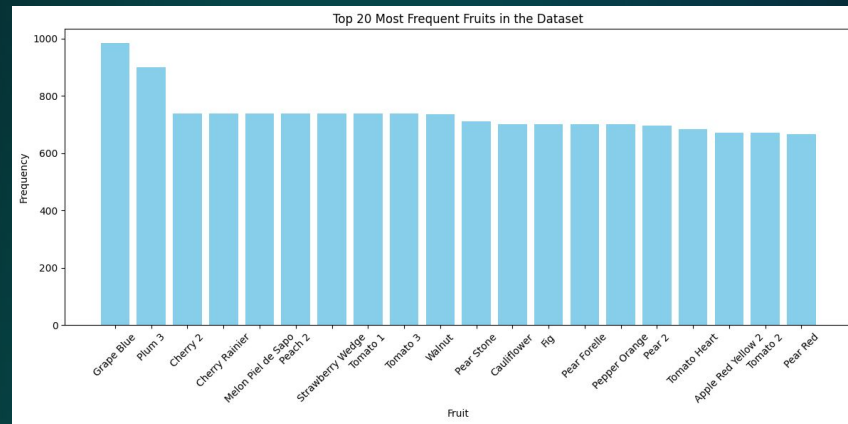
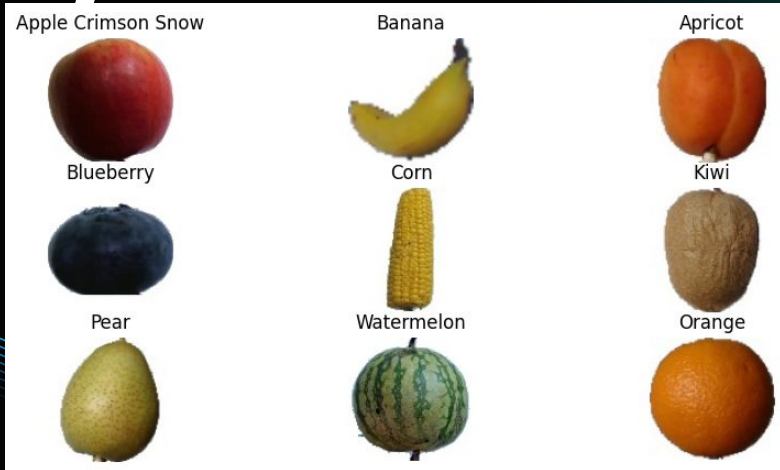
Both directories contain these files:

- **Test** - images use to test the model
- **Training** - images used to train the model
- **Test-multiple\_fruits** - contains images with multiple fruits in them to further test the model
- LICENSE
- readme.md
- Papers



# Dataset Statistics

Here is a brief set of images from the training folder within the dataset.




This is the a visualization of a the top 20 fruits that contains the most images within the dataset.






# Main Motivation

---



The main goal of this project is to test multiple different CNN architectures and hyperparameters and analyze the differences and what makes the model more efficient for image classification



# Main Problems/ Issues faced

The main issue I faced was struggling with was between using google colab or just use VScode I had initially started on VScode but the training times took way longer so I decided to switch to Google Colab in order to use a GPU but the main issue with that is trying to import the dataset to work with it I had used kaggle API token and save it to my google drive but unzipping the file would take so long my browser would go unresponsive. I couldn't figure out how to get passed this so I just switched to VScode and imported it locally. I fixed the issue with the long training times but experimenting with different batch sizes





Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
activation (Activation)	(None, 98, 98, 32)	0
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 32)	9,248
activation_1 (Activation)	(None, 47, 47, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 32)	0
conv2d_2 (Conv2D)	(None, 21, 21, 64)	18,496
activation_2 (Activation)	(None, 21, 21, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 64)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 1024)	6,554,624
activation_3 (Activation)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 131)	134,275
activation_4 (Activation)	(None, 131)	0

Total params: 6,717,539 (25.63 MB)

Trainable params: 6,717,539 (25.63 MB)

Non-trainable params: 0 (0.00 B)

# Model



# Image Data Augmentation and Model Training

## Data Augmentation:

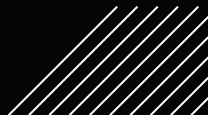
For the Training dataset I have rescaled the images as well as zooming into it a little bit as well as flipping the pictures horizontally and for the Testing dataset I have just rescaled the images by '1/255'

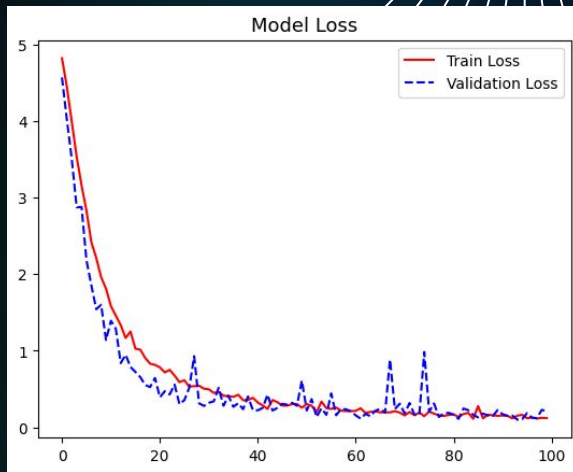
## Data Generators:

I have made 2 data generators for the test set and one for the training set. I have changed the target size: Based on input shape `x.shape[:2]`, I also changed the batch size to 64, made the color mode: RGB and class mode as Categorical. The Training Images: has 67,692 images across 131 classes and the testing Images has 22,688 images across 131 classes.

## Training Params:

- Epochs: 100
- Batch Size: 64
- Training Steps per Epoch: 25
- Validation Steps per Epoch: 12

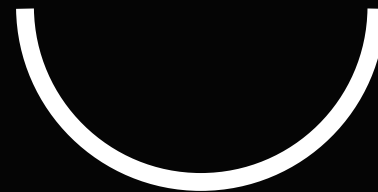
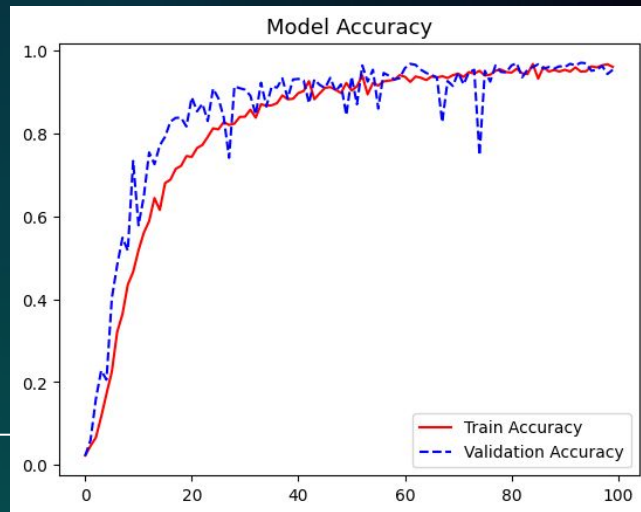




Test Loss: 0.25146737694740295  
Test Accuracy: 0.9244791865348816



# Model Evaluation



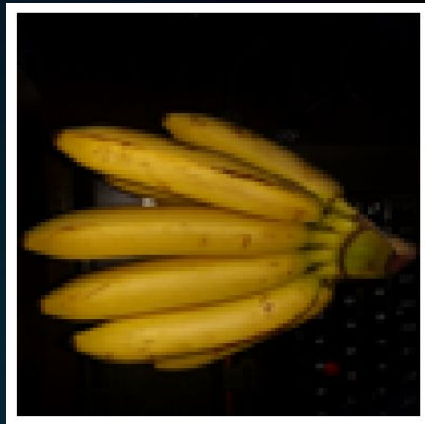


I have test 3 different images one from the test dataset and the other 2 from the test multiple images dataset. The model correctly predicted the single fruit (Mango) and set of multiple fruits of the same kind (Banana Lady Finger). However it struggled with predicting images with multiple different kinds of fruit (like the grape, pear, mandarin fruit bowl).



1/1 — 0s 32ms/step  
Predicted class index: [64]

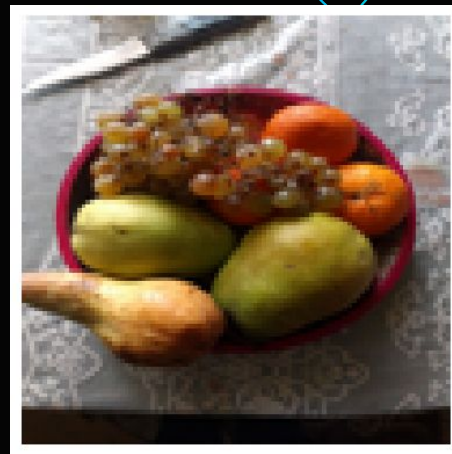
Predicted class label: ['Mango']



1/1 — 0s 32ms/step  
Predicted class index: [17]

Predicted class label: ['Banana Lady Finger']

# Testing



1/1 — 0s 34ms/step  
Predicted class index: [116]

Predicted class label: ['Strawberry']



# Alternate Model 1: Batch Normalization Model

This model has a batch normalization layer after each convolutional layer before max pooling. Batch normalization helps in normalizing the input layer by adjusting and scaling the activations. Overall, this model is more likely to train faster and more effectively because of normalization.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 98, 98, 32)	896
activation_6 (Activation)	(None, 98, 98, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 98, 98, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_5 (Conv2D)	(None, 47, 47, 64)	18,496
activation_7 (Activation)	(None, 47, 47, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 47, 47, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_6 (Conv2D)	(None, 21, 21, 128)	73,856
activation_8 (Activation)	(None, 21, 21, 128)	0
batch_normalization_3 (BatchNormalization)	(None, 21, 21, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten_1 (Flatten)	(None, 12800)	0
dense_2 (Dense)	(None, 1024)	13,108,224
dropout_1 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 131)	134,275

Total params: 13,336,643 (50.88 MB)

Trainable params: 13,336,195 (50.87 MB)

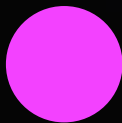
Non-trainable params: 448 (1.75 KB)



# Alternate Model 2:

## Learning Rate and Dropout Model

This model has a increased dropout rates after each max pooling operation and it also has a tuned learning rate for the Adam optimizer. Dropout helps prevent overfitting because it randomly sets the outgoing edges of the hidden units to zero when training This might help get a better generalization on unseen data.



Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 98, 98, 32)	896
activation_9 (Activation)	(None, 98, 98, 32)	0
max_pooling2d_6 (MaxPooling2D)	(None, 49, 49, 32)	0
dropout_2 (Dropout)	(None, 49, 49, 32)	0
conv2d_8 (Conv2D)	(None, 47, 47, 64)	18,496
activation_10 (Activation)	(None, 47, 47, 64)	0
max_pooling2d_7 (MaxPooling2D)	(None, 23, 23, 64)	0
dropout_3 (Dropout)	(None, 23, 23, 64)	0
conv2d_9 (Conv2D)	(None, 21, 21, 128)	73,856
activation_11 (Activation)	(None, 21, 21, 128)	0
max_pooling2d_8 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten_2 (Flatten)	(None, 12800)	0
dense_4 (Dense)	(None, 1024)	13,108,224
dropout_4 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 131)	134,275

Total params: 13,335,747 (50.87 MB)

Trainable params: 13,335,747 (50.87 MB)

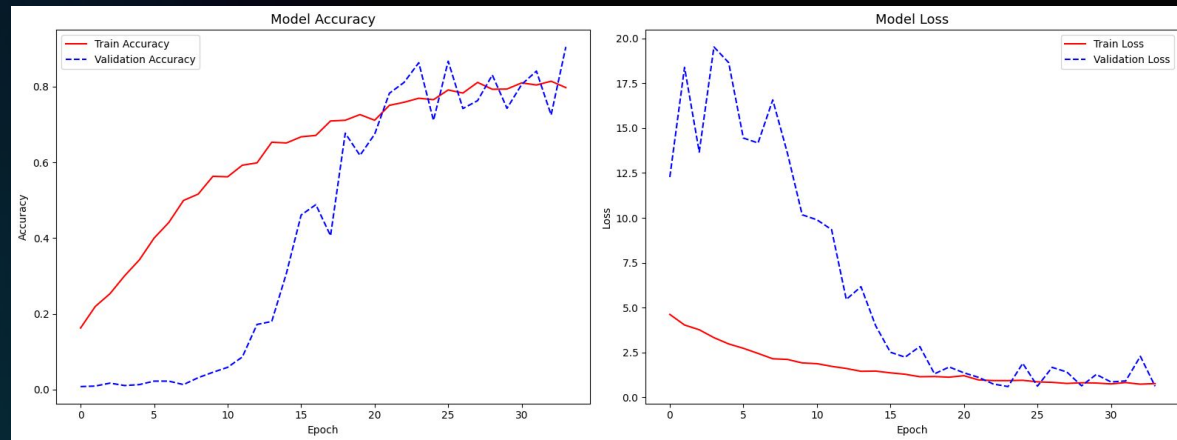
Non-trainable params: 0 (0.00 B)



# Model 1 Evaluation

The Batch Normalization Model might have some overfitting because there's a big gap between the Validation Accuracy (90.49%) and its Training Accuracy (79.75) and the Validation Loss is significantly higher than the second Model

```
Final Training Accuracy: 0.7975
Final Validation Accuracy: 0.9049
Final Training Loss: 0.7583
Final Validation Loss: 0.6092
12/12 ————— 5s 479ms/step - accuracy: 0.8621 - loss: 0.6208
Batch Normalization Model - Test Loss: 0.6170501112937927
Batch Normalization Model - Test Accuracy: 0.8567708134651184
```

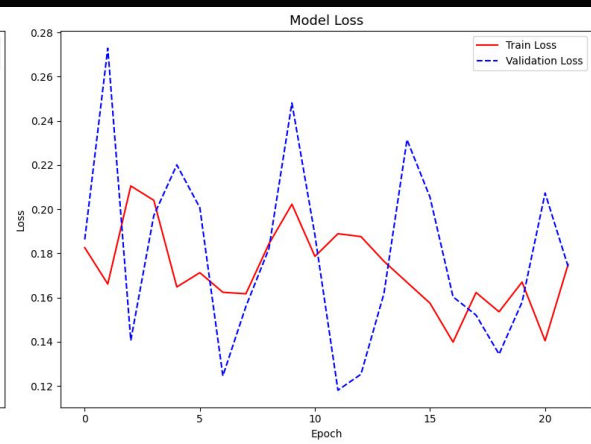
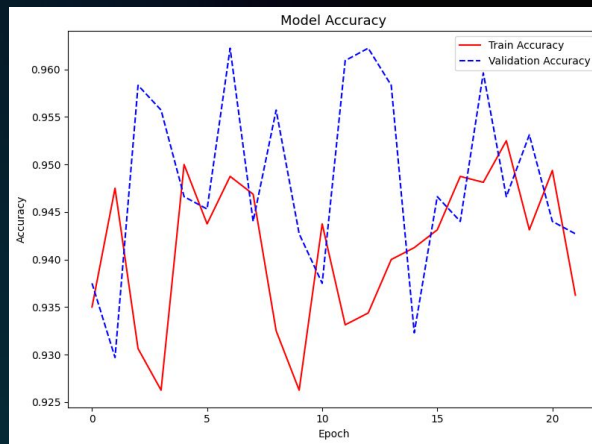


# Model 2 Evaluation

The Learning Rate and Dropout Model has a final training accuracy (93.62%) and validation accuracy (94.27%). Both training and validation loss is also incredibly low which means the model has learned is generalizing well

```
Final Training Accuracy: 0.9362  
Final Validation Accuracy: 0.9427  
Final Training Loss: 0.1750  
Final Validation Loss: 0.1740
```

```
12/12 ————— 4s 390ms/step - accuracy: 0.9567 - loss: 0.1725  
Learning Rate and Dropout Model - Test Loss: 0.1714550256729126  
Learning Rate and Dropout Model - Test Accuracy: 0.9583333134651184
```





Because the second Model had significantly better results I had tried to use that one for testing on a couple of images I preprocessed for this model here are the results the model still isn't perfect as it sometimes mistakes foods of similar color, shape, and size (unripe mangos and avocados)

# Testing

Predicted class label: Apple Red Yellow 2



1/1 — 0s 38ms/step  
Predicted class index: 12



1/1 — 0s 24ms/step  
Predicted class index: 19

Predicted class label: Beetroot



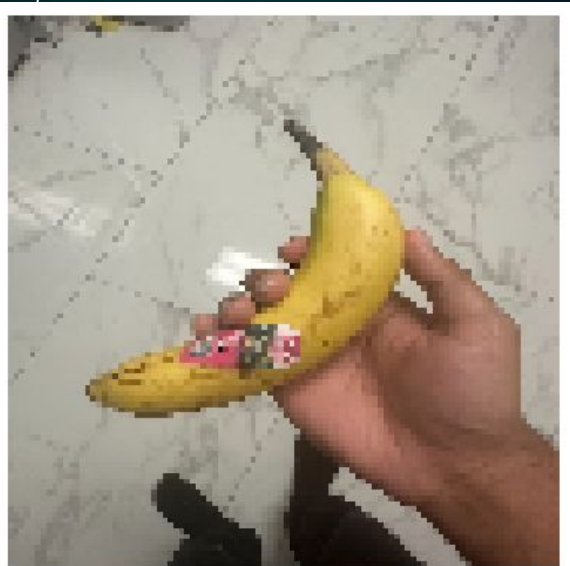
1/1 — 0s 33ms/step  
Predicted class index: 14

Predicted class label: Avocado



Here are the tests of pics that I took at my house for fun. For some reason it predicted it as a Pear Abate

# Testing pt .2



1/1 — 0s 32ms/step  
Predicted class index: 85



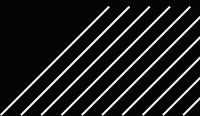
Predicted class label: Pear Abate



# Conclusion and future work

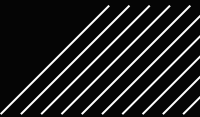
The Learning Rate and Dropout Model performed well, with high accuracy (95.83%) and low loss (0.1715) on the test set, which probably means that its at generalizing from unseen data from training. It outperformed the Batch Normalization Model in both training consistency and test evaluation because it handled overfitting through tuned hyperparameters and dropout.

As this project is not finished yet there are still many different things i want to implement into it like more advanced data augmentation techniques. I also want to implement hyper parameter tuning as well as one more architecture like ResNet or Inception networks



# Video Presentation

[https://drive.google.com/file/d/1fvc9UJL-2Sc3bJ\\_oliVIgKFQUscZdNr4/view?usp=sharing](https://drive.google.com/file/d/1fvc9UJL-2Sc3bJ_oliVIgKFQUscZdNr4/view?usp=sharing)



# Resources

<https://www.kaggle.com/datasets/moltean/fruits/data>

<https://medium.com/hackerdawn/fruit-image-classification-using-cnn-on-google-colab-4fe7274418a5>

**CREDITS:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon** and infographics & images by **Freepik**

Please keep this slide for attribution

