CSE485 Deep Learning

Project Documentation

Submitted to:

Prof. Dr. Alaa Hamdy

Submitted by:

Anthony Amgad Fayek 19P9880

Fruit Recognition

Table of Contents

1. Introduction

This project report presents a deep learning fruit recognition model that can classify images of different types of fruits with high accuracy and speed. The model is based on a convolutional neural network (CNN) architecture that learns features from the raw pixel values of the images. The model is trained and tested on 2 large datasets of fruit images collected from various sources. The report describes the motivation, design, implementation, evaluation, and possible future additions to the project.

The project went through different phases and versions, which are going to be explained and compared throughout the report.

This project can have various possible uses in different domains and scenarios. For example, a fruit recognition model can be used to:

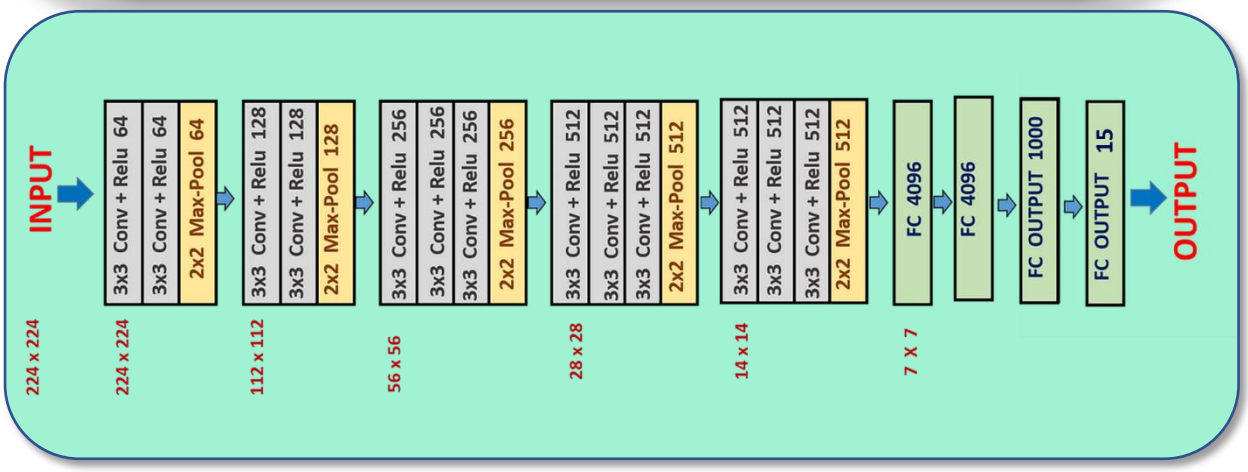
1. Automate the sorting and grading of fruits in farms or factories, reducing manual labor and increasing efficiency and accuracy.
2. Provide nutritional information and health benefits of fruits to consumers by scanning them with their smartphones or smart glasses, encouraging healthy eating habits and lifestyles.
3. Create interactive and educational games or apps for children or adults that involve recognizing and learning about different fruits, their origins, cultures, and cuisines.

2. Version 1.0

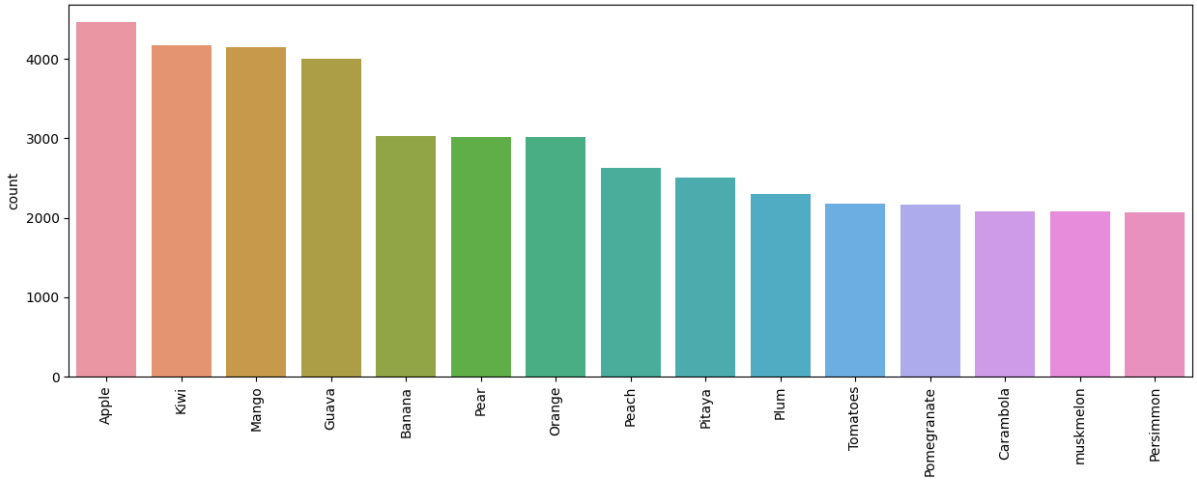
2.1. Implementation

At first a VGG-16 neural network was applied:

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=15, activation="softmax"))
model.summary()
```

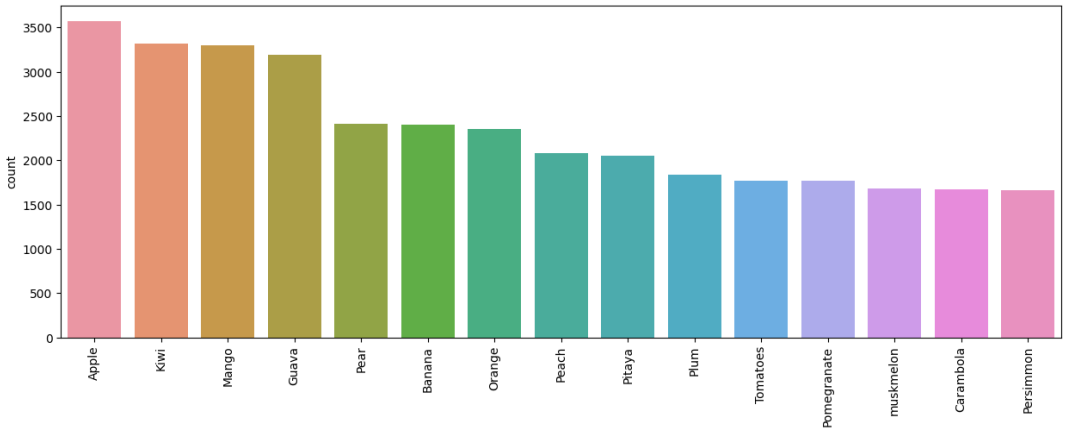


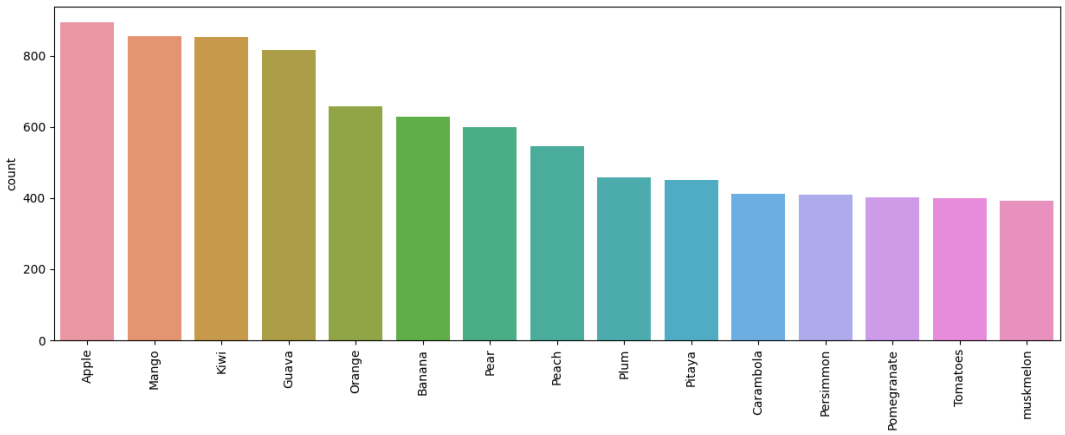
The dataset used had around 40000 images in total.



These numbers were then split into an 80:20 split into training and testing datasets respectively.

The training data was later split into a 90:10 split into training and validation datasets respectively.






```
train_set , test_set = train_test_split(data,test_size=0.2, shuffle=True)
```

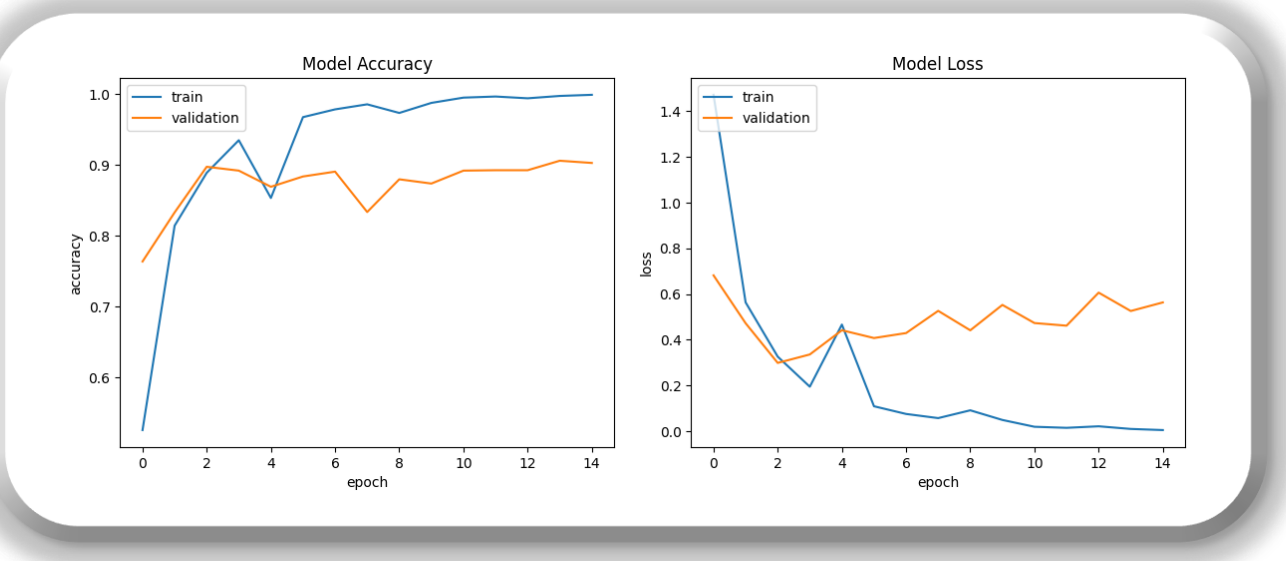
Since the number of images was too large to enter the memory in one batch, a flow from a data-frame was created containing the paths of the images and their labels. This flow would let the images enter one by one to be resized and used for training then exit the ram waiting for the next epoch to re-enter.

```
train_gen = ImageDataGenerator(validation_split=0.1)
test_gen = ImageDataGenerator()

train_data = train_gen.flow_from_dataframe(
    dataframe = train_set,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = True,
    subset = 'training'
)

val_data = train_gen.flow_from_dataframe(
    dataframe = train_set,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = True,
    subset = 'validation'
)

test_data = test_gen.flow_from_dataframe(
    dataframe = test_set,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = False
)
```

The model was then compiled using an SDG optimizer as the Adam optimizer doesn't work well with VGG networks due to it skipping many trainable parameters resulting in bad results.

```
model.compile(
    optimizer = tf.optimizers.SGD(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
history = model.fit(train_data,epochs=15,validation_data=val_data)
```

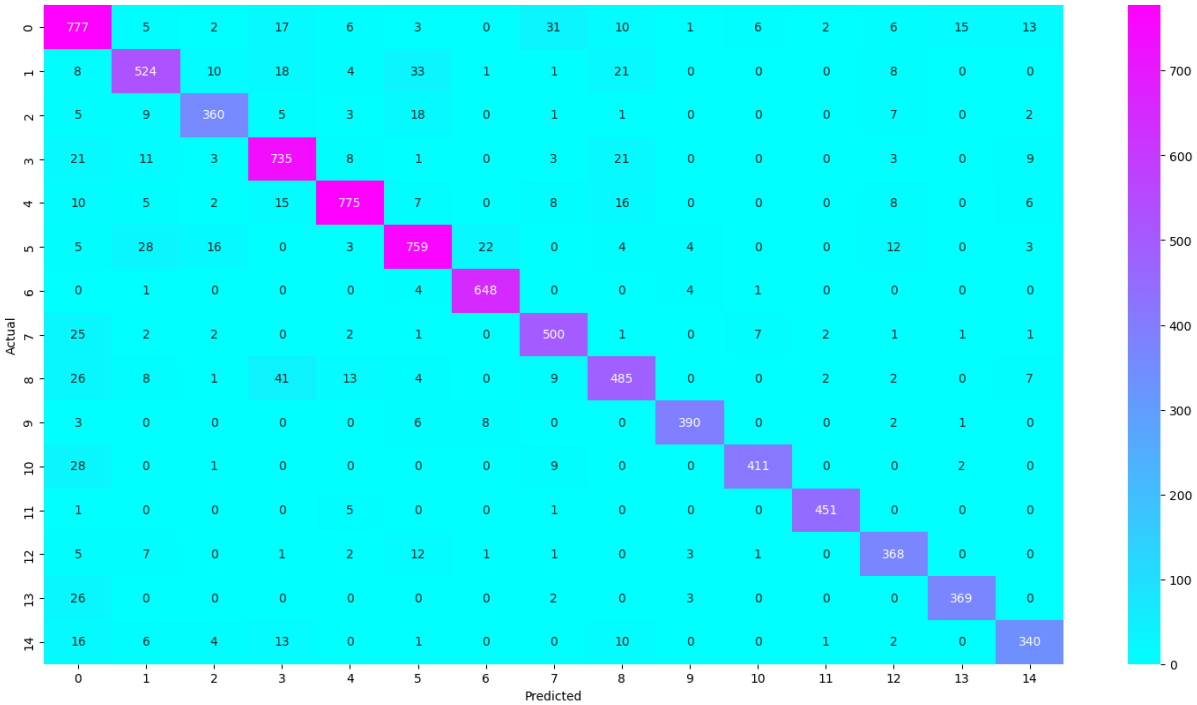


2.2. Validation and Testing Observations

The graphs show that the validation accuracy was at around 90% while the training accuracy was at almost 100% which is a sign of a little bit of overfitting.

```
pred = model.predict(test_data)
pred = np.argmax(pred,axis=1)
print(classification_report(test_data.labels,pred))
```

	precision	recall	f1-score	support
0	0.81	0.87	0.84	894
1	0.86	0.83	0.85	628
2	0.90	0.88	0.89	411
3	0.87	0.90	0.89	815
4	0.94	0.91	0.93	852
5	0.89	0.89	0.89	856
6	0.95	0.98	0.97	658
7	0.88	0.92	0.90	545
8	0.85	0.81	0.83	598
9	0.96	0.95	0.96	410
10	0.96	0.91	0.94	451
11	0.98	0.98	0.98	458
12	0.88	0.92	0.90	401
13	0.95	0.92	0.94	400
14	0.89	0.87	0.88	393
accuracy			0.90	8770
macro avg	0.91	0.90	0.90	8770
weighted avg	0.90	0.90	0.90	8770



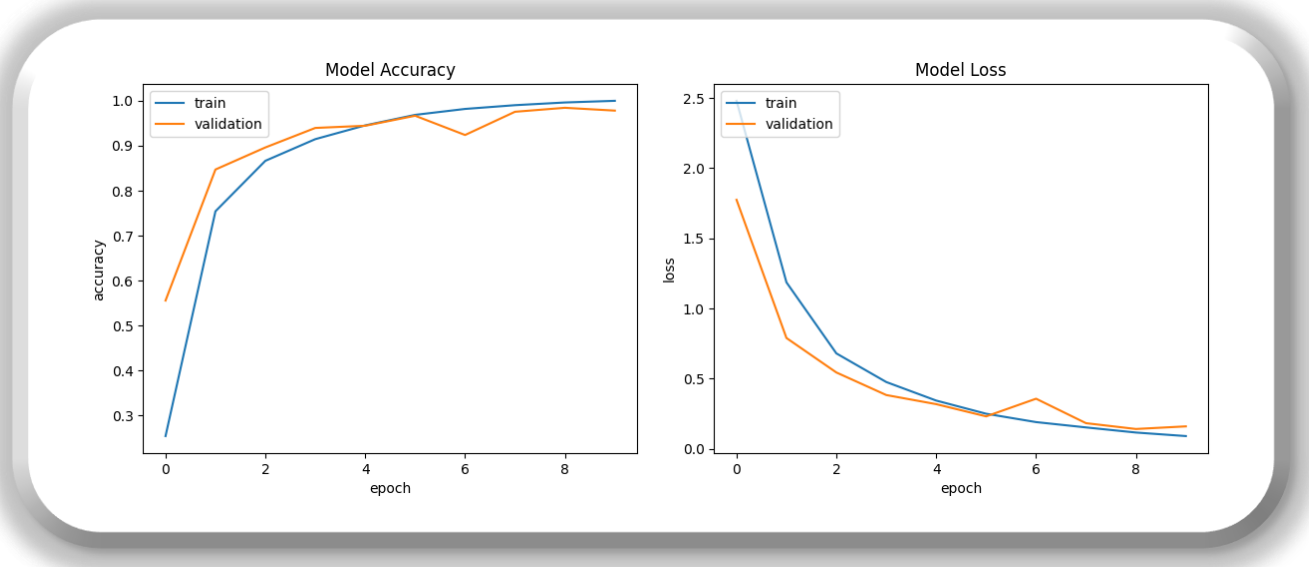
3. Version 1.1

3.1. Implementation

Model 1.1 was the same as 1.0 just with an added layer of regularization to overcome the over fitting.

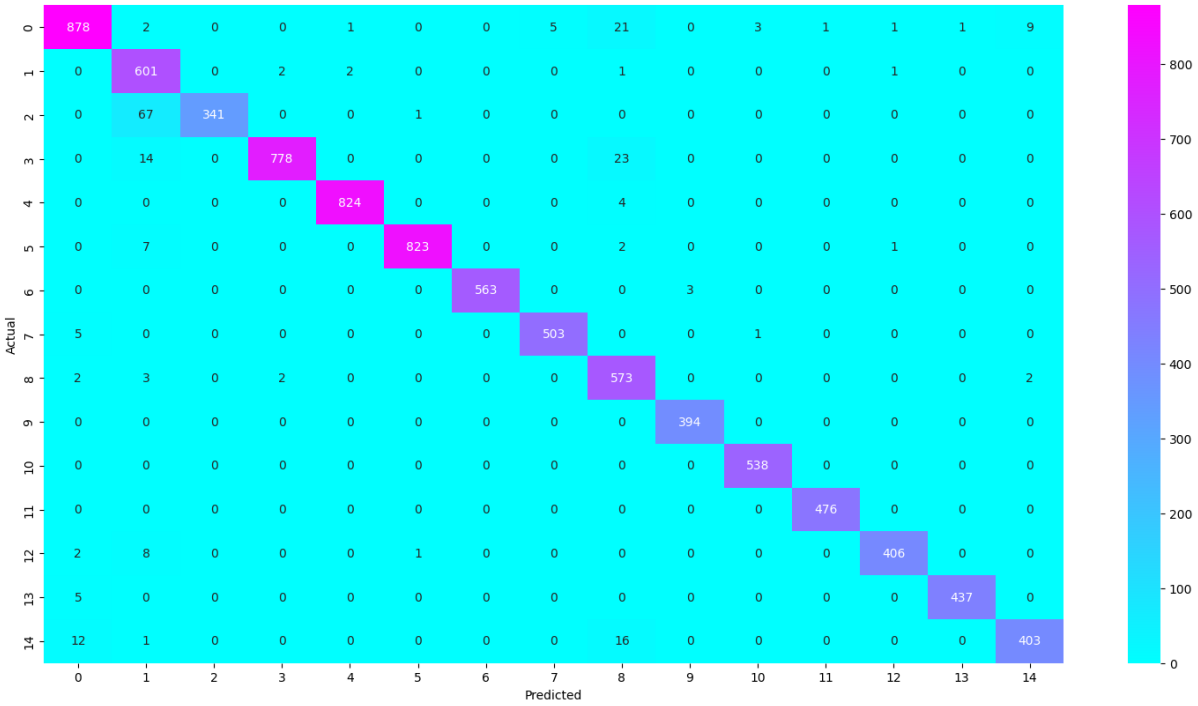
```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096,activation="relu", activity_regularizer=regularizers.L2(0.01)))
model.add(Dense(units=1000,activation="relu"))
model.add(Dense(units=15, activation="softmax"))
model.summary()
```

3.2. Validation and Testing Observations



This shows better results with no slight sense of overfitting due to the addition.

	precision	recall	f1-score	support
0	0.97	0.95	0.96	922
1	0.85	0.99	0.92	607
2	1.00	0.83	0.91	409
3	0.99	0.95	0.97	815
4	1.00	1.00	1.00	828
5	1.00	0.99	0.99	833
6	1.00	0.99	1.00	566
7	0.99	0.99	0.99	509
8	0.90	0.98	0.94	582
9	0.99	1.00	1.00	394
10	0.99	1.00	1.00	538
11	1.00	1.00	1.00	476
12	0.99	0.97	0.98	417
13	1.00	0.99	0.99	442
14	0.97	0.93	0.95	432
accuracy			0.97	8770
macro avg	0.98	0.97	0.97	8770
weighted avg	0.98	0.97	0.97	8770

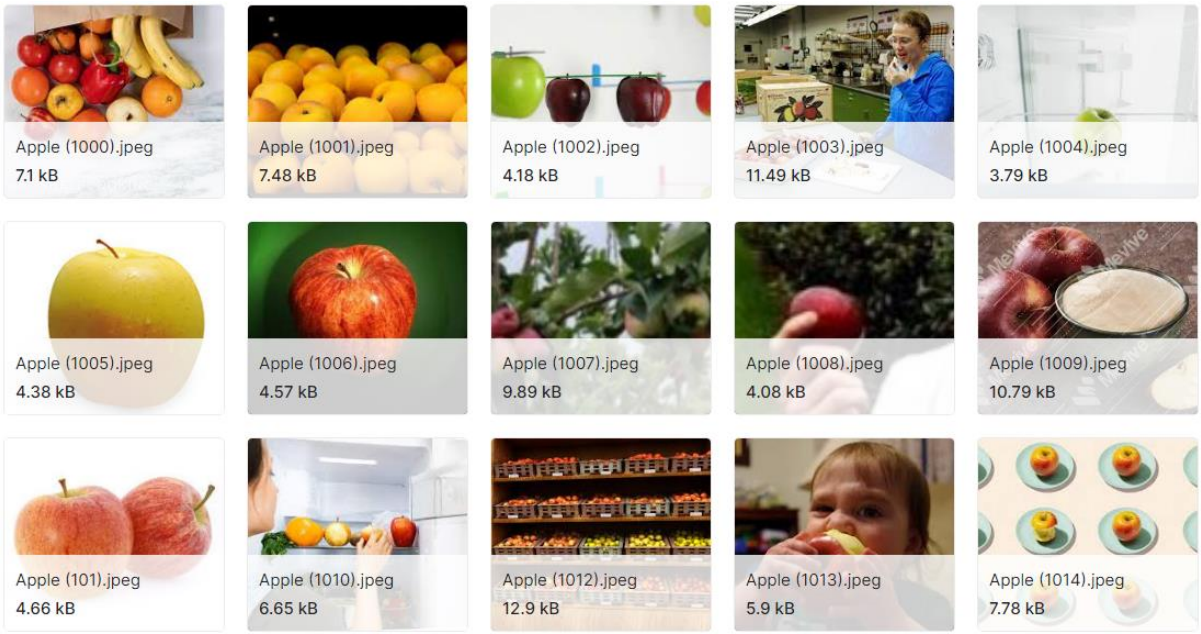


However, these results were slightly deceiving as the dataset wouldn't work on any universal input. The dataset consisted of images that were all made up of the same layout and orientation with the fruit being laid out on a silver tray with it being perfectly centered with nothing even around it.

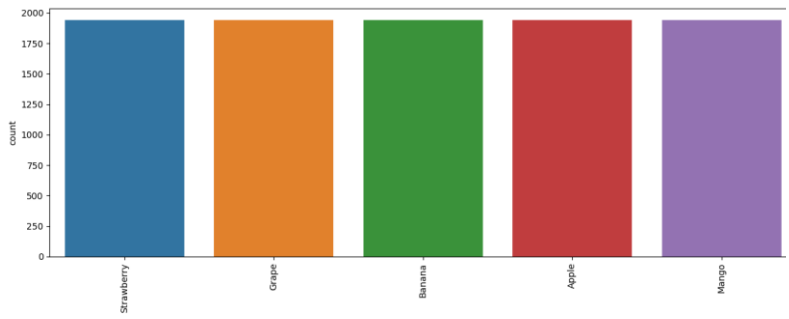
4. Version 2.0

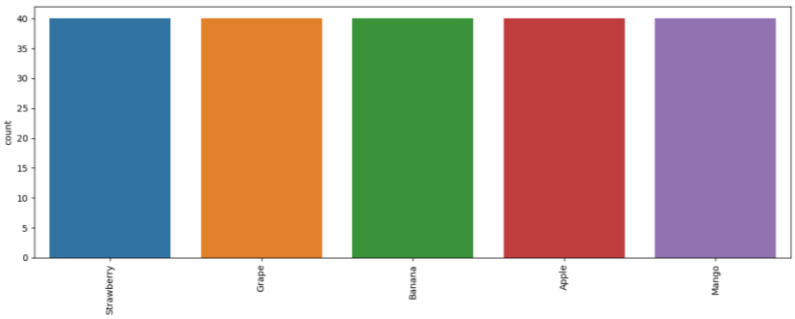
4.1. Implementation

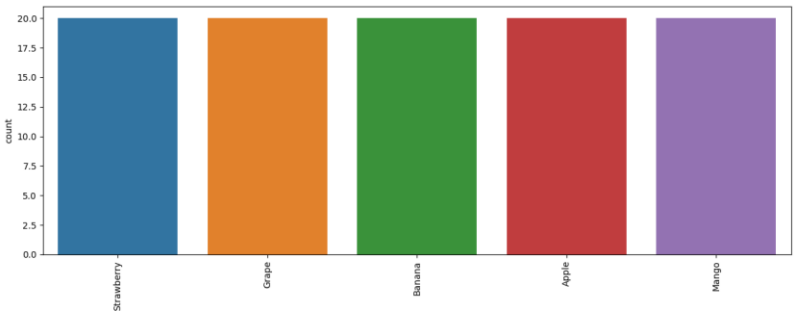
The same model with the regularization was used in Model 2.0. However, the data set was changed entirely by a new dataset that looked more universal and applicable.



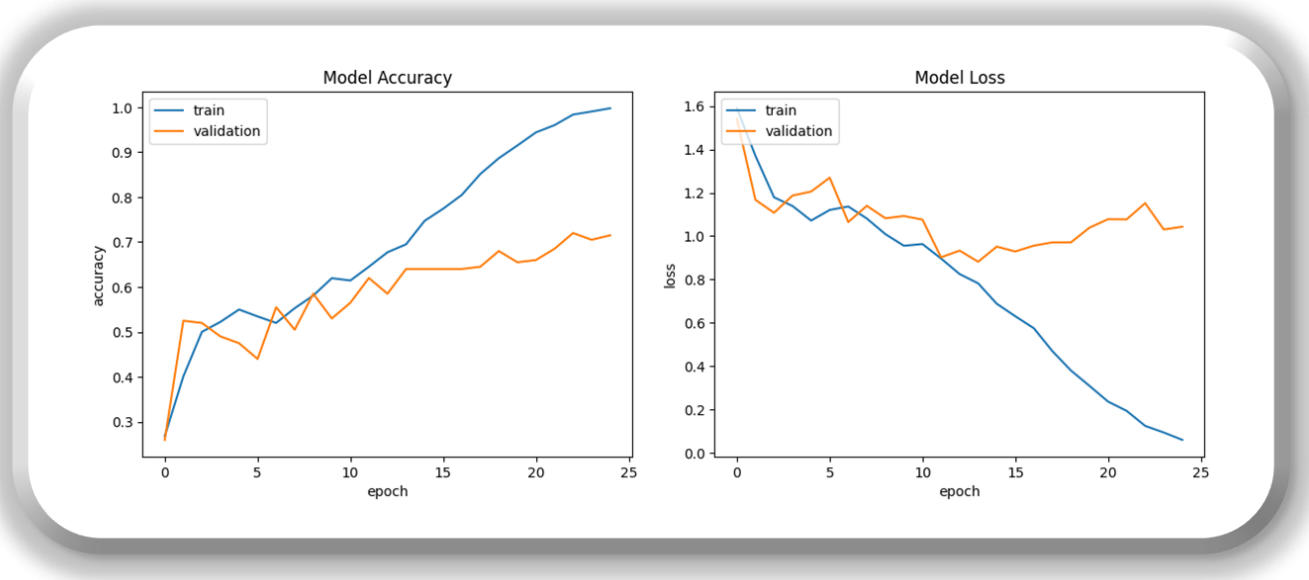
On the other hand, there was a major downgrade with the dataset as it only contained 10000 photos in total which were then divided into 9700 Training, 200 Validation and 100 Testing. The images were also equally distributed among 5 classes which was also another downgrade as now the model can only classify into only 5 fruits instead of 15.





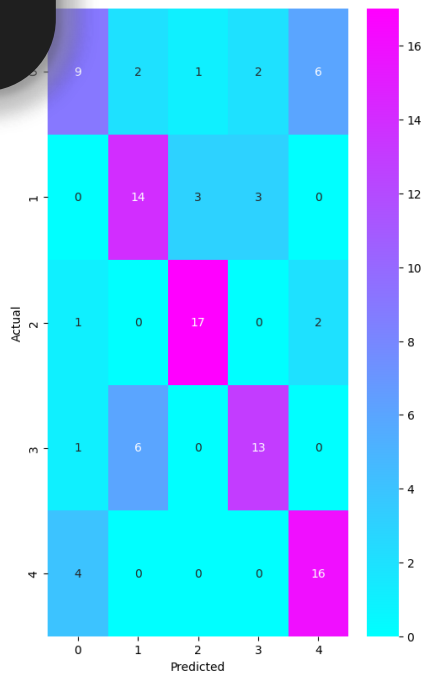


4.2. Validation and Testing Observations



As we can see the validation accuracy this time got a major drop due to the results being more realistic to a true universal model as it only reached 70% accuracy.

	precision	recall	f1-score	support
0	0.60	0.45	0.51	20
1	0.64	0.70	0.67	20
2	0.81	0.85	0.83	20
3	0.72	0.65	0.68	20
4	0.67	0.80	0.73	20
accuracy			0.69	100
macro avg	0.69	0.69	0.68	100
weighted avg	0.69	0.69	0.68	100



5. Version 2.1

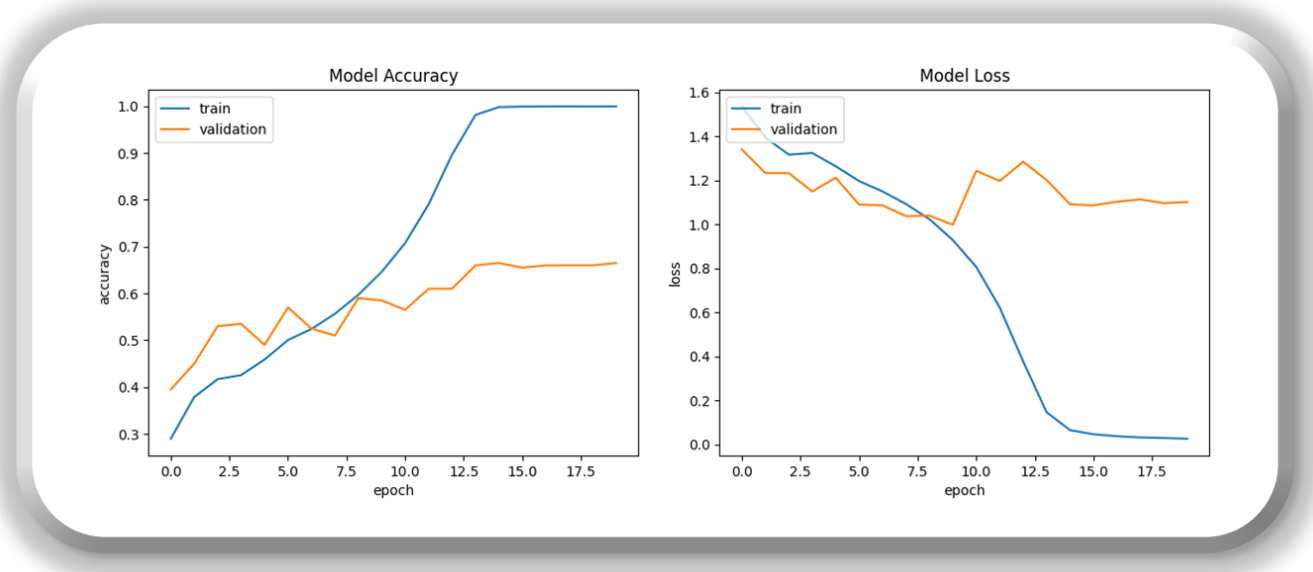
5.1. Implementation

What was decided after testing is that the system relies heavily on colors, so an additional layer of augmentation was added doubling the dataset but in grayscale.

```python
def to_grayscale_then_rgb(image):
    image = tf.image.rgb_to_grayscale(image)
    image = tf.image.grayscale_to_rgb(image)
    return image
```

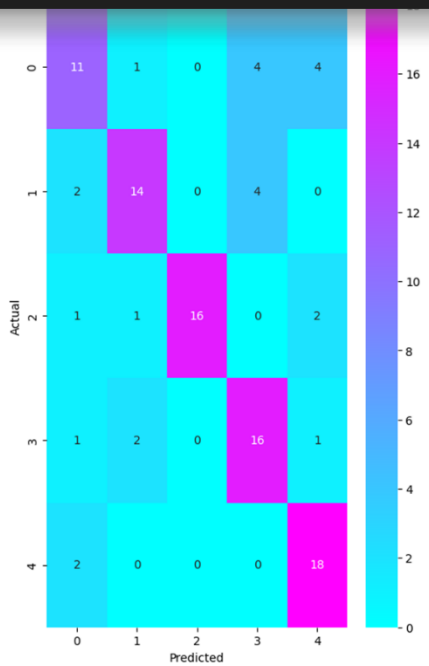
```python
train_gen = ImageDataGenerator()
val_gen = ImageDataGenerator()
test_gen = ImageDataGenerator()
aug_gen = ImageDataGenerator(preprocessing_function=to_grayscale_then_rgb)


train_data = train_gen.flow_from_dataframe(
    dataframe = trainData,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = True
)

aug_data = aug_gen.flow_from_dataframe(
    dataframe = trainData,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = True
)
```

5.2. Validation and Testing Observations



As we can see little to no change happened as it is still around the 70% mark.

	precision	recall	f1-score	support
0	0.65	0.55	0.59	20
1	0.78	0.70	0.74	20
2	1.00	0.80	0.89	20
3	0.67	0.80	0.73	20
4	0.72	0.90	0.80	20
accuracy			0.75	100
macro avg	0.76	0.75	0.75	100
weighted avg	0.76	0.75	0.75	100

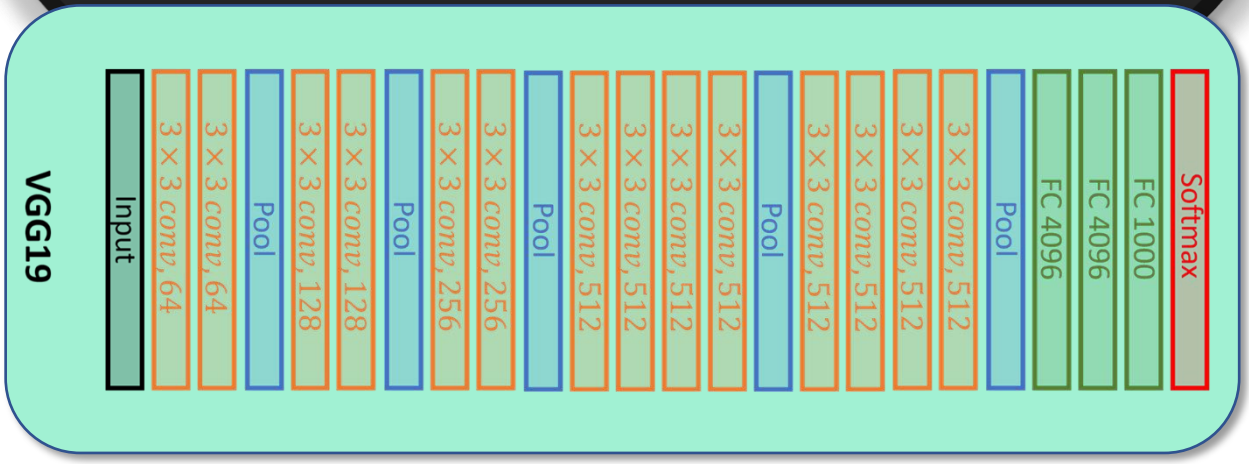


6. Version 2.2

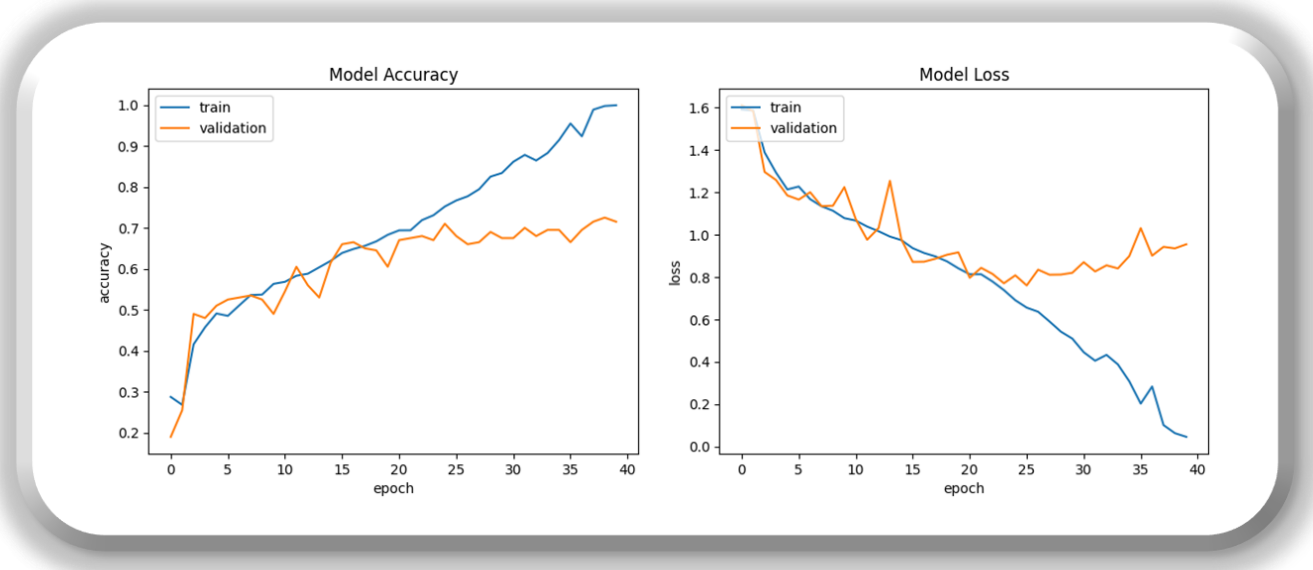
6.1. Implementation

The VGG-16 model was entirely replaced with VGG-19 in hopes of achieving better performance because of the added decision planes.

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(units=4096,activation="relu",activity_regularizer=regularizers.L2(0.01)))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=1000, activation="relu"))
model.add(Dense(units=5, activation="softmax"))
```

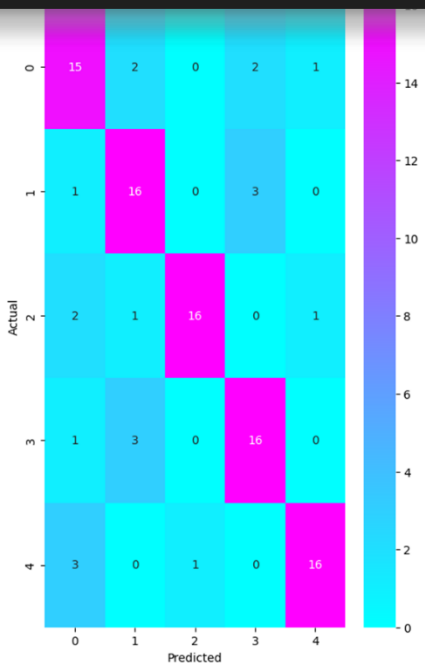


6.2. Validation and Testing Observations



Little change occurred as testing results are now around the 80% mark.

	precision	recall	f1-score	support
0	0.68	0.75	0.71	20
1	0.73	0.80	0.76	20
2	0.94	0.80	0.86	20
3	0.76	0.80	0.78	20
4	0.89	0.80	0.84	20
accuracy			0.79	100
macro avg	0.80	0.79	0.79	100
weighted avg	0.80	0.79	0.79	100



7. Version 2.3

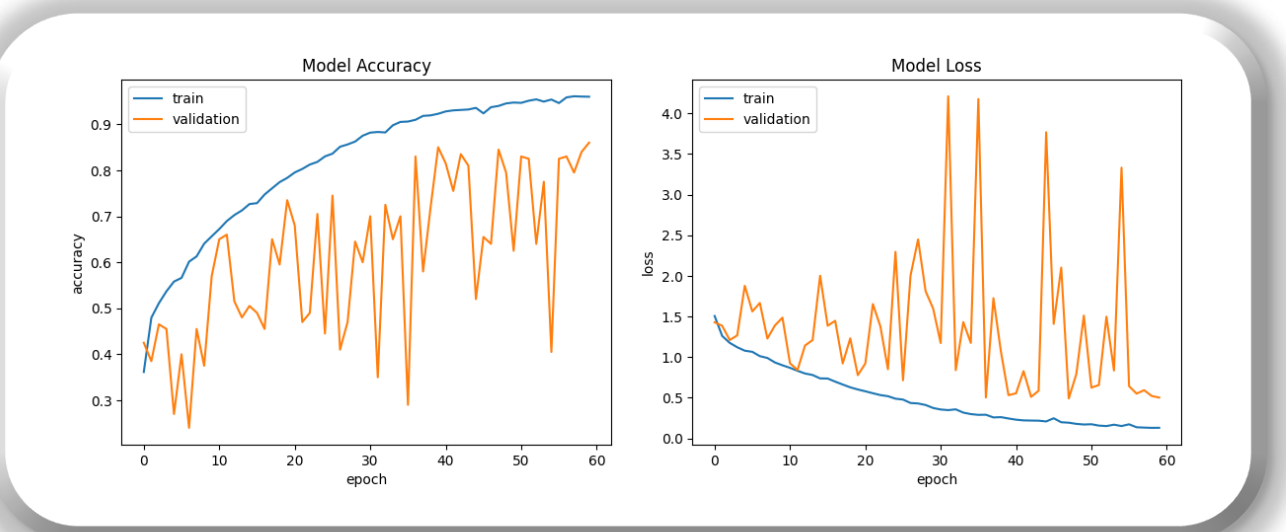
7.1. Implementation

Our final Model had us double the Training dataset using random augmenters that flip, rotate, zoom, shift and mess with the images' brightness. This allowed the training dataset to increase from being at 9700 image to 19400 image. So, in theory this should drastically improve our accuracy.

```
aug_gen = ImageDataGenerator(
          rotation_range=10, # rotation
          width_shift_range=0.2, # horizontal shift
          height_shift_range=0.2, # vertical shift
          zoom_range=0.2, # zoom
          horizontal_flip=True, # horizontal flip
          brightness_range=[0.2,1.2]) # brightness

aug_data = aug_gen.flow_from_dataframe(
    dataframe = trainData,
    x_col = 0,
    y_col = 1,
    target_size = (224,224),
    color_mode = 'rgb',
    class_mode = 'categorical',
    shuffle = True
)
```

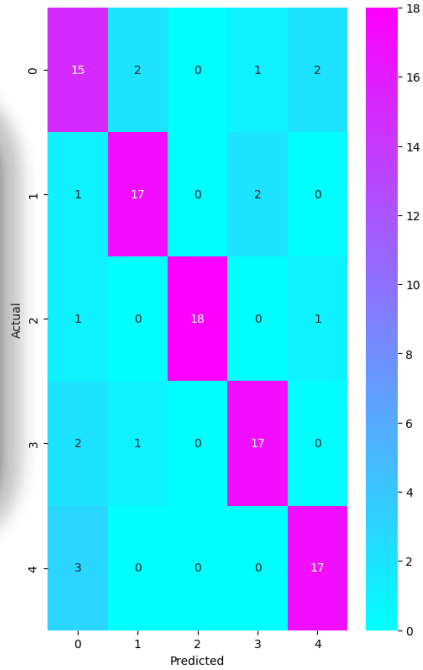
```python
def combine_gen(*gens):
    while True:
        for g in gens:
            yield next(g)

history1 = model.fit(combine_gen(train_data, aug_data),
steps_per_epoch=len(train_data)+len(aug_data), epochs=60, validation_data=val_data)
```

7.2. Validation and Testing Observations



As we can see the validation accuracy kept fluctuating massively but ended on a high of around 85% which is a significant improvement, the fluctuations could be a result of the validation data being very small in size relative to the training data.

	precision	recall	f1-score	support
0	0.68	0.75	0.71	20
1	0.85	0.85	0.85	20
2	1.00	0.90	0.95	20
3	0.85	0.85	0.85	20
4	0.85	0.85	0.85	20
accuracy			0.84	100
macro avg	0.85	0.84	0.84	100
weighted avg	0.85	0.84	0.84	100

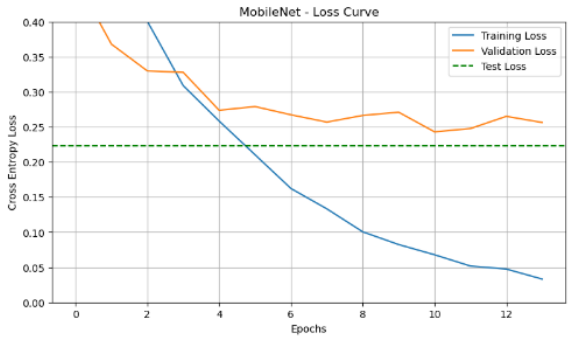


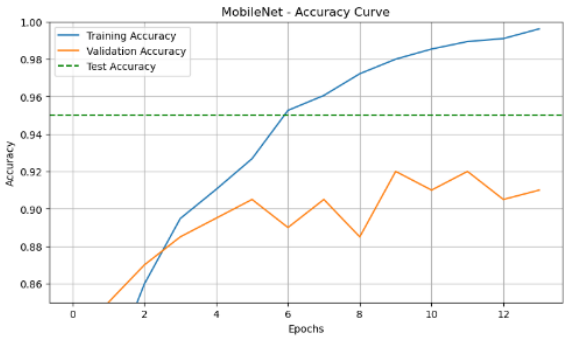
8. Comparisons

We can see clearly that VGG-19 was a significant improvement over VGG-16 but what about other models created online.

In a model created by a user named Chesta on Kaggle for the same dataset. ResNet50 was used. However, it was noted that they could only reach around 85% accuracy on validation and testing. There were also massive fluctuations like Model 2.3 with their val_accuracy midway reaching a high of 92% but dropping immediately to around 76% showing that it was just a coincidence.

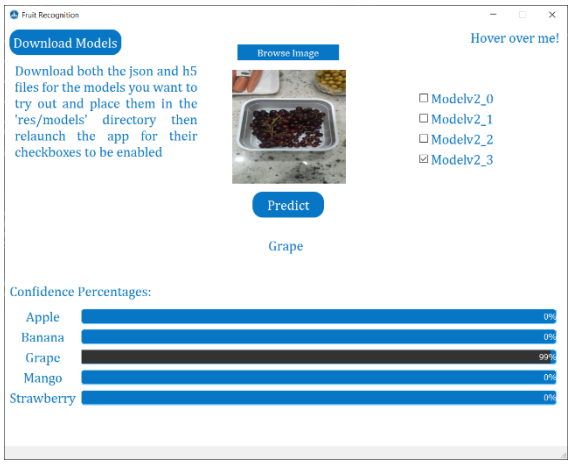
Another fantastic notebook that was created by a user named DeepNets on Kaggle for the same dataset shows a comparison between all the backbone architectures and networks which include ResNet50V2, ResNet152V2, InceptionV3, Xception, and MobileNetV2. Showing that none of them could break the 85% accuracy mark except for MobileNetV2, which after some hypertuning reached a whopping 95% accuracy which is the highest yet.

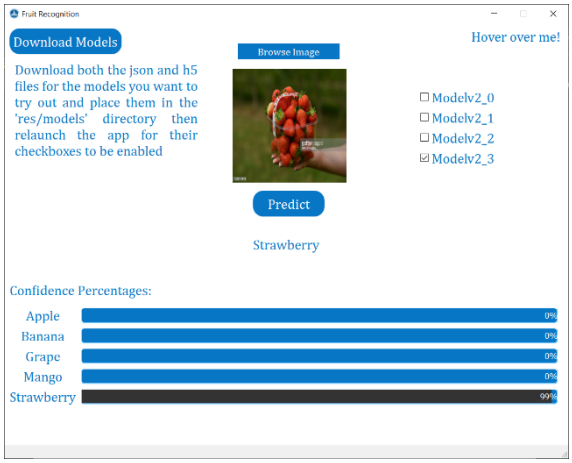


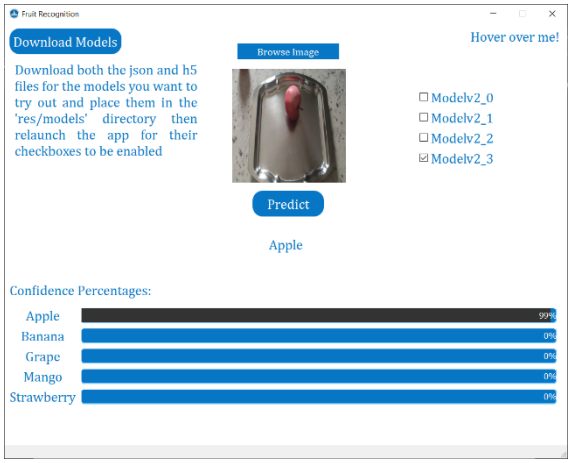


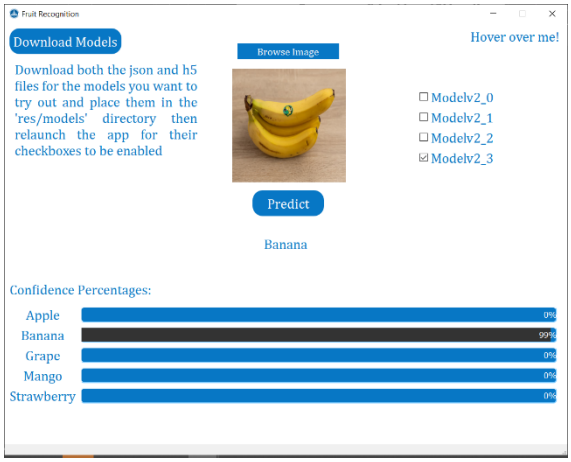
	precision	recall	f1-score	support
0	0.86	0.95	0.90	20
1	0.95	1.00	0.98	20
2	0.95	0.95	0.95	20
3	1.00	0.95	0.97	20
4	1.00	0.90	0.95	20
accuracy			0.95	100
macro avg	0.95	0.95	0.95	100
weighted avg	0.95	0.95	0.95	100

9. Screenshots









10. Important Links and Information

GitHub Repository

Executable Download

Models Download

After you download both the json and h5 files for the model place them in the ‘res/models’ directory next to the exe and relaunch the exe for them to be detected.

This build was created using Pyinstaller and the source code as well as the spec file can be found under the directory ‘Project Program’ in the GitHub repository.

First Dataset

Second Dataset

Here’s a list of the pips required to run the notebooks:

- `keras`
- `tensorflow`
- `numpy`
- `seaborn`
- `pandas`
- `matplotlib`
- `opencv-python`
- `scikit-learn`

Here’s a list of the pips required to run the python program file:

(THE EXE DOESN’T REQUIRE ANY PREREQUISITES TO RUN)

- `keras`
- `tensorflow`
- `numpy`