```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
import pandas as pd
import seaborn as sns
import random
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout, Activation, BatchNormalization
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.optimizers import Adam
from keras.utils import plot_model
from PIL import Image
from collections import Counter
from glob import glob
from tensorflow.keras.utils import plot_model
```

Loading The Data

```python
train_path = 'fruits-360_dataset/fruits-360/Training/'
test_path = 'fruits-360_dataset/fruits-360/Test/'
```

Dataset Overview

Title: Fruit and Vegetable Image Recognition
Source: Kaggle Fruits- 360 Dataset

Dataset Characteristics:

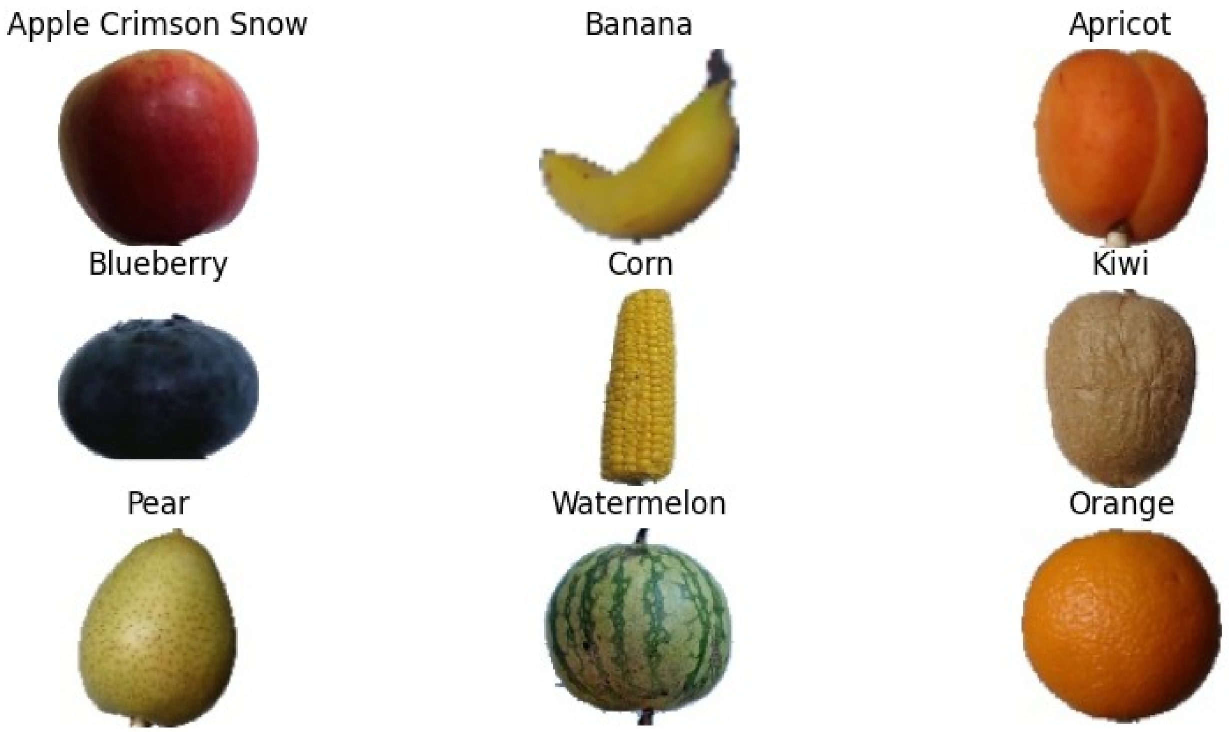
- Total number of images: 90,483
- Number of classes: 131
- Image dimensions: 100x100 pixels
- Training set size: 67,692 images
- Test set size: 22,688 images

Credit for base CNN: https://medium.com/hackerdawn/fruit-image-classification-using-cnn-on-google-colab-4fe7274418a5

```
img = load_img(train_path + "Apple Braeburn/0_100.jpg",
target_size=(100,100))
plt.imshow(img)
plt.axis("off")
plt.show()

x = img_to_array(img)
print(x.shape)
```

```
(100, 100, 3)
```

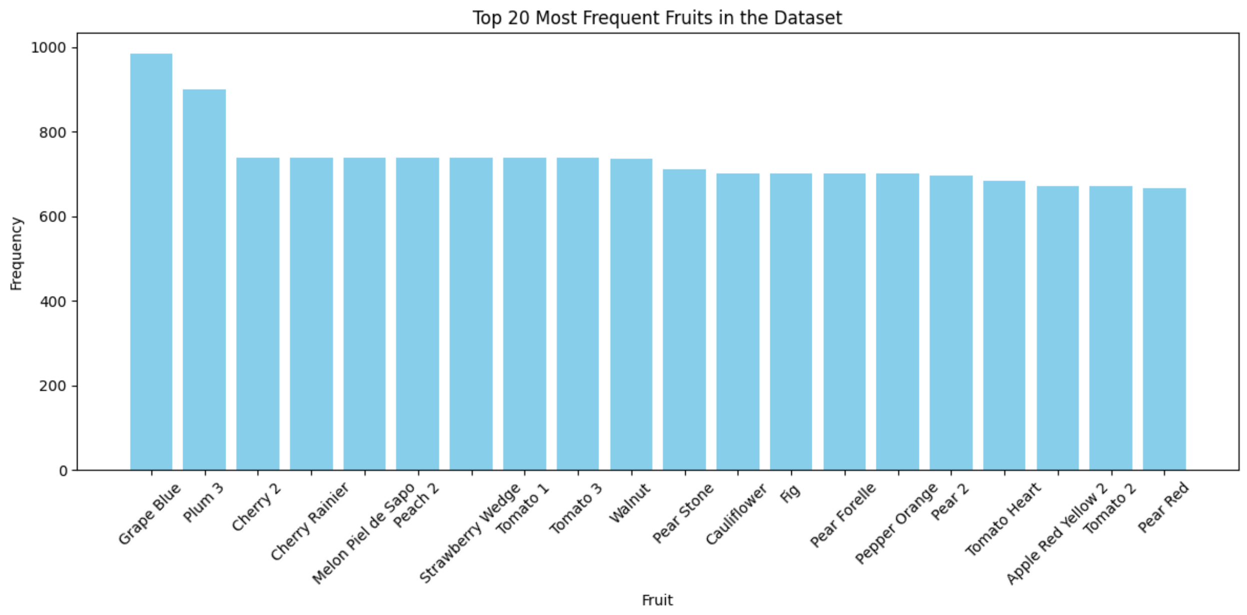
```
images = ['Apple Crimson Snow', 'Banana', 'Apricot', 'Blueberry',
'Corn', 'Kiwi', 'Pear', 'Watermelon', 'Orange']
fig = plt.figure(figsize =(10,5))
for i in range(len(images)):
    ax = fig.add_subplot(3,3,i+1,xticks=[],yticks=[])
    plt.title(images[i])
    plt.axis("off")
    ax.imshow(load_img(train_path + images[i] +"/0_100.jpg",
target_size=(100,100)))
```



```
fruits = []
fruits_image = []
for i in os.listdir(train_path):
    for image_filename in os.listdir(train_path + i):
        fruits.append(i)
        fruits_image.append(i + '/' + image_filename)


newData = Counter(fruits)
frequent_fruits = newData.most_common(20)
print("Top 20 frequent Fruits:")
frequent_fruits
```

```
Top 20 frequent Fruits:
```

```
[('Grape Blue', 984),
 ('Plum 3', 900),
 ('Cherry 2', 738),
 ('Cherry Rainier', 738),
 ('Melon Piel de Sapo', 738),
 ('Peach 2', 738),
 ('Strawberry Wedge', 738),
 ('Tomato 1', 738),
 ('Tomato 3', 738),
 ('Walnut', 735),
 ('Pear Stone', 711),
 ('Cauliflower', 702),
 ('Fig', 702),
 ('Pear Forelle', 702),
 ('Pepper Orange', 702),
 ('Pear 2', 696),
 ('Tomato Heart', 684),
 ('Apple Red Yellow 2', 672),
 ('Tomato 2', 672),
 ('Pear Red', 666)]

fruit_names = [fruit[0] for fruit in frequent_fruits]
fruit_counts = [fruit[1] for fruit in frequent_fruits]

plt.figure(figsize=(12, 6))
plt.bar(fruit_names, fruit_counts, color='skyblue')
plt.xlabel('Fruit')
plt.ylabel('Frequency')
plt.title('Top 20 Most Frequent Fruits in the Dataset')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Model

```
className = glob(train_path + '/*')
number_of_class = len(className)
```

```
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape = x.shape))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(32,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Conv2D(64,(3,3)))
model.add(Activation("relu"))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(number_of_class))
model.add(Activation("softmax"))

model.compile(loss = "categorical_crossentropy",
optimizer = "rmsprop",
metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━┩
│ conv2d_3 (Conv2D)               │ (None, 98, 98, 32)     │
896 │
├─────────────────────────────────┼────────────────────────┼───────
─────┤
│ activation_5 (Activation)       │ (None, 98, 98, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼───────
─────┤
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 49, 49, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼───────
─────┤
│ conv2d_4 (Conv2D)               │ (None, 47, 47, 32)     │
9,248 │
├─────────────────────────────────┼────────────────────────┼───────
─────┤
│ activation_6 (Activation)       │ (None, 47, 47, 32)     │
```

```
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ max_pooling2d_4 (MaxPooling2D)  │ (None, 23, 23, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ conv2d_5 (Conv2D)               │ (None, 21, 21, 64)     │
18,496 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ activation_7 (Activation)       │ (None, 21, 21, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ max_pooling2d_5 (MaxPooling2D)  │ (None, 10, 10, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ flatten_1 (Flatten)             │ (None, 6400)           │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dense_2 (Dense)                 │ (None, 1024)           │
6,554,624 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ activation_8 (Activation)       │ (None, 1024)           │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dropout_1 (Dropout)             │ (None, 1024)           │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ dense_3 (Dense)                 │ (None, 131)            │
134,275 │
├─────────────────────────────────┼────────────────────────┼──────────
───────┤
│ activation_9 (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)
```

```
epochs = 100
batch_size = 64
```

Augmented Data

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.3,
    horizontal_flip=True,
    zoom_range=0.3
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    directory = train_path,
    target_size= x.shape[:2],
    batch_size = batch_size,
    color_mode= "rgb",
    class_mode= "categorical"
)

test_generator = test_datagen.flow_from_directory(
    directory = test_path,
    target_size= x.shape[:2],
    batch_size = batch_size,
    color_mode= "rgb",
    class_mode= "categorical"
)
```

```
Found 67692 images belonging to 131 classes.
Found 22688 images belonging to 131 classes.
```

Fitting the model

```
hist = model.fit(
    x=train_generator,
    steps_per_epoch=1600 // batch_size,
    epochs=epochs,
    validation_data=test_generator,
    validation_steps=800 // batch_size
)
```

```
Epoch 1/100
```

```
c:\Users\adamn\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120:
UserWarning: Your `PyDataset` class should call
`super().__init__(**kwargs)` in its constructor. `**kwargs` can
include `workers`, `use_multiprocessing`, `max_queue_size`. Do not
pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()

25/25 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.0155 - loss:
4.9032 - val_accuracy: 0.0339 - val_loss: 4.6321
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 896ms/step - accuracy: 0.0357 - loss:
4.5676 - val_accuracy: 0.0911 - val_loss: 3.8680
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 816ms/step - accuracy: 0.0756 - loss:
3.9581 - val_accuracy: 0.1536 - val_loss: 3.3425
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 645ms/step - accuracy: 0.1032 - loss:
3.6045 - val_accuracy: 0.2148 - val_loss: 2.9657
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 660ms/step - accuracy: 0.1923 - loss:
3.1241 - val_accuracy: 0.2917 - val_loss: 2.6141
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 787ms/step - accuracy: 0.2547 - loss:
2.7536 - val_accuracy: 0.4583 - val_loss: 2.1070
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 668ms/step - accuracy: 0.2606 - loss:
2.6075 - val_accuracy: 0.5560 - val_loss: 1.7412
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 663ms/step - accuracy: 0.3575 - loss:
2.1925 - val_accuracy: 0.5664 - val_loss: 1.6075
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 680ms/step - accuracy: 0.3864 - loss:
2.0316 - val_accuracy: 0.5365 - val_loss: 1.6849
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 647ms/step - accuracy: 0.4462 - loss:
1.8306 - val_accuracy: 0.6432 - val_loss: 1.3559
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 654ms/step - accuracy: 0.5071 - loss:
1.5842 - val_accuracy: 0.7266 - val_loss: 1.0184
Epoch 12/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 637ms/step - accuracy: 0.5518 - loss:
1.4552 - val_accuracy: 0.6784 - val_loss: 1.0842
Epoch 13/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 645ms/step - accuracy: 0.6137 - loss:
1.2439 - val_accuracy: 0.7604 - val_loss: 0.7982
Epoch 14/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 622ms/step - accuracy: 0.6011 - loss:
1.2616 - val_accuracy: 0.6771 - val_loss: 1.0708
Epoch 15/100
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 611ms/step - accuracy: 0.6136 - loss:
1.2278 - val_accuracy: 0.7839 - val_loss: 0.7646
Epoch 16/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 651ms/step - accuracy: 0.6506 - loss:
1.0628 - val_accuracy: 0.8346 - val_loss: 0.6399
Epoch 17/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 698ms/step - accuracy: 0.6981 - loss:
0.9152 - val_accuracy: 0.8255 - val_loss: 0.6009
Epoch 18/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 710ms/step - accuracy: 0.7365 - loss:
0.8702 - val_accuracy: 0.8385 - val_loss: 0.5897
Epoch 19/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 686ms/step - accuracy: 0.7089 - loss:
0.8786 - val_accuracy: 0.7500 - val_loss: 0.7517
Epoch 20/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 681ms/step - accuracy: 0.6862 - loss:
0.9383 - val_accuracy: 0.8333 - val_loss: 0.5603
Epoch 21/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 653ms/step - accuracy: 0.7594 - loss:
0.7431 - val_accuracy: 0.8516 - val_loss: 0.4850
Epoch 22/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 568ms/step - accuracy: 0.7745 - loss:
0.7082 - val_accuracy: 0.7357 - val_loss: 0.8744
Epoch 23/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 598ms/step - accuracy: 0.7620 - loss:
0.7071 - val_accuracy: 0.8685 - val_loss: 0.4581
Epoch 24/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 588ms/step - accuracy: 0.8019 - loss:
0.6223 - val_accuracy: 0.8568 - val_loss: 0.4625
Epoch 25/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 565ms/step - accuracy: 0.8242 - loss:
0.5616 - val_accuracy: 0.8542 - val_loss: 0.4282
Epoch 26/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 553ms/step - accuracy: 0.7786 - loss:
0.6924 - val_accuracy: 0.8828 - val_loss: 0.3729
Epoch 27/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 558ms/step - accuracy: 0.8177 - loss:
0.5418 - val_accuracy: 0.8789 - val_loss: 0.3964
Epoch 28/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 531ms/step - accuracy: 0.8086 - loss:
0.6033 - val_accuracy: 0.7956 - val_loss: 0.6671
Epoch 29/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 542ms/step - accuracy: 0.8144 - loss:
0.5980 - val_accuracy: 0.8477 - val_loss: 0.5070
Epoch 30/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 535ms/step - accuracy: 0.8289 - loss:
0.5321 - val_accuracy: 0.8534 - val_loss: 0.4676
Epoch 31/100
```

```
c:\Users\adamn\AppData\Local\Programs\Python\Python311\Lib\
contextlib.py:155: UserWarning: Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches. You may need to
use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)

25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 634ms/step - accuracy: 0.8224 - loss:
0.5218 - val_accuracy: 0.8854 - val_loss: 0.3928
Epoch 32/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 559ms/step - accuracy: 0.8543 - loss:
0.4470 - val_accuracy: 0.9115 - val_loss: 0.3055
Epoch 33/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 524ms/step - accuracy: 0.8776 - loss:
0.3978 - val_accuracy: 0.8958 - val_loss: 0.3904
Epoch 34/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 543ms/step - accuracy: 0.8724 - loss:
0.4136 - val_accuracy: 0.9036 - val_loss: 0.3497
Epoch 35/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 615ms/step - accuracy: 0.8716 - loss:
0.3837 - val_accuracy: 0.9271 - val_loss: 0.2213
Epoch 36/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 627ms/step - accuracy: 0.8747 - loss:
0.3846 - val_accuracy: 0.9336 - val_loss: 0.2214
Epoch 37/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 608ms/step - accuracy: 0.8872 - loss:
0.3367 - val_accuracy: 0.8763 - val_loss: 0.3846
Epoch 38/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 580ms/step - accuracy: 0.8426 - loss:
0.4743 - val_accuracy: 0.9089 - val_loss: 0.3090
Epoch 39/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 630ms/step - accuracy: 0.8862 - loss:
0.3530 - val_accuracy: 0.9089 - val_loss: 0.2730
Epoch 40/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 644ms/step - accuracy: 0.8989 - loss:
0.3220 - val_accuracy: 0.8945 - val_loss: 0.3653
Epoch 41/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 618ms/step - accuracy: 0.8721 - loss:
0.4229 - val_accuracy: 0.9049 - val_loss: 0.2685
Epoch 42/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 536s 22s/step - accuracy: 0.8933 - loss:
0.3362 - val_accuracy: 0.9401 - val_loss: 0.2523
Epoch 43/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 6s 198ms/step - accuracy: 0.8661 - loss:
0.4096 - val_accuracy: 0.8828 - val_loss: 0.3680
Epoch 44/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 25s 858ms/step - accuracy: 0.8987 - loss:
0.3102 - val_accuracy: 0.9323 - val_loss: 0.2154
Epoch 45/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 869ms/step - accuracy: 0.9102 - loss:
```

```
0.2597 - val_accuracy: 0.9167 - val_loss: 0.3112
Epoch 46/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 30s 1s/step - accuracy: 0.9079 - loss:
0.2747 - val_accuracy: 0.9297 - val_loss: 0.2825
Epoch 47/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 23s 914ms/step - accuracy: 0.8993 - loss:
0.3322 - val_accuracy: 0.9167 - val_loss: 0.3388
Epoch 48/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 771ms/step - accuracy: 0.9003 - loss:
0.2969 - val_accuracy: 0.9049 - val_loss: 0.2696
Epoch 49/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 766ms/step - accuracy: 0.9134 - loss:
0.2714 - val_accuracy: 0.8034 - val_loss: 0.6712
Epoch 50/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 791ms/step - accuracy: 0.9180 - loss:
0.2679 - val_accuracy: 0.9245 - val_loss: 0.2539
Epoch 51/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 846ms/step - accuracy: 0.8887 - loss:
0.3421 - val_accuracy: 0.9062 - val_loss: 0.3192
Epoch 52/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 712ms/step - accuracy: 0.9032 - loss:
0.2909 - val_accuracy: 0.9154 - val_loss: 0.2352
Epoch 53/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 24s 985ms/step - accuracy: 0.9031 - loss:
0.2764 - val_accuracy: 0.8112 - val_loss: 0.6436
Epoch 54/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 24s 981ms/step - accuracy: 0.9111 - loss:
0.2893 - val_accuracy: 0.9362 - val_loss: 0.2041
Epoch 55/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 811ms/step - accuracy: 0.9328 - loss:
0.2052 - val_accuracy: 0.9453 - val_loss: 0.1981
Epoch 56/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 764ms/step - accuracy: 0.8929 - loss:
0.2724 - val_accuracy: 0.8958 - val_loss: 0.3159
Epoch 57/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 708ms/step - accuracy: 0.9323 - loss:
0.2121 - val_accuracy: 0.9336 - val_loss: 0.2355
Epoch 58/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 675ms/step - accuracy: 0.9186 - loss:
0.2236 - val_accuracy: 0.9349 - val_loss: 0.1716
Epoch 59/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 635ms/step - accuracy: 0.9421 - loss:
0.1598 - val_accuracy: 0.9505 - val_loss: 0.2241
Epoch 60/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 639ms/step - accuracy: 0.9295 - loss:
0.2465 - val_accuracy: 0.9639 - val_loss: 0.1678
Epoch 61/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 829ms/step - accuracy: 0.9204 - loss:
0.2243 - val_accuracy: 0.9414 - val_loss: 0.1853
```

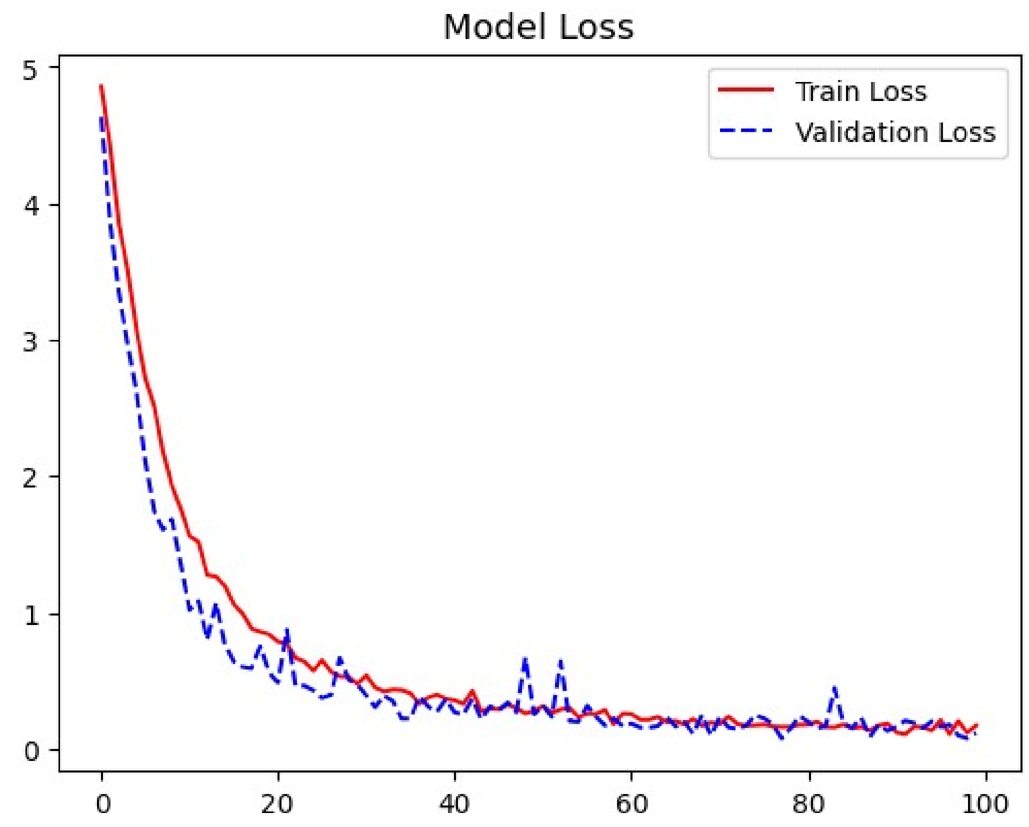
```
Epoch 62/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 751ms/step - accuracy: 0.9311 - loss:
0.2061 - val_accuracy: 0.9557 - val_loss: 0.1570
Epoch 63/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 663ms/step - accuracy: 0.9280 - loss:
0.2147 - val_accuracy: 0.9544 - val_loss: 0.1568
Epoch 64/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 595ms/step - accuracy: 0.9178 - loss:
0.2469 - val_accuracy: 0.9492 - val_loss: 0.1677
Epoch 65/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 593ms/step - accuracy: 0.9284 - loss:
0.2176 - val_accuracy: 0.9271 - val_loss: 0.2306
Epoch 66/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 615ms/step - accuracy: 0.9146 - loss:
0.2355 - val_accuracy: 0.9583 - val_loss: 0.1645
Epoch 67/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 592ms/step - accuracy: 0.9415 - loss:
0.1631 - val_accuracy: 0.9453 - val_loss: 0.1972
Epoch 68/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 598ms/step - accuracy: 0.9384 - loss:
0.1988 - val_accuracy: 0.9727 - val_loss: 0.1101
Epoch 69/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 606ms/step - accuracy: 0.9414 - loss:
0.1780 - val_accuracy: 0.9414 - val_loss: 0.2507
Epoch 70/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 601ms/step - accuracy: 0.9380 - loss:
0.1702 - val_accuracy: 0.9661 - val_loss: 0.0969
Epoch 71/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 597ms/step - accuracy: 0.9414 - loss:
0.1788 - val_accuracy: 0.9453 - val_loss: 0.2345
Epoch 72/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 14s 571ms/step - accuracy: 0.9382 - loss:
0.2138 - val_accuracy: 0.9440 - val_loss: 0.1587
Epoch 73/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 713ms/step - accuracy: 0.9554 - loss:
0.1440 - val_accuracy: 0.9596 - val_loss: 0.1454
Epoch 74/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 610ms/step - accuracy: 0.9384 - loss:
0.1797 - val_accuracy: 0.9583 - val_loss: 0.1718
Epoch 75/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 607ms/step - accuracy: 0.9430 - loss:
0.1613 - val_accuracy: 0.9245 - val_loss: 0.2478
Epoch 76/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 718ms/step - accuracy: 0.9363 - loss:
0.1952 - val_accuracy: 0.9336 - val_loss: 0.2234
Epoch 77/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 670ms/step - accuracy: 0.9471 - loss:
0.1735 - val_accuracy: 0.9570 - val_loss: 0.1801
Epoch 78/100
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 703ms/step - accuracy: 0.9369 - loss:
0.1933 - val_accuracy: 0.9688 - val_loss: 0.0760
Epoch 79/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 734ms/step - accuracy: 0.9321 - loss:
0.1955 - val_accuracy: 0.9492 - val_loss: 0.1476
Epoch 80/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 706ms/step - accuracy: 0.9411 - loss:
0.1811 - val_accuracy: 0.9154 - val_loss: 0.2404
Epoch 81/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 719ms/step - accuracy: 0.9276 - loss:
0.1974 - val_accuracy: 0.9414 - val_loss: 0.1991
Epoch 82/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 742ms/step - accuracy: 0.9488 - loss:
0.1503 - val_accuracy: 0.9492 - val_loss: 0.1502
Epoch 83/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 716ms/step - accuracy: 0.9544 - loss:
0.1544 - val_accuracy: 0.9609 - val_loss: 0.1680
Epoch 84/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 698ms/step - accuracy: 0.9523 - loss:
0.1385 - val_accuracy: 0.8997 - val_loss: 0.4473
Epoch 85/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 670ms/step - accuracy: 0.9198 - loss:
0.2810 - val_accuracy: 0.9583 - val_loss: 0.1869
Epoch 86/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 5s 200ms/step - accuracy: 0.9584 - loss:
0.1381 - val_accuracy: 0.9557 - val_loss: 0.1537
Epoch 87/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 28s 894ms/step - accuracy: 0.9450 - loss:
0.1475 - val_accuracy: 0.9466 - val_loss: 0.2282
Epoch 88/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 820ms/step - accuracy: 0.9514 - loss:
0.1357 - val_accuracy: 0.9714 - val_loss: 0.0851
Epoch 89/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 715ms/step - accuracy: 0.9600 - loss:
0.1269 - val_accuracy: 0.9557 - val_loss: 0.1820
Epoch 90/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 774ms/step - accuracy: 0.9453 - loss:
0.1844 - val_accuracy: 0.9663 - val_loss: 0.1307
Epoch 91/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 24s 971ms/step - accuracy: 0.9626 - loss:
0.1054 - val_accuracy: 0.9622 - val_loss: 0.1649
Epoch 92/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 892ms/step - accuracy: 0.9668 - loss:
0.0992 - val_accuracy: 0.9336 - val_loss: 0.2061
Epoch 93/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 820ms/step - accuracy: 0.9382 - loss:
0.2057 - val_accuracy: 0.9466 - val_loss: 0.1889
Epoch 94/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 747ms/step - accuracy: 0.9496 - loss:
```

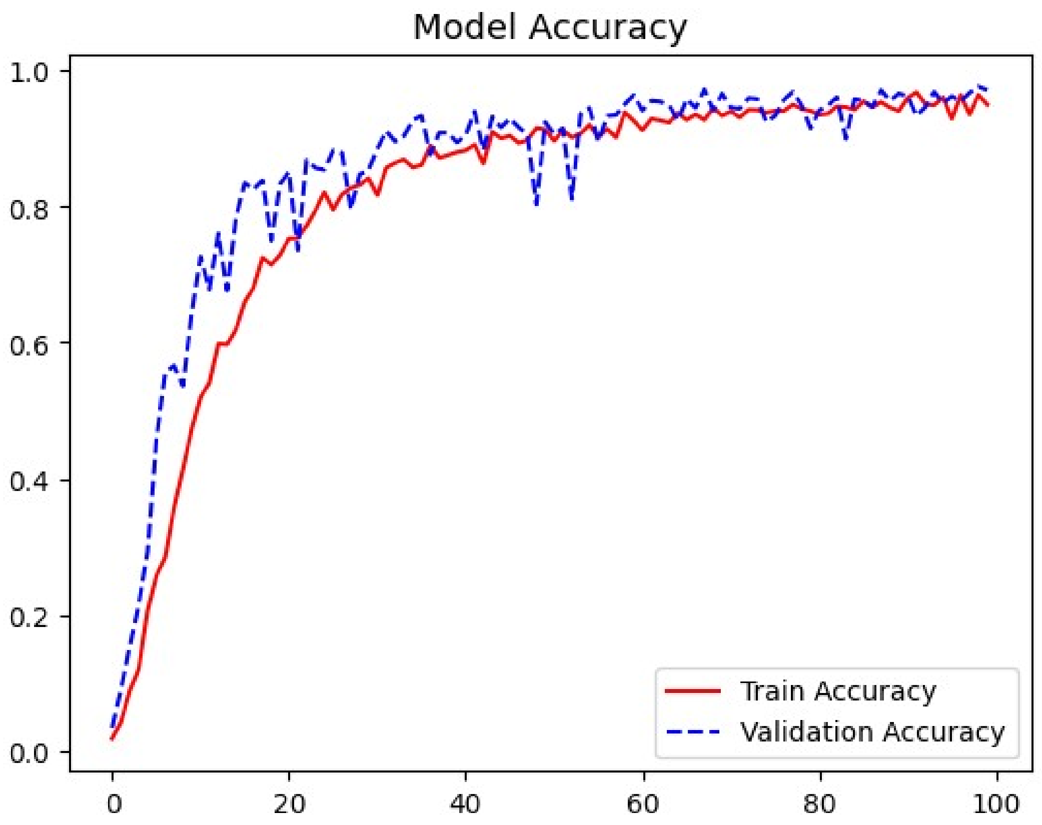
```
0.1690 - val_accuracy: 0.9688 - val_loss: 0.1536
Epoch 95/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 757ms/step - accuracy: 0.9566 - loss:
0.1446 - val_accuracy: 0.9518 - val_loss: 0.2025
Epoch 96/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 813ms/step - accuracy: 0.9418 - loss:
0.1785 - val_accuracy: 0.9622 - val_loss: 0.1621
Epoch 97/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 703ms/step - accuracy: 0.9659 - loss:
0.0995 - val_accuracy: 0.9544 - val_loss: 0.1786
Epoch 98/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 831ms/step - accuracy: 0.9260 - loss:
0.2235 - val_accuracy: 0.9674 - val_loss: 0.0970
Epoch 99/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 674ms/step - accuracy: 0.9673 - loss:
0.1011 - val_accuracy: 0.9766 - val_loss: 0.0771
Epoch 100/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 672ms/step - accuracy: 0.9515 - loss:
0.1677 - val_accuracy: 0.9714 - val_loss: 0.1154
```

Evaluation

```
plt.figure()
plt.plot(hist.history["loss"],label = "Train Loss", color = "red")
plt.plot(hist.history["val_loss"],label = "Validation Loss", color =
"blue", linestyle="dashed",markeredgecolor = "purple", markeredgewidth
= 2)
plt.title("Model Loss",  size = 13)
plt.legend()
plt.show()
```



```
plt.figure()
plt.plot(hist.history["accuracy"],label = "Train Accuracy", color =
"red")
plt.plot(hist.history["val_accuracy"],label = "Validation Accuracy",
color = "blue", linestyle="dashed",markeredgecolor = "purple",
markeredgewidth = 2)
plt.title("Model Accuracy", size = 13)
plt.legend()
plt.show()
```

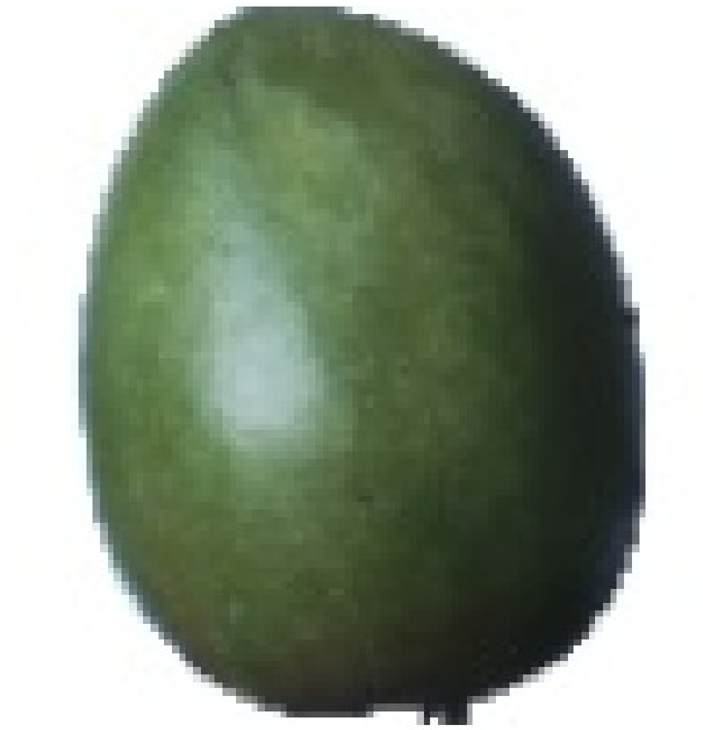


```
loss, accuracy = model.evaluate(test_generator, steps=800 //
batch_size)

print("Test Loss:", loss)
print("Test Accuracy:", accuracy)

12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 250ms/step - accuracy: 0.9542 - loss:
0.1452
Test Loss: 0.14572185277938843
Test Accuracy: 0.96484375
```

Testing

```
def load_and_preprocess_image(filename):
    np_image = Image.open(filename)
    np_image = np_image.resize((100, 100))
    np_image = np.array(np_image).astype('float32') / 255
    np_image = np.expand_dims(np_image, axis=0)
    return np_image

image = load_and_preprocess_image(test_path + "/Mango/0_100.jpg")
```

```
plt.imshow(np.squeeze(image))
plt.axis("off")
plt.show()
```



```
prediction = model.predict(image)
predicted_class = np.argmax(prediction, axis=-1)
print("Predicted class index:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Predicted class index: [64]
```

```
class_indices = test_generator.class_indices
predicted_class_label = [label for label, index in
class_indices.items() if index == 64]
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: ['Mango']
```

```
img_path =
'fruits-360_dataset/fruits-360/test-multiple_fruits/Bananas(lady_finge
r)1.jpg'
preprocessed_image = load_and_preprocess_image(img_path)
```

```
plt.imshow(np.squeeze(preprocessed_image), interpolation='nearest')
plt.axis('off')
plt.show()
```

```
prediction = model.predict(preprocessed_image)
predicted_class = np.argmax(prediction, axis=-1)
print("Predicted class index:", predicted_class)
```

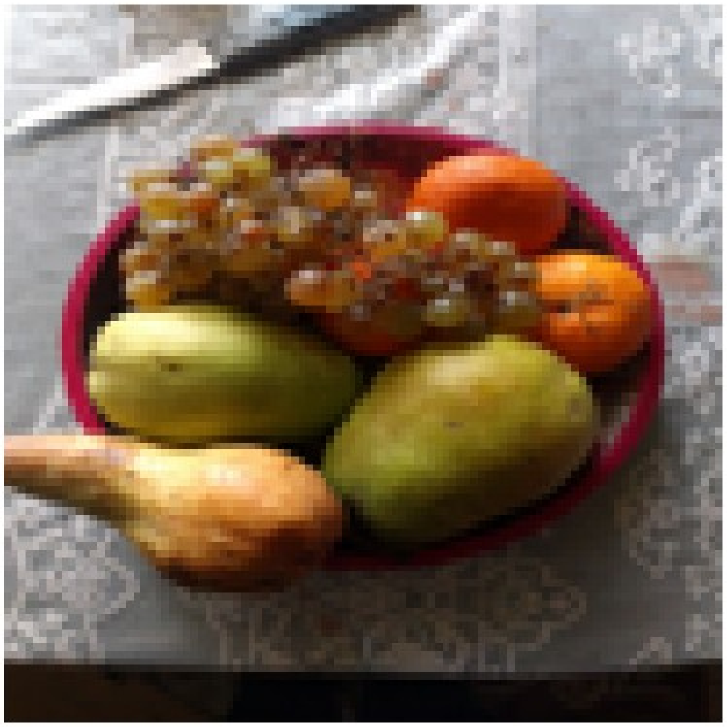
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Predicted class index: [26]
```

```
predicted_class_label = [label for label, index in
class_indices.items() if index == 17]
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: ['Banana Lady Finger']
```

```
img_path =
'fruits-360_dataset/fruits-360/test-multiple_fruits/grape_pear_mandari
ne.jpg'
preprocessed_image = load_and_preprocess_image(img_path)
```

```
plt.imshow(np.squeeze(preprocessed_image), interpolation='nearest')
plt.axis('off')
plt.show()
```



```
prediction = model.predict(preprocessed_image)
predicted_class = np.argmax(prediction, axis=-1)
print("Predicted class index:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Predicted class index: [37]
```

```
predicted_class_label = [label for label, index in
class_indices.items() if index == 116]
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: ['Strawberry']
```

Model Evaluation

Based off what we see from the Model evaluation visualizations and training loss and decreases really quickly with the exact opposite effect for the Model Accuracy. By prinint out the loss and accuracy we can get a better understanding of the loss and accuracy of the model which the Test Loss: 0.25146737694740295 and the Test Accurac is 0.9244791865348816. From testing multiple images above we can see a succsuss rate with individual images like the mango but with testing multiple images it geta s little confusing when working with images that have multiple differnt fruit (like the grap, pears, and mandarines). However it picks up on multiple of the same fruit fairly well(Banana Lady Finger).

Alt Models

```
input_shape = x.shape

# Batch Normalization Model
def create_bn_model():
    model = Sequential([
        Conv2D(32, (3, 3), input_shape=input_shape),
        Activation('relu'),
        BatchNormalization(),
        MaxPooling2D(),
        Conv2D(64, (3, 3)),
        Activation('relu'),
        BatchNormalization(),
        MaxPooling2D(),
        Conv2D(128, (3, 3)),
        Activation('relu'),
        BatchNormalization(),
        MaxPooling2D(),
        Flatten(),
        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(number_of_class, activation='softmax')
    ])
    model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Adjusted Learning Rate and Additional Dropout
def create_lr_dropout_model():
    model = Sequential([
        Conv2D(32, (3, 3), input_shape=input_shape),
        Activation('relu'),
        MaxPooling2D(),
        Dropout(0.3),
        Conv2D(64, (3, 3)),
        Activation('relu'),
        MaxPooling2D(),
        Dropout(0.3),
        Conv2D(128, (3, 3)),
        Activation('relu'),
        MaxPooling2D(),
        Flatten(),
        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(number_of_class, activation='softmax')
    ])
    optimizer = Adam(learning_rate=0.0005)
    model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
    return model

bn_model = create_bn_model()
lr_dropout_model = create_lr_dropout_model()

print("\nBatch Normalization Model Summary:")
bn_model.summary()
print("\nLearning Rate and Dropout Model Summary:")
lr_dropout_model.summary()
```

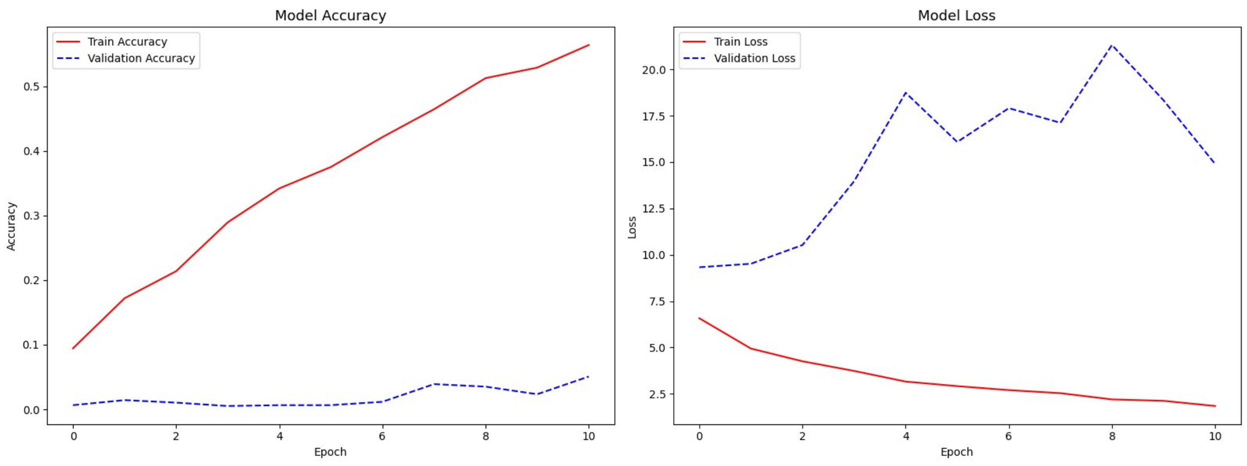
```
Batch Normalization Model Summary:

Model: "sequential_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d_6 (Conv2D)                │ (None, 98, 98, 32)     │
896 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ activation_10 (Activation)       │ (None, 98, 98, 32)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization              │ (None, 98, 98, 32)     │
128 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_6 (MaxPooling2D)   │ (None, 49, 49, 32)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_7 (Conv2D)                │ (None, 47, 47, 64)     │
18,496 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ activation_11 (Activation)       │ (None, 47, 47, 64)     │
```

```
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ batch_normalization_1            │ (None, 47, 47, 64)      │
256 │
│ (BatchNormalization)             │                         │
│
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ max_pooling2d_7 (MaxPooling2D)   │ (None, 23, 23, 64)      │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ conv2d_8 (Conv2D)                │ (None, 21, 21, 128)     │
73,856 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ activation_12 (Activation)       │ (None, 21, 21, 128)     │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ batch_normalization_2            │ (None, 21, 21, 128)     │
512 │
│ (BatchNormalization)             │                         │
│
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ max_pooling2d_8 (MaxPooling2D)   │ (None, 10, 10, 128)     │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ flatten_2 (Flatten)              │ (None, 12800)           │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ dense_4 (Dense)                  │ (None, 1024)            │
13,108,224 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ dropout_2 (Dropout)              │ (None, 1024)            │
0 │
├──────────────────────────────────┼─────────────────────────┼───────────
──────┤
│ dense_5 (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)


Learning Rate and Dropout Model Summary:

Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d_9 (Conv2D)                │ (None, 98, 98, 32)      │
896 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ activation_13 (Activation)       │ (None, 98, 98, 32)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ max_pooling2d_9 (MaxPooling2D)   │ (None, 49, 49, 32)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ dropout_3 (Dropout)              │ (None, 49, 49, 32)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ conv2d_10 (Conv2D)               │ (None, 47, 47, 64)      │
18,496 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ activation_14 (Activation)       │ (None, 47, 47, 64)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ max_pooling2d_10 (MaxPooling2D)  │ (None, 23, 23, 64)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ dropout_4 (Dropout)              │ (None, 23, 23, 64)      │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ conv2d_11 (Conv2D)               │ (None, 21, 21, 128)     │
73,856 │
├──────────────────────────────────┼─────────────────────────┼──────────
```

```
───────┤
│ activation_15 (Activation)        │ (None, 21, 21, 128)      │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ max_pooling2d_11 (MaxPooling2D) │ (None, 10, 10, 128)      │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ flatten_3 (Flatten)               │ (None, 12800)            │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ dense_6 (Dense)                   │ (None, 1024)             │
13,108,224 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ dropout_5 (Dropout)               │ (None, 1024)             │
0 │
├───────────────────────────────────┼──────────────────────────┼──────────
───────┤
│ dense_7 (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)
```

```python
checkpoint_bn = ModelCheckpoint('best_model_bn.keras',
monitor='val_accuracy', mode='max', save_best_only=True)
early_stop_bn = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

checkpoint_lr_dropout = ModelCheckpoint('best_model_lr_dropout.keras',
monitor='val_accuracy', mode='max', save_best_only=True)
early_stop_lr_dropout = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the Batch Normalization Model
history_bn = bn_model.fit(
    x=train_generator,
    steps_per_epoch=1600 // batch_size,
    epochs=epochs,
    validation_data=test_generator,
    validation_steps=800 // batch_size,
    callbacks=[checkpoint_bn, early_stop_bn]
)

Epoch 1/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 29s 1s/step - accuracy: 0.0780 - loss:
7.2950 - val_accuracy: 0.0065 - val_loss: 9.3268
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.1544 - loss:
5.2696 - val_accuracy: 0.0143 - val_loss: 9.5143
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.2160 - loss:
4.2411 - val_accuracy: 0.0104 - val_loss: 10.5254
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.2858 - loss:
3.7232 - val_accuracy: 0.0052 - val_loss: 13.9616
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 27s 1s/step - accuracy: 0.3263 - loss:
3.2050 - val_accuracy: 0.0065 - val_loss: 18.7386
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 27s 1s/step - accuracy: 0.3549 - loss:
3.1484 - val_accuracy: 0.0065 - val_loss: 16.0833
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 906ms/step - accuracy: 0.4018 - loss:
2.7747 - val_accuracy: 0.0117 - val_loss: 17.9076
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 24s 960ms/step - accuracy: 0.4551 - loss:
2.5361 - val_accuracy: 0.0391 - val_loss: 17.1191
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 23s 949ms/step - accuracy: 0.5026 - loss:
2.1727 - val_accuracy: 0.0352 - val_loss: 21.3140
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 23s 951ms/step - accuracy: 0.4857 - loss:
2.4426 - val_accuracy: 0.0234 - val_loss: 18.3315
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 26s 1s/step - accuracy: 0.5790 - loss:
1.7099 - val_accuracy: 0.0508 - val_loss: 14.8973

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

axs[0].plot(history_bn.history['accuracy'], label='Train Accuracy',
color='red')
axs[0].plot(history_bn.history['val_accuracy'], label='Validation
Accuracy', color='blue', linestyle='dashed', markeredgecolor='purple',
markeredgewidth=2)
axs[0].set_title('Model Accuracy', size=13)
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend()
```

```
axs[1].plot(history_bn.history['loss'], label='Train Loss',
color='red')
axs[1].plot(history_bn.history['val_loss'], label='Validation Loss',
color='blue', linestyle='dashed', markeredgecolor='purple',
markeredgewidth=2)
axs[1].set_title('Model Loss', size=13)
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Loss')
axs[1].legend()

# Display the plots
plt.tight_layout()
plt.show()
```



```
final_train_accuracy = history_bn.history['accuracy'][-1]
final_val_accuracy = history_bn.history['val_accuracy'][-1]
final_train_loss = history_bn.history['loss'][-1]
final_val_loss = history_bn.history['val_loss'][-1]

print(f"Final Training Accuracy: {final_train_accuracy:.4f}")
print(f"Final Validation Accuracy: {final_val_accuracy:.4f}")
print(f"Final Training Loss: {final_train_loss:.4f}")
print(f"Final Validation Loss: {final_val_loss:.4f}")


bn_loss, bn_accuracy = bn_model.evaluate(test_generator, steps=800 //
batch_size)
print("Batch Normalization Model - Test Loss:", bn_loss)
print("Batch Normalization Model - Test Accuracy:", bn_accuracy)
```

```
Final Training Accuracy: 0.5638
Final Validation Accuracy: 0.0508
Final Training Loss: 1.8425
Final Validation Loss: 14.8973
```

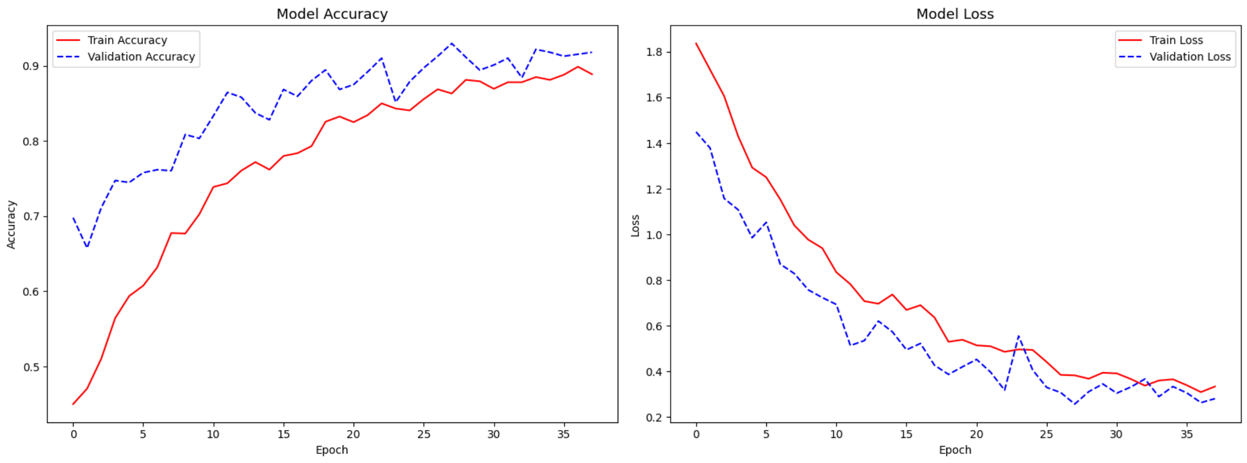
```
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 235ms/step - accuracy: 0.0069 - loss:
9.1953
Batch Normalization Model - Test Loss: 9.354384422302246
Batch Normalization Model - Test Accuracy: 0.0065104165114462376

# Train the Learning Rate and Dropout Model
history_lr_dropout = lr_dropout_model.fit(
    x=train_generator,
    steps_per_epoch=1600 // batch_size,
    epochs=epochs,
    validation_data=test_generator,
    validation_steps=800 // batch_size,
    callbacks=[checkpoint_lr_dropout, early_stop_lr_dropout]
)

Epoch 1/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 729ms/step - accuracy: 0.4360 - loss:
1.8765 - val_accuracy: 0.6979 - val_loss: 1.4491
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 657ms/step - accuracy: 0.4629 - loss:
1.7675 - val_accuracy: 0.6576 - val_loss: 1.3786
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 633ms/step - accuracy: 0.4831 - loss:
1.6620 - val_accuracy: 0.7109 - val_loss: 1.1578
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 597ms/step - accuracy: 0.5676 - loss:
1.4479 - val_accuracy: 0.7474 - val_loss: 1.1082
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 598ms/step - accuracy: 0.5992 - loss:
1.2889 - val_accuracy: 0.7448 - val_loss: 0.9857
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 627ms/step - accuracy: 0.6163 - loss:
1.2194 - val_accuracy: 0.7578 - val_loss: 1.0530
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 621ms/step - accuracy: 0.6190 - loss:
1.1668 - val_accuracy: 0.7617 - val_loss: 0.8701
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 633ms/step - accuracy: 0.6818 - loss:
1.0526 - val_accuracy: 0.7604 - val_loss: 0.8290
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 662ms/step - accuracy: 0.6808 - loss:
0.9771 - val_accuracy: 0.8086 - val_loss: 0.7572
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 644ms/step - accuracy: 0.7112 - loss:
0.9261 - val_accuracy: 0.8034 - val_loss: 0.7234
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 669ms/step - accuracy: 0.7375 - loss:
0.8374 - val_accuracy: 0.8333 - val_loss: 0.6933
Epoch 12/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 678ms/step - accuracy: 0.7474 - loss:
```

```
0.7785 - val_accuracy: 0.8646 - val_loss: 0.5132
Epoch 13/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 643ms/step - accuracy: 0.7643 - loss:
0.7036 - val_accuracy: 0.8581 - val_loss: 0.5351
Epoch 14/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 656ms/step - accuracy: 0.7733 - loss:
0.6941 - val_accuracy: 0.8372 - val_loss: 0.6205
Epoch 15/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 676ms/step - accuracy: 0.7628 - loss:
0.7335 - val_accuracy: 0.8281 - val_loss: 0.5734
Epoch 16/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 708ms/step - accuracy: 0.7795 - loss:
0.6694 - val_accuracy: 0.8685 - val_loss: 0.4946
Epoch 17/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 643ms/step - accuracy: 0.7715 - loss:
0.6904 - val_accuracy: 0.8594 - val_loss: 0.5224
Epoch 18/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 648ms/step - accuracy: 0.7736 - loss:
0.6570 - val_accuracy: 0.8802 - val_loss: 0.4269
Epoch 19/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 758ms/step - accuracy: 0.8234 - loss:
0.5267 - val_accuracy: 0.8945 - val_loss: 0.3869
Epoch 20/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 728ms/step - accuracy: 0.8214 - loss:
0.5637 - val_accuracy: 0.8685 - val_loss: 0.4203
Epoch 21/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 753ms/step - accuracy: 0.8348 - loss:
0.4779 - val_accuracy: 0.8750 - val_loss: 0.4527
Epoch 22/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 754ms/step - accuracy: 0.8338 - loss:
0.5117 - val_accuracy: 0.8919 - val_loss: 0.3967
Epoch 23/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 823ms/step - accuracy: 0.8549 - loss:
0.4923 - val_accuracy: 0.9102 - val_loss: 0.3177
Epoch 24/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 756ms/step - accuracy: 0.8487 - loss:
0.4899 - val_accuracy: 0.8516 - val_loss: 0.5551
Epoch 25/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 795ms/step - accuracy: 0.8282 - loss:
0.5090 - val_accuracy: 0.8789 - val_loss: 0.4061
Epoch 26/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 790ms/step - accuracy: 0.8456 - loss:
0.4416 - val_accuracy: 0.8971 - val_loss: 0.3299
Epoch 27/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.8825 - loss:
0.3597 - val_accuracy: 0.9128 - val_loss: 0.3073
Epoch 28/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 888ms/step - accuracy: 0.8598 - loss:
0.3801 - val_accuracy: 0.9297 - val_loss: 0.2569
```

```
Epoch 29/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 668ms/step - accuracy: 0.8766 - loss:
0.3679 - val_accuracy: 0.9115 - val_loss: 0.3112
Epoch 30/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 670ms/step - accuracy: 0.8893 - loss:
0.3698 - val_accuracy: 0.8942 - val_loss: 0.3451
Epoch 31/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 787ms/step - accuracy: 0.8572 - loss:
0.4376 - val_accuracy: 0.9010 - val_loss: 0.3048
Epoch 32/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 754ms/step - accuracy: 0.8837 - loss:
0.3672 - val_accuracy: 0.9102 - val_loss: 0.3315
Epoch 33/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 781ms/step - accuracy: 0.8835 - loss:
0.3297 - val_accuracy: 0.8841 - val_loss: 0.3667
Epoch 34/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 775ms/step - accuracy: 0.8847 - loss:
0.3652 - val_accuracy: 0.9219 - val_loss: 0.2896
Epoch 35/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 912ms/step - accuracy: 0.8887 - loss:
0.3383 - val_accuracy: 0.9180 - val_loss: 0.3337
Epoch 36/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 653ms/step - accuracy: 0.8852 - loss:
0.3400 - val_accuracy: 0.9128 - val_loss: 0.3055
Epoch 37/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 599ms/step - accuracy: 0.8896 - loss:
0.3411 - val_accuracy: 0.9154 - val_loss: 0.2632
Epoch 38/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 620ms/step - accuracy: 0.8837 - loss:
0.3421 - val_accuracy: 0.9180 - val_loss: 0.2806
```

```python
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

axs[0].plot(history_lr_dropout.history['accuracy'], label='Train
Accuracy', color='red')
axs[0].plot(history_lr_dropout.history['val_accuracy'],
label='Validation Accuracy', color='blue', linestyle='dashed',
markeredgecolor='purple', markeredgewidth=2)
axs[0].set_title('Model Accuracy', size=13)
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Accuracy')
axs[0].legend()

axs[1].plot(history_lr_dropout.history['loss'], label='Train Loss',
color='red')
axs[1].plot(history_lr_dropout.history['val_loss'], label='Validation
Loss', color='blue', linestyle='dashed', markeredgecolor='purple',
markeredgewidth=2)
axs[1].set_title('Model Loss', size=13)
axs[1].set_xlabel('Epoch')
```

```
axs[1].set_ylabel('Loss')
axs[1].legend()

plt.tight_layout()
plt.show()
```



```
final_train_accuracy = history_lr_dropout.history['accuracy'][-1]
final_val_accuracy = history_lr_dropout.history['val_accuracy'][-1]
final_train_loss = history_lr_dropout.history['loss'][-1]
final_val_loss = history_lr_dropout.history['val_loss'][-1]

print(f"Final Training Accuracy: {final_train_accuracy:.4f}")
print(f"Final Validation Accuracy: {final_val_accuracy:.4f}")
print(f"Final Training Loss: {final_train_loss:.4f}")
print(f"Final Validation Loss: {final_val_loss:.4f}")

lr_dropout_loss, lr_dropout_accuracy =
lr_dropout_model.evaluate(test_generator, steps=800 // batch_size)
print("Learning Rate and Dropout Model - Test Loss:", lr_dropout_loss)
print("Learning Rate and Dropout Model - Test Accuracy:",
lr_dropout_accuracy)

Final Training Accuracy: 0.8888
Final Validation Accuracy: 0.9180
Final Training Loss: 0.3339
Final Validation Loss: 0.2806
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 137ms/step - accuracy: 0.9161 - loss:
0.3143
Learning Rate and Dropout Model - Test Loss: 0.31704944372177124
Learning Rate and Dropout Model - Test Accuracy: 0.91015625
```

Testing

```
def load_and_preprocess_image(image_path):
    img = load_img(image_path, target_size=(100, 100))
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

def get_class_label(class_index, class_indices):
    labels = dict((v,k) for k,v in class_indices.items())
    return labels[class_index]

img_path =
'fruits-360_dataset/fruits-360/test-multiple_fruits/mangos1.jpg'
preprocessed_image = load_and_preprocess_image(img_path)
plt.imshow(np.squeeze(preprocessed_image))
plt.axis('off')
plt.show()
```

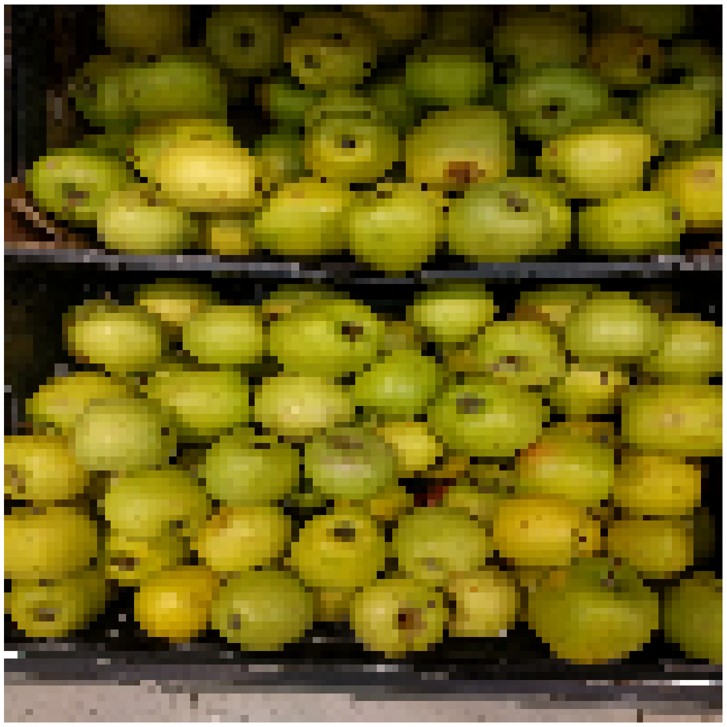
```
prediction = lr_dropout_model.predict(preprocessed_image)
predicted_class_index = np.argmax(prediction, axis=-1)[0]
print("Predicted class index:", predicted_class_index)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
Predicted class index: 14
```

```
predicted_class_label = get_class_label(predicted_class_index,
train_generator.class_indices)
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: Avocado
```

```
img_path =
'fruits-360_dataset/fruits-360/test-multiple_fruits/apple.jpg'
preprocessed_image = load_and_preprocess_image(img_path)
plt.imshow(np.squeeze(preprocessed_image))
plt.axis('off')
plt.show()
```



```
prediction = lr_dropout_model.predict(preprocessed_image)
predicted_class_index = np.argmax(prediction, axis=-1)[0]
print("Predicted class index:", predicted_class_index)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
Predicted class index: 12

predicted_class_label = get_class_label(predicted_class_index,
train_generator.class_indices)
print("Predicted class label:", predicted_class_label)

Predicted class label: Apple Red Yellow 2
```

```
image = test_path + "/Beetroot/23_100.jpg"

preprocessed_image = load_and_preprocess_image(image)
plt.imshow(np.squeeze(preprocessed_image))
plt.axis('off')
plt.show()
```

```
prediction = lr_dropout_model.predict(preprocessed_image)
predicted_class_index = np.argmax(prediction, axis=-1)[0]
print("Predicted class index:", predicted_class_index)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Predicted class index: 19
```

```
predicted_class_label = get_class_label(predicted_class_index,
train_generator.class_indices)
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: Beetroot
```

Own test pics (for fun)

```
image=  "Adam test imgs/ban.png"

preprocessed_image = load_and_preprocess_image(image)
```

```python
plt.imshow(np.squeeze(preprocessed_image))
plt.axis('off')
plt.show()
```

```python
prediction = lr_dropout_model.predict(preprocessed_image)
predicted_class_index = np.argmax(prediction, axis=-1)[0]
print("Predicted class index:", predicted_class_index)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
Predicted class index: 17
```

```python
predicted_class_label = get_class_label(predicted_class_index,
train_generator.class_indices)
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: Banana Lady Finger
```

```python
image = "Adam test imgs/shrimp.png"
```

```python
preprocessed_image = load_and_preprocess_image(image)
plt.imshow(np.squeeze(preprocessed_image))
plt.axis('off')
plt.show()
```

```
prediction = lr_dropout_model.predict(preprocessed_image)
predicted_class_index = np.argmax(prediction, axis=-1)[0]
print("Predicted class index:", predicted_class_index)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Predicted class index: 36
```

```
predicted_class_label = get_class_label(predicted_class_index,
train_generator.class_indices)
print("Predicted class label:", predicted_class_label)
```

```
Predicted class label: Corn Husk
```

Maybe its because shrimp isnt in the training set ¯_(ツ)_/¯

```
def create_improved_model(input_shape, number_of_classes):
    model = Sequential([
        Conv2D(16, (2, 2), padding='same', input_shape=input_shape),
        Activation('relu'),
        MaxPooling2D(pool_size=2),
        Conv2D(32, (2, 2), activation='relu', padding='same'),
        MaxPooling2D(pool_size=2),
        Conv2D(64, (2, 2), activation='relu', padding='same'),
        MaxPooling2D(pool_size=2),
        Conv2D(128, (2, 2), activation='relu', padding='same'),
        MaxPooling2D(pool_size=2),
        Dropout(0.3),
        Flatten(),
        Dense(150, activation='relu'),
        Dropout(0.4),
        Dense(number_of_classes, activation='softmax')
    ])
    return model

model = create_improved_model(input_shape=(100, 100, 3),
number_of_classes=number_of_class)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

print("\nBatch Normalization Model Summary:")
model.summary()
```

```
Batch Normalization Model Summary:

Model: "sequential_4"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d_12 (Conv2D)               │ (None, 100, 100, 16)   │
208 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ activation_16 (Activation)       │ (None, 100, 100, 16)   │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_12 (MaxPooling2D) │ (None, 50, 50, 16)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_13 (Conv2D)               │ (None, 50, 50, 32)     │
2,080 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_13 (MaxPooling2D) │ (None, 25, 25, 32)     │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_14 (Conv2D)               │ (None, 25, 25, 64)     │
8,256 │
```

```
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ max_pooling2d_14 (MaxPooling2D) │ (None, 12, 12, 64)     │
0 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ conv2d_15 (Conv2D)               │ (None, 12, 12, 128)    │
32,896 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ max_pooling2d_15 (MaxPooling2D) │ (None, 6, 6, 128)      │
0 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ dropout_6 (Dropout)              │ (None, 6, 6, 128)      │
0 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ flatten_4 (Flatten)              │ (None, 4608)           │
0 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ dense_8 (Dense)                  │ (None, 150)            │
691,350 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ dropout_7 (Dropout)              │ (None, 150)            │
0 │
├──────────────────────────────────┼────────────────────────┼────────
───────┤
│ dense_9 (Dense)                  │ (None, 131)            │
19,781 │
└──────────────────────────────────┴────────────────────────┴────────
───────┘

 Total params: 1,509,144 (5.76 MB)

 Trainable params: 754,571 (2.88 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 754,573 (2.88 MB)

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(100, 100),  # Ensure the target size is the same as
`input_shape`
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    directory=test_path,
    target_size=(100, 100),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 67692 images belonging to 131 classes.
Found 22688 images belonging to 131 classes.
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=1600 // 32,
    epochs=30,
    validation_data=test_generator,
    validation_steps=800 // 32
)
```

```
Epoch 1/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 405ms/step - accuracy: 0.0100 - loss:
4.8697 - val_accuracy: 0.0188 - val_loss: 4.8105
Epoch 2/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 363ms/step - accuracy: 0.0201 - loss:
4.7297 - val_accuracy: 0.0463 - val_loss: 4.3688
Epoch 3/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 361ms/step - accuracy: 0.0442 - loss:
4.4008 - val_accuracy: 0.1187 - val_loss: 3.7958
Epoch 4/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 16s 333ms/step - accuracy: 0.0830 - loss:
3.9222 - val_accuracy: 0.1750 - val_loss: 3.5075
Epoch 5/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 256ms/step - accuracy: 0.1137 - loss:
3.6508 - val_accuracy: 0.2788 - val_loss: 3.0553
Epoch 6/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 260ms/step - accuracy: 0.1511 - loss:
3.3337 - val_accuracy: 0.3438 - val_loss: 2.6870
Epoch 7/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 275ms/step - accuracy: 0.1811 - loss:
3.0400 - val_accuracy: 0.4162 - val_loss: 2.3219
```

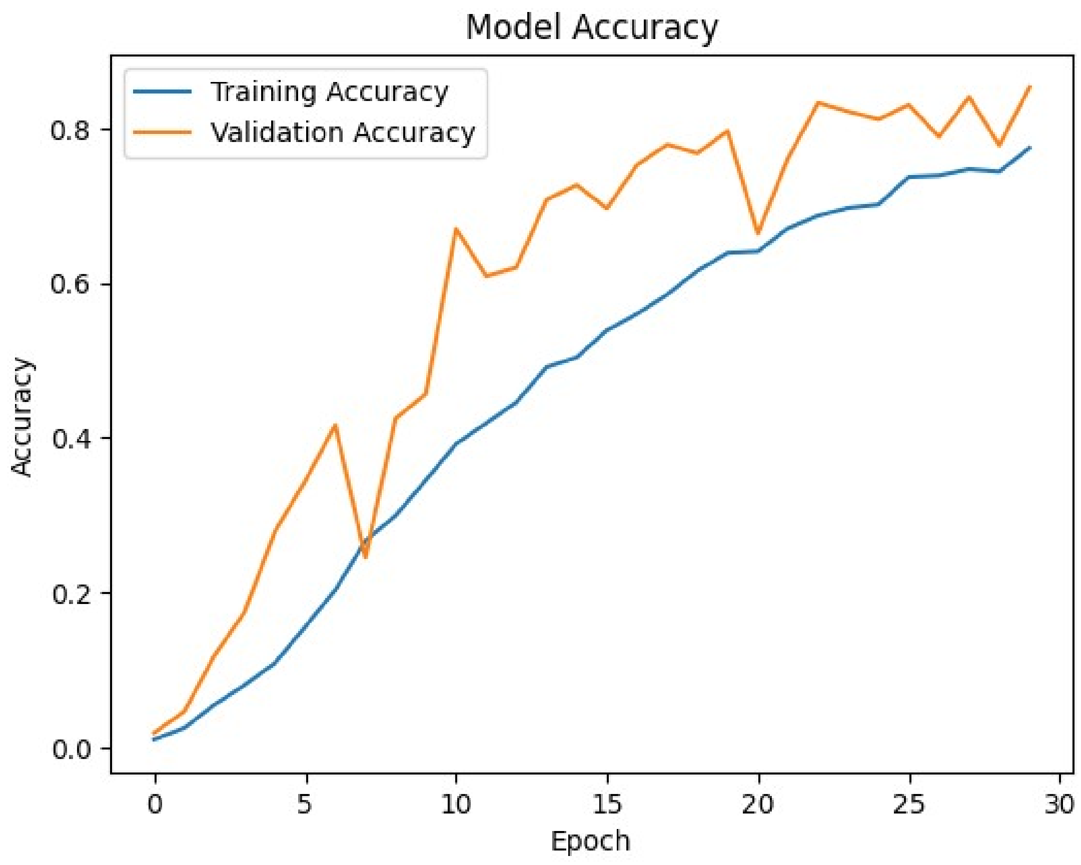
```
Epoch 8/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 265ms/step - accuracy: 0.2497 - loss:
2.7689 - val_accuracy: 0.2450 - val_loss: 2.6393
Epoch 9/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 265ms/step - accuracy: 0.2904 - loss:
2.5951 - val_accuracy: 0.4250 - val_loss: 2.0041
Epoch 10/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 255ms/step - accuracy: 0.3485 - loss:
2.3020 - val_accuracy: 0.4563 - val_loss: 1.8233
Epoch 11/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 274ms/step - accuracy: 0.3686 - loss:
2.1273 - val_accuracy: 0.6700 - val_loss: 1.3654
Epoch 12/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 266ms/step - accuracy: 0.4079 - loss:
2.0002 - val_accuracy: 0.6087 - val_loss: 1.4417
Epoch 13/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 258ms/step - accuracy: 0.4121 - loss:
1.9040 - val_accuracy: 0.6200 - val_loss: 1.2504
Epoch 14/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 246ms/step - accuracy: 0.4691 - loss:
1.7533 - val_accuracy: 0.7075 - val_loss: 1.0530
Epoch 15/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 232ms/step - accuracy: 0.5144 - loss:
1.6541 - val_accuracy: 0.7262 - val_loss: 1.0454
Epoch 16/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 261ms/step - accuracy: 0.5234 - loss:
1.4869 - val_accuracy: 0.6963 - val_loss: 1.0630
Epoch 17/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 230ms/step - accuracy: 0.5367 - loss:
1.4758 - val_accuracy: 0.7525 - val_loss: 0.9088
Epoch 18/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 216ms/step - accuracy: 0.5754 - loss:
1.2621 - val_accuracy: 0.7788 - val_loss: 0.8164
Epoch 19/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 215ms/step - accuracy: 0.6259 - loss:
1.2302 - val_accuracy: 0.7675 - val_loss: 0.8469
Epoch 20/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 242ms/step - accuracy: 0.6363 - loss:
1.0980 - val_accuracy: 0.7962 - val_loss: 0.7364
Epoch 21/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 226ms/step - accuracy: 0.6318 - loss:
1.1893 - val_accuracy: 0.6637 - val_loss: 1.0925
Epoch 22/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 220ms/step - accuracy: 0.6654 - loss:
1.1253 - val_accuracy: 0.7613 - val_loss: 0.7678
Epoch 23/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 220ms/step - accuracy: 0.6778 - loss:
1.0192 - val_accuracy: 0.8325 - val_loss: 0.5632
Epoch 24/30
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 10s 201ms/step - accuracy: 0.6976 - loss:
0.9627 - val_accuracy: 0.8213 - val_loss: 0.5988
Epoch 25/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 10s 201ms/step - accuracy: 0.7020 - loss:
0.9555 - val_accuracy: 0.8112 - val_loss: 0.6308
Epoch 26/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 10s 200ms/step - accuracy: 0.7252 - loss:
0.8156 - val_accuracy: 0.8300 - val_loss: 0.5948
Epoch 27/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 11s 231ms/step - accuracy: 0.7383 - loss:
0.8273 - val_accuracy: 0.7887 - val_loss: 0.6722
Epoch 28/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 10s 209ms/step - accuracy: 0.7360 - loss:
0.7699 - val_accuracy: 0.8400 - val_loss: 0.5895
Epoch 29/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 10s 201ms/step - accuracy: 0.7282 - loss:
0.8008 - val_accuracy: 0.7778 - val_loss: 0.6918
Epoch 30/30
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.7691 - loss:
0.6893 - val_accuracy: 0.8525 - val_loss: 0.5375
```

```
test_loss, test_accuracy = model.evaluate(test_generator)
print("Test Accuracy:", test_accuracy)
print("Test Loss:", test_loss)
```

```
709/709 ━━━━━━━━━━━━━━━━━━━━ 108s 152ms/step - accuracy: 0.8441 -
loss: 0.5282
Test Accuracy: 0.8421191573143005
Test Loss: 0.5355644226074219
```

```
plt.figure()
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



Redo Preprocessing

we restarting

```
from sklearn.datasets import load_files

train_dir = 'fruits-360_dataset/fruits-360/Training/'
test_dir = 'fruits-360_dataset/fruits-360/Test/'

def load_dataset(data_path):
    data_loading = load_files(data_path)
    files_add = np.array(data_loading['filenames'])
    targets_fruits = np.array(data_loading['target'])
    target_labels_fruits = np.array(data_loading['target_names'])
    return files_add,targets_fruits,target_labels_fruits

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)

from tensorflow.keras.utils import to_categorical

no_of_classes = len(np.unique(y_train))
```

```
y_train = to_categorical(y_train, num_classes=no_of_classes)
y_test = to_categorical(y_test, num_classes=no_of_classes)

print(y_train[0])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
x_test,x_valid = x_test[7000:],x_test[:7000]
y_test,y_valid = y_test[7000:],y_test[:7000]
print('Valiation X : ',x_valid.shape)
print('Validation y :',y_valid.shape)
print('Test X : ',x_test.shape)
print('Test y : ',y_test.shape)
```

```
Vaildation X :  (7000,)
Vaildation y : (7000, 131)
Test X :  (15688,)
Test y :  (15688, 131)
```

```
def convert_image_to_array_form(files):
    images_array=[]
    for file in files:
        images_array.append(img_to_array(load_img(file)))
    return images_array

x_train = np.array(convert_image_to_array_form(x_train))
print('Training set shape : ',x_train.shape)

x_valid = np.array(convert_image_to_array_form(x_valid))
print('Validation set shape : ',x_valid.shape)

x_test = np.array(convert_image_to_array_form(x_test))
print('Test set shape : ',x_test.shape)

print('1st training image shape ',x_train[0].shape)
```

```
Training set shape :  (67692, 100, 100, 3)
Validation set shape :  (7000, 100, 100, 3)
Test set shape :  (15688, 100, 100, 3)
1st training image shape  (100, 100, 3)
```

```
# Tryna reduce training time
x_train = x_train.astype('float32')/255
x_valid = x_valid.astype('float32')/255
x_test = x_test.astype('float32')/255
```

Now lets try this again

```
def tensorflow_based_model():
    model = Sequential()
    model.add(Conv2D(filters=16, kernel_size=2, input_shape=(100, 100,
3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=2))

    model.add(Conv2D(filters=32, kernel_size=2, activation='relu',
padding='same'))
    model.add(MaxPooling2D(pool_size=2))

    model.add(Conv2D(filters=64, kernel_size=2, activation='relu',
padding='same'))
    model.add(MaxPooling2D(pool_size=2))

    model.add(Conv2D(filters=128, kernel_size=2, activation='relu',
padding='same'))
    model.add(MaxPooling2D(pool_size=2))

    model.add(Dropout(0.3))
    model.add(Flatten())

    model.add(Dense(150))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))

    model.add(Dense(no_of_classes, activation='softmax'))

    return model

model = tensorflow_based_model()
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])
```

```
c:\Users\adamn\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(
```

```
bn_model = create_bn_model()
lr_dropout_model = create_lr_dropout_model()
```

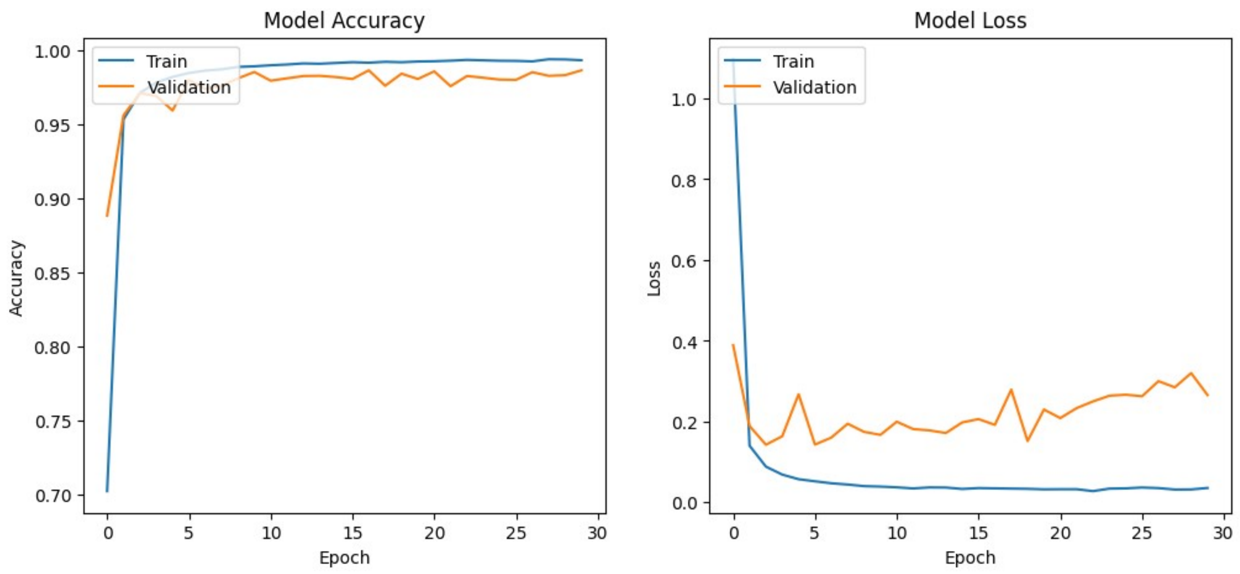
```
print("\nBatch Normalization Model Summary:")
bn_model.summary()
```

```
Batch Normalization Model Summary:

Model: "sequential_6"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d_20 (Conv2D)              │ (None, 98, 98, 32)     │
896 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ activation_19 (Activation)      │ (None, 98, 98, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_3           │ (None, 98, 98, 32)     │
128 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_20 (MaxPooling2D) │ (None, 49, 49, 32)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_21 (Conv2D)              │ (None, 47, 47, 64)     │
18,496 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ activation_20 (Activation)      │ (None, 47, 47, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_4           │ (None, 47, 47, 64)     │
256 │
│ (BatchNormalization)            │                        │
│
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_21 (MaxPooling2D) │ (None, 23, 23, 64)     │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
```

```
│ conv2d_22 (Conv2D)                │ (None, 21, 21, 128)       │
73,856 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ activation_21 (Activation)        │ (None, 21, 21, 128)       │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ batch_normalization_5             │ (None, 21, 21, 128)       │
512 │
│ (BatchNormalization)              │                           │
│
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ max_pooling2d_22 (MaxPooling2D) │ (None, 10, 10, 128)       │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ flatten_6 (Flatten)               │ (None, 12800)             │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dense_12 (Dense)                  │ (None, 1024)              │
13,108,224 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dropout_10 (Dropout)              │ (None, 1024)              │
0 │
├───────────────────────────────────┼───────────────────────────┼──────────
───────┤
│ dense_13 (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)

history = model.fit(x_train,y_train,
        batch_size = 32,
        epochs=30,
        validation_data=(x_valid, y_valid),
        verbose=2, shuffle=True)

Epoch 1/30
2116/2116 - 112s - 53ms/step - accuracy: 0.7025 - loss: 1.0969 -
val_accuracy: 0.8886 - val_loss: 0.3891
```

```
Epoch 2/30
2116/2116 - 100s - 47ms/step - accuracy: 0.9535 - loss: 0.1405 -
val_accuracy: 0.9563 - val_loss: 0.1892
Epoch 3/30
2116/2116 - 93s - 44ms/step - accuracy: 0.9714 - loss: 0.0886 -
val_accuracy: 0.9713 - val_loss: 0.1428
Epoch 4/30
2116/2116 - 82s - 39ms/step - accuracy: 0.9784 - loss: 0.0689 -
val_accuracy: 0.9694 - val_loss: 0.1639
Epoch 5/30
2116/2116 - 96s - 45ms/step - accuracy: 0.9823 - loss: 0.0575 -
val_accuracy: 0.9596 - val_loss: 0.2677
Epoch 6/30
2116/2116 - 90s - 42ms/step - accuracy: 0.9849 - loss: 0.0523 -
val_accuracy: 0.9801 - val_loss: 0.1432
Epoch 7/30
2116/2116 - 93s - 44ms/step - accuracy: 0.9865 - loss: 0.0474 -
val_accuracy: 0.9741 - val_loss: 0.1606
Epoch 8/30
2116/2116 - 91s - 43ms/step - accuracy: 0.9874 - loss: 0.0443 -
val_accuracy: 0.9764 - val_loss: 0.1948
Epoch 9/30
2116/2116 - 88s - 41ms/step - accuracy: 0.9890 - loss: 0.0404 -
val_accuracy: 0.9813 - val_loss: 0.1747
Epoch 10/30
2116/2116 - 91s - 43ms/step - accuracy: 0.9894 - loss: 0.0393 -
val_accuracy: 0.9856 - val_loss: 0.1673
Epoch 11/30
2116/2116 - 88s - 42ms/step - accuracy: 0.9901 - loss: 0.0375 -
val_accuracy: 0.9797 - val_loss: 0.1998
Epoch 12/30
2116/2116 - 106s - 50ms/step - accuracy: 0.9906 - loss: 0.0346 -
val_accuracy: 0.9813 - val_loss: 0.1816
Epoch 13/30
2116/2116 - 130s - 62ms/step - accuracy: 0.9914 - loss: 0.0372 -
val_accuracy: 0.9829 - val_loss: 0.1783
Epoch 14/30
2116/2116 - 105s - 50ms/step - accuracy: 0.9911 - loss: 0.0369 -
val_accuracy: 0.9830 - val_loss: 0.1718
Epoch 15/30
2116/2116 - 120s - 57ms/step - accuracy: 0.9917 - loss: 0.0334 -
val_accuracy: 0.9821 - val_loss: 0.1979
Epoch 16/30
2116/2116 - 115s - 54ms/step - accuracy: 0.9922 - loss: 0.0355 -
val_accuracy: 0.9809 - val_loss: 0.2063
Epoch 17/30
2116/2116 - 115s - 54ms/step - accuracy: 0.9919 - loss: 0.0348 -
val_accuracy: 0.9867 - val_loss: 0.1920
Epoch 18/30
```

```
2116/2116 - 113s - 54ms/step - accuracy: 0.9925 - loss: 0.0342 -
val_accuracy: 0.9763 - val_loss: 0.2792
Epoch 19/30
2116/2116 - 116s - 55ms/step - accuracy: 0.9922 - loss: 0.0337 -
val_accuracy: 0.9844 - val_loss: 0.1519
Epoch 20/30
2116/2116 - 116s - 55ms/step - accuracy: 0.9927 - loss: 0.0325 -
val_accuracy: 0.9809 - val_loss: 0.2300
Epoch 21/30
2116/2116 - 116s - 55ms/step - accuracy: 0.9928 - loss: 0.0327 -
val_accuracy: 0.9860 - val_loss: 0.2087
Epoch 22/30
2116/2116 - 104s - 49ms/step - accuracy: 0.9932 - loss: 0.0328 -
val_accuracy: 0.9760 - val_loss: 0.2334
Epoch 23/30
2116/2116 - 102s - 48ms/step - accuracy: 0.9938 - loss: 0.0279 -
val_accuracy: 0.9829 - val_loss: 0.2501
Epoch 24/30
2116/2116 - 143s - 67ms/step - accuracy: 0.9935 - loss: 0.0342 -
val_accuracy: 0.9817 - val_loss: 0.2641
Epoch 25/30
2116/2116 - 107s - 51ms/step - accuracy: 0.9932 - loss: 0.0348 -
val_accuracy: 0.9804 - val_loss: 0.2666
Epoch 26/30
2116/2116 - 114s - 54ms/step - accuracy: 0.9931 - loss: 0.0369 -
val_accuracy: 0.9803 - val_loss: 0.2628
Epoch 27/30
2116/2116 - 119s - 56ms/step - accuracy: 0.9927 - loss: 0.0355 -
val_accuracy: 0.9854 - val_loss: 0.3001
Epoch 28/30
2116/2116 - 107s - 51ms/step - accuracy: 0.9942 - loss: 0.0320 -
val_accuracy: 0.9830 - val_loss: 0.2849
Epoch 29/30
2116/2116 - 111s - 52ms/step - accuracy: 0.9941 - loss: 0.0322 -
val_accuracy: 0.9834 - val_loss: 0.3201
Epoch 30/30
2116/2116 - 116s - 55ms/step - accuracy: 0.9935 - loss: 0.0358 -
val_accuracy: 0.9867 - val_loss: 0.2658

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```



```
acc_score = model.evaluate(x_test, y_test)
print("Test Loss:", acc_score[0])
print("Test Accuracy:", acc_score[1])

491/491 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9876 - loss: 
0.2267
Test Loss: 0.24676187336444855
Test Accuracy: 0.9871239066123962

jupyter nbconvert --to html FinalProject.ipynb

  Cell In[1], line 1
    jupyter nbconvert --to html FinalProject.ipynb
            ^
SyntaxError: invalid syntax
```