This notebook will classify fruits from the fruit 360 dataset. The dataset can be found at https://www.kaggle.com/datasets/moltean/fruits.

This dataset originally contained 131 classes of fruits and vegetables but I selected 16 of them for this notebook.

It is important to note that the data was already split into train and test and had the same image size.

The model should be able to classify the fruits into the correct category.

Importing packages

In [ ]:
```
import numpy as np
import os
import cv2
import tensorflow as tf
from tqdm import tqdm

from sklearn.metrics import confusion_matrix
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.utils import shuffle

import pandas as pd
```

Loading the data

In [ ]:
```
#names of classes/folders in the file
names = ['Apple Braeburn', 'Banana', 'Blueberry', 'Cherry', 'Grape White', 'Kiv
#labels for each class/folder
namesLabel = {names:i for i, names in enumerate(names)}
#set size to (100,100)
SIZE = (100, 100)
```

In [ ]:
```
%%capture
#load data from drive
from google.colab import drive
drive.mount('/content/gdrive')
!unzip gdrive/MyDrive/fruit.zip;
#!cp '/content/drive/MyDrive/fruit 360 2.zip' <data>
```

```
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_543_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
492_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_492_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
482_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_482_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
476_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_476_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/28
9_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._289_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/29
9_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._299_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
241_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_241_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/35
8_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._358_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
251_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_251_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
329_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_329_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
127_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_127_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/15
6_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._156_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/r_
137_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._r_137_100.jpg
  inflating: fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grape Blue/14
6_100.jpg
  inflating: __MACOSX/fruit 360/fruits-360_dataset/fruits-360/Test/Cherry/Grap
e Blue/._146_100.jpg
```

In [ ]:

```
#confirm that the classes got labelled correctly
namesLabel
```

Out[ ]:
```
{'Apple Braeburn': 0,
 'Banana': 1,
 'Blueberry': 2,
 'Cherry': 3,
 'Grape White': 4,
 'Kiwi': 5,
 'Lemon': 6,
 'Mango': 7,
 'Orange': 8,
 'Pear': 9,
 'Pineapple': 10,
 'Plum': 11,
 'Pomegranate': 12,
 'Strawberry': 13,
 'Tomato': 14,
 'Watermelon': 15}
```

In [ ]:
```
# Define a function to load the data
def load_data():

    train_dir = "/content/fruit 360/fruits-360_dataset/fruits-360/Train/"
    test_dir = "/content/fruit 360/fruits-360_dataset/fruits-360/Test/"
    image_size = 64

    output = []

    # Iterate through training and test sets
    for data in [train_dir, test_dir]:

        images = []
        imageLabels = []

        print("Loading {}".format(data))

        # Iterate through each folder corresponding to a category
        for folder in os.listdir(data):
            Label = namesLabel[folder]

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(data, folder))):

                # Get the path name of the image
                image_path = os.path.join(os.path.join(data, folder), file)

                # Open the img
                image = cv2.imread(image_path)

                # Check if image is valid
                if image is None or image.size == 0:
                    continue

                # Resize and convert the img
                image = cv2.resize(image, (image_size, image_size))
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # convert BGR t

                # Append the image and its corresponding label to the output
                images.append(image)
                imageLabels.append(Label)

        # Convert images and imageLabels
```

```
        images = np.array(images, dtype = 'float32')
        imageLabels = np.array(imageLabels, dtype = 'int32')

        output.append((images, imageLabels))

    return output
```

This is an extra step that I had to perform becasue I was getting an error called DS_Store which means that there are hidden files that are not useful for python programs and can cause them to crash.

In [ ]:
```
!find /content/fruit\ 360/fruits-360_dataset/fruits-360/Train -name .DS_Store -
!find /content/fruit\ 360/fruits-360_dataset/fruits-360/Test -name .DS_Store -c
```

In [ ]:
```
#Loading training and testing images and their corresponding labels.
(ImagesForTraining, ImageLabelsForTraining), (ImagesForTesting, ImageLabelsForT
```

```
Loading /content/fruit 360/fruits-360_dataset/fruits-360/Train/
100%|██████████| 492/492 [00:00<00:00, 3405.45it/s]
100%|██████████| 475/475 [00:00<00:00, 4152.39it/s]
100%|██████████| 466/466 [00:00<00:00, 4490.96it/s]
100%|██████████| 490/490 [00:00<00:00, 4224.32it/s]
100%|██████████| 492/492 [00:00<00:00, 4504.96it/s]
100%|██████████| 492/492 [00:00<00:00, 3185.80it/s]
100%|██████████| 492/492 [00:00<00:00, 2491.38it/s]
100%|██████████| 462/462 [00:00<00:00, 2817.68it/s]
100%|██████████| 479/479 [00:00<00:00, 2774.64it/s]
100%|██████████| 738/738 [00:00<00:00, 2854.29it/s]
100%|██████████| 447/447 [00:00<00:00, 2845.59it/s]
100%|██████████| 492/492 [00:00<00:00, 2901.13it/s]
100%|██████████| 490/490 [00:00<00:00, 2526.17it/s]
100%|██████████| 490/490 [00:00<00:00, 3036.70it/s]
100%|██████████| 490/490 [00:00<00:00, 3368.15it/s]
100%|██████████| 492/492 [00:00<00:00, 2831.51it/s]
Loading /content/fruit 360/fruits-360_dataset/fruits-360/Test/
100%|██████████| 164/164 [00:00<00:00, 2829.77it/s]
100%|██████████| 157/157 [00:00<00:00, 2315.66it/s]
100%|██████████| 156/156 [00:00<00:00, 1614.23it/s]
100%|██████████| 166/166 [00:00<00:00, 1806.91it/s]
100%|██████████| 164/164 [00:00<00:00, 2470.16it/s]
100%|██████████| 165/165 [00:00<00:00, 2483.18it/s]
100%|██████████| 164/164 [00:00<00:00, 2466.33it/s]
100%|██████████| 154/154 [00:00<00:00, 2515.79it/s]
100%|██████████| 160/160 [00:00<00:00, 2650.70it/s]
100%|██████████| 246/246 [00:00<00:00, 2747.29it/s]
100%|██████████| 151/151 [00:00<00:00, 2438.03it/s]
100%|██████████| 164/164 [00:00<00:00, 2540.58it/s]
100%|██████████| 166/166 [00:00<00:00, 2687.17it/s]
100%|██████████| 166/166 [00:00<00:00, 2885.01it/s]
100%|██████████| 166/166 [00:00<00:00, 3183.49it/s]
100%|██████████| 164/164 [00:00<00:00, 2632.50it/s]
```

In [ ]:
```
#lets check the size of train and test
numTrain = ImageLabelsForTraining.shape[0]
numTest = ImageLabelsForTesting.shape[0]

print ("There are {} training images".format(numTrain))
```

```
print ("There are {} testing images".format(numTest))
print ("The size of each image is: {}".format(SIZE))
```

```
There are 7979 training images
There are 2672 testing images
The size of each image is: (100, 100)
```

I also normalized the data since I read that it was good practice

In [ ]:
```
# Normalize the data
ImagesForTraining = ImagesForTraining / 255.0
ImagesForTesting = ImagesForTesting / 255.0

ImagesForTraining
ImagesForTesting
```

Out[ ]:
```
array([[[[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        ...,

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]]],


       [[[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
```

```
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 ...,
 [1.        , 1.        , 0.99607843],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ]],

[[1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 ...,
 [1.        , 1.        , 0.99215686],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ]],

...,

[[1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 ...,
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ]],

[[1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 ...,
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ]],

[[1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 ...,
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ],
 [1.        , 1.        , 1.        ]]],


[[[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [0.99607843, 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 0.99215686],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 0.99215686],
```

```
  [1.        , 1.        , 0.9882353 ],
  ...,
  [1.        , 1.        , 0.99215686],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 ...,

 [[0.99607843, 1.        , 0.99607843],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]]],


...,


[[[1.        , 1.        , 0.99215686],
  [0.99607843, 1.        , 1.        ],
  [0.99215686, 1.        , 0.99607843],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [0.99607843, 0.99607843, 0.99607843],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[0.99607843, 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [0.99607843, 0.99607843, 0.99607843],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],
```

```
        ...,

        [[0.9882353 , 1.        , 0.99215686],
         [1.        , 1.        , 1.        ],
         [1.        , 0.99215686, 1.        ],
         ...,
         [1.        , 0.99607843, 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]]],


       [[[0.99607843, 1.        , 0.99215686],
         [1.        , 0.99215686, 0.99607843],
         [1.        , 0.99215686, 0.99607843],
         ...,
         [0.99215686, 1.        , 0.99607843],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.972549  , 1.        , 0.99215686],
         [0.99215686, 1.        , 0.99607843],
         [1.        , 0.99607843, 1.        ],
         ...,
         [0.99607843, 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.972549  , 1.        , 0.99607843],
         [0.9882353 , 1.        , 0.99607843],
         [1.        , 0.99215686, 0.99607843],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        ...,

        [[0.9882353 , 1.        , 0.99215686],
         [1.        , 1.        , 1.        ],
         [1.        , 0.99215686, 1.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],
```

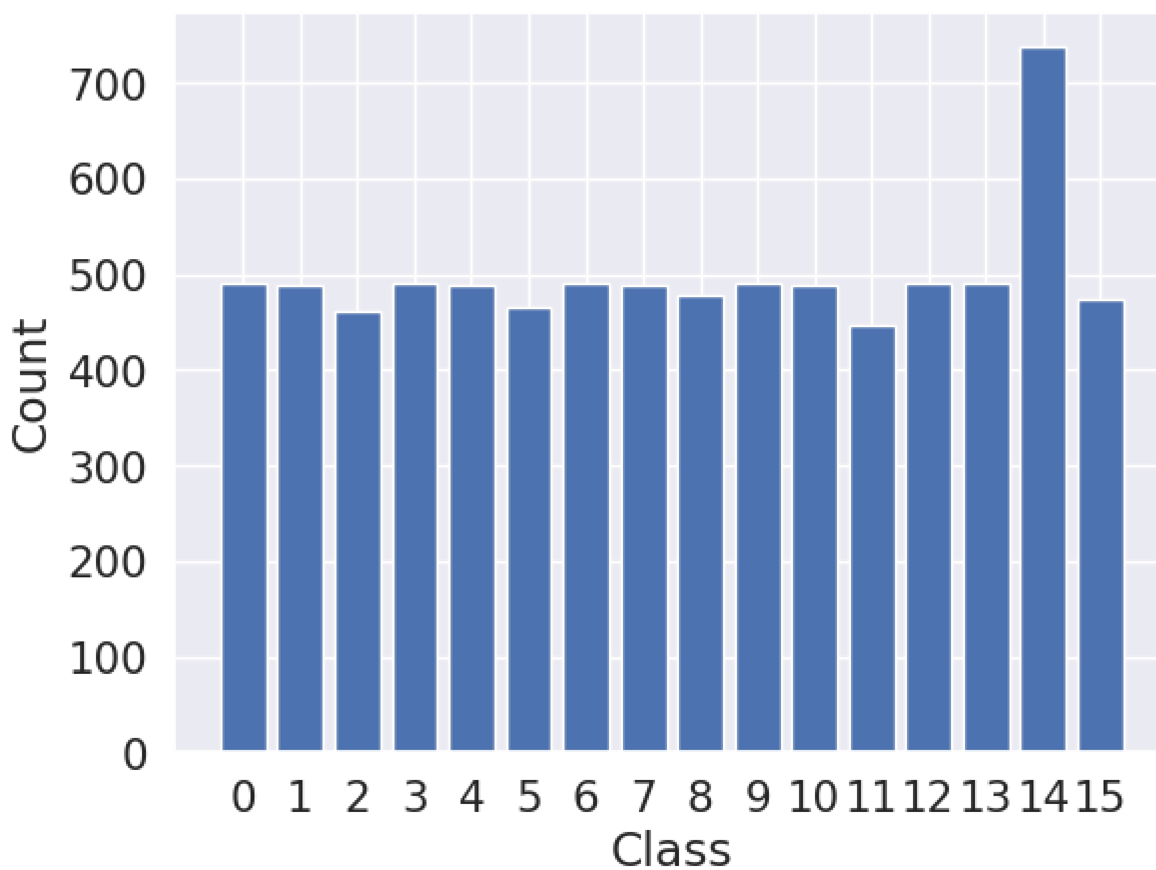
```
 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]]],


[[[0.99215686, 1.        , 0.99215686],
  [0.99607843, 0.99215686, 0.99607843],
  [1.        , 0.99215686, 0.99607843],
  ...,
  [0.99607843, 0.99607843, 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[0.96862745, 1.        , 0.99215686],
  [0.9843137 , 1.        , 0.99607843],
  [0.99607843, 1.        , 0.99607843],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[0.9843137 , 1.        , 0.99607843],
  [0.9843137 , 1.        , 0.99607843],
  [0.99215686, 1.        , 0.99607843],
  ...,
  [0.99607843, 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 ...,

 [[0.9882353 , 1.        , 0.99215686],
  [1.        , 1.        , 0.99607843],
  [1.        , 0.99215686, 0.99215686],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],
```

```
[[1.         , 1.         , 1.         ],
 [1.         , 1.         , 1.         ],
 [1.         , 1.         , 1.         ],
 ...,
 [1.         , 1.         , 1.         ],
 [1.         , 1.         , 1.         ],
 [1.         , 1.         , 1.         ]]]], dtype=float32)
```

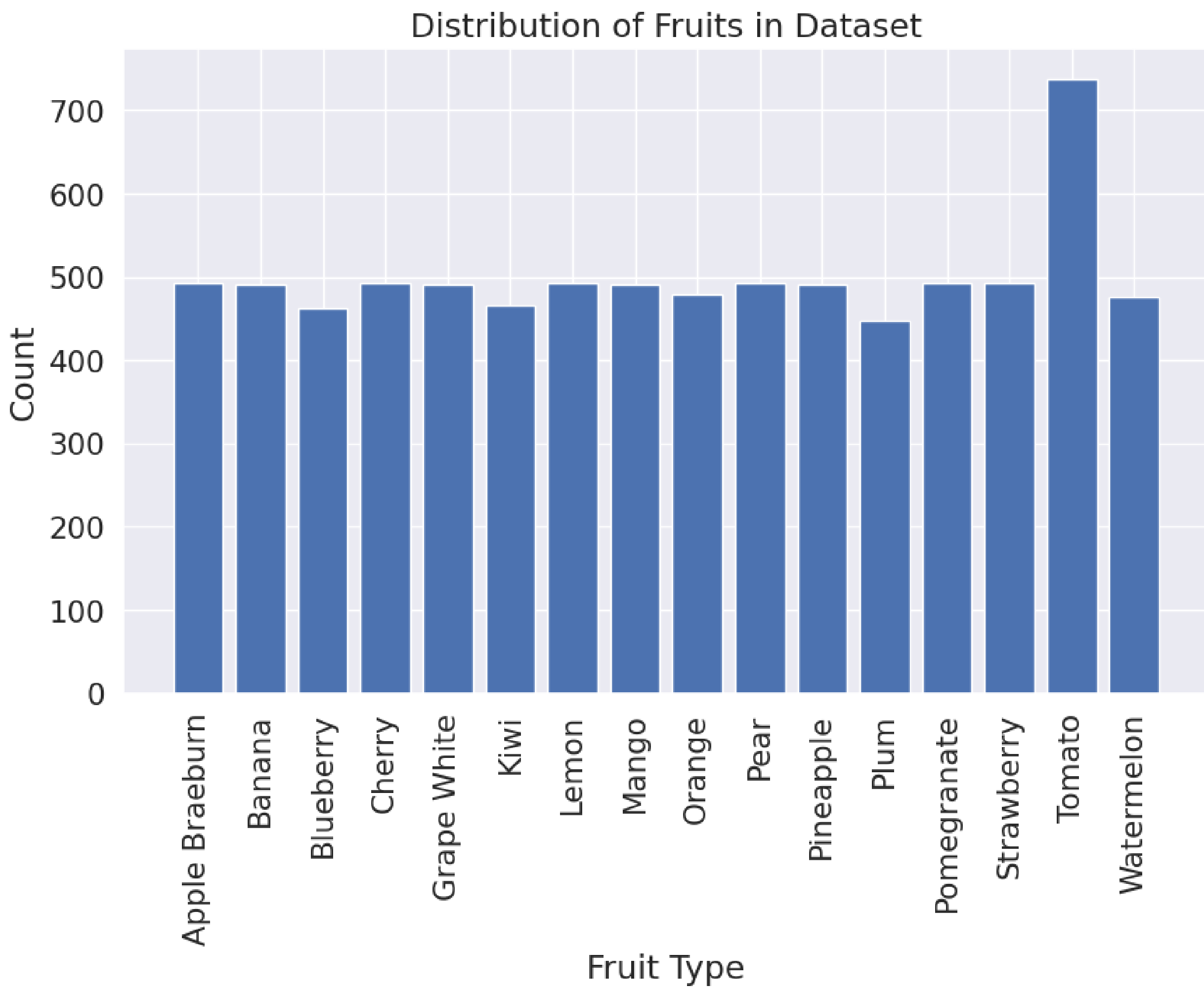
Data visualization

Next we create graphs showing the distribution of target classes. I had to do this twice because I felt that the first graph didn't give enough information. The second graph shows us that 15 of the fruits had a count between 450 and 500 however tomatoes had a significantly higher count of about 750.

In [ ]:
```
import matplotlib.pyplot as plt

# Count the number of images for each class in the training set
class_counts = np.bincount(ImageLabelsForTraining)

# Create a bar chart
x = np.arange(len(class_counts))
plt.bar(x, class_counts)
plt.xticks(x)
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



In [ ]:
```
# Create a graph showing the distribution of the target classes
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(names, class_counts)
ax.set_xlabel("Fruit Type")
ax.set_ylabel("Count")
ax.set_title("Distribution of Fruits in Dataset")

# Rotate the x-axis labels vertically
plt.xticks(rotation='vertical')

plt.show()
```



Sequential model

I will now create a sequential model.

In [ ]:
```
num_classes=len(names)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(64, 64, 3)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
```

```
    tf.keras.layers.Dense(num_classes, activation='softmax'),
])
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 12288)             0         
                                                                 
 dense (Dense)               (None, 512)               6291968   
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 512)               262656    
                                                                 
 dropout_1 (Dropout)         (None, 512)               0         
                                                                 
 dense_2 (Dense)             (None, 16)                8208      
                                                                 
=================================================================
Total params: 6,562,832
Trainable params: 6,562,832
Non-trainable params: 0
_________________________________________________________________
```

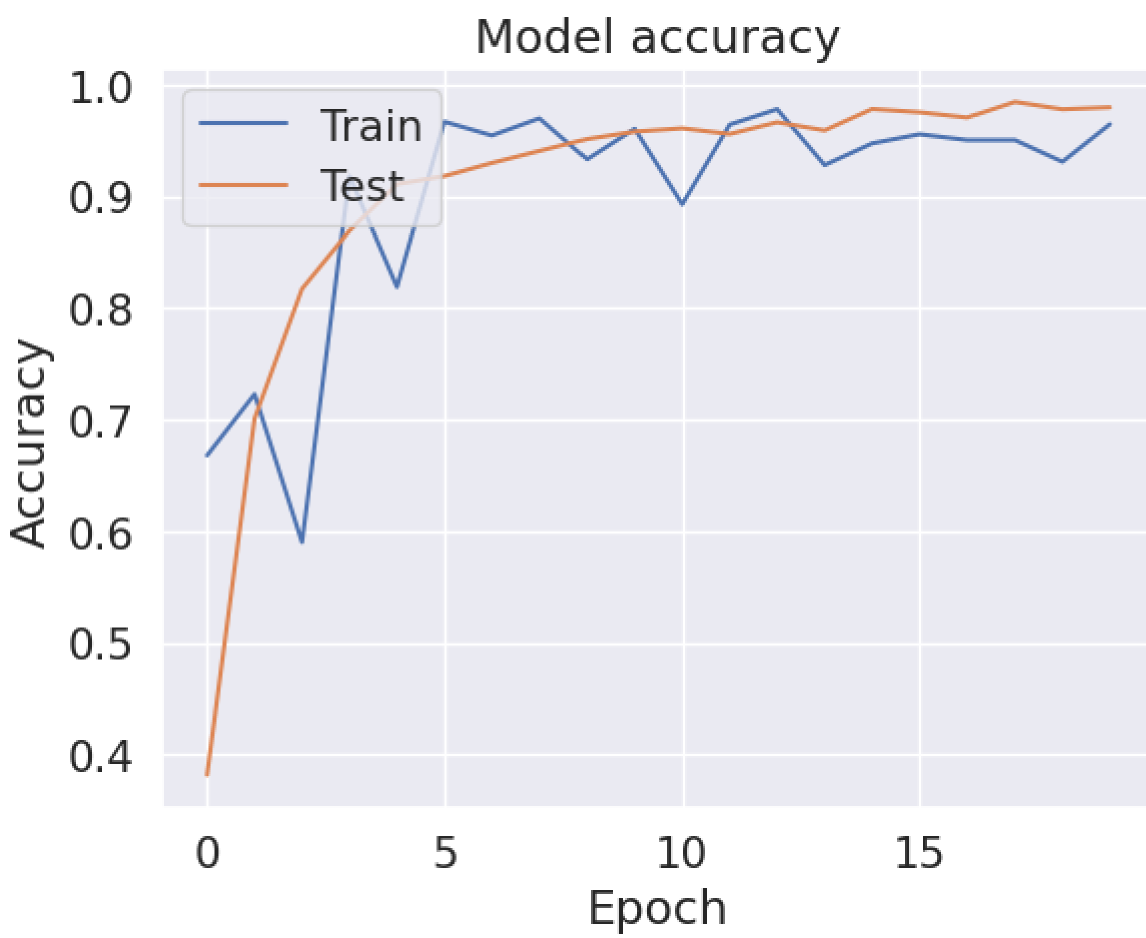
In [ ]:
```
from tensorflow.keras.utils import to_categorical

# one-hot encode the labels
num_classes = 16
y_train = to_categorical(ImageLabelsForTraining, num_classes)
y_test = to_categorical(ImageLabelsForTesting, num_classes)

# train the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(ImagesForTraining, y_train,
                    batch_size=128,
                    epochs=20,
                    verbose=1,
                    validation_data=(ImagesForTesting, y_test))
```

```
Epoch 1/20
63/63 [==============================] - 6s 16ms/step - loss: 3.5711 - accurac
y: 0.3819 - val_loss: 0.9658 - val_accuracy: 0.6680
Epoch 2/20
63/63 [==============================] - 1s 10ms/step - loss: 0.8544 - accurac
y: 0.7011 - val_loss: 0.8006 - val_accuracy: 0.7231
Epoch 3/20
63/63 [==============================] - 1s 11ms/step - loss: 0.5580 - accurac
y: 0.8175 - val_loss: 1.6763 - val_accuracy: 0.5902
Epoch 4/20
63/63 [==============================] - 1s 14ms/step - loss: 0.4623 - accurac
y: 0.8695 - val_loss: 0.1984 - val_accuracy: 0.9184
Epoch 5/20
63/63 [==============================] - 1s 12ms/step - loss: 0.3437 - accurac
y: 0.9110 - val_loss: 0.8611 - val_accuracy: 0.8192
Epoch 6/20
63/63 [==============================] - 1s 12ms/step - loss: 0.3098 - accurac
y: 0.9183 - val_loss: 0.0827 - val_accuracy: 0.9671
Epoch 7/20
63/63 [==============================] - 1s 13ms/step - loss: 0.2843 - accurac
y: 0.9301 - val_loss: 0.1452 - val_accuracy: 0.9547
Epoch 8/20
63/63 [==============================] - 1s 12ms/step - loss: 0.2460 - accurac
y: 0.9408 - val_loss: 0.1219 - val_accuracy: 0.9701
Epoch 9/20
63/63 [==============================] - 1s 11ms/step - loss: 0.2102 - accurac
y: 0.9515 - val_loss: 0.2877 - val_accuracy: 0.9334
Epoch 10/20
63/63 [==============================] - 1s 9ms/step - loss: 0.1816 - accurac
y: 0.9581 - val_loss: 0.1206 - val_accuracy: 0.9607
Epoch 11/20
63/63 [==============================] - 1s 10ms/step - loss: 0.1665 - accurac
y: 0.9611 - val_loss: 0.4632 - val_accuracy: 0.8930
Epoch 12/20
63/63 [==============================] - 1s 10ms/step - loss: 0.1688 - accurac
y: 0.9560 - val_loss: 0.1496 - val_accuracy: 0.9644
Epoch 13/20
63/63 [==============================] - 1s 10ms/step - loss: 0.1333 - accurac
y: 0.9665 - val_loss: 0.0959 - val_accuracy: 0.9783
Epoch 14/20
63/63 [==============================] - 1s 10ms/step - loss: 0.1715 - accurac
y: 0.9593 - val_loss: 0.2463 - val_accuracy: 0.9281
Epoch 15/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0842 - accurac
y: 0.9784 - val_loss: 0.1389 - val_accuracy: 0.9476
Epoch 16/20
63/63 [==============================] - 1s 10ms/step - loss: 0.0834 - accurac
y: 0.9756 - val_loss: 0.2033 - val_accuracy: 0.9558
Epoch 17/20
63/63 [==============================] - 1s 9ms/step - loss: 0.1209 - accurac
y: 0.9708 - val_loss: 0.2448 - val_accuracy: 0.9506
Epoch 18/20
63/63 [==============================] - 1s 10ms/step - loss: 0.0499 - accurac
y: 0.9848 - val_loss: 0.1525 - val_accuracy: 0.9506
Epoch 19/20
63/63 [==============================] - 1s 10ms/step - loss: 0.0850 - accurac
y: 0.9782 - val_loss: 0.3193 - val_accuracy: 0.9311
Epoch 20/20
63/63 [==============================] - 1s 10ms/step - loss: 0.0818 - accurac
y: 0.9799 - val_loss: 0.1240 - val_accuracy: 0.9648
```

In [ ]:
```
history.history.keys()
```

Out[ ]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
score = model.evaluate(ImagesForTesting, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.12395673990249634
Test accuracy: 0.964820384979248
```

CNN

In [ ]:
```
batch_size = 128
num_classes = 16
epochs = 20
```

In [ ]:
```
#Loading training and testing images and their corresponding labels.
(ImagesForTraining, ImageLabelsForTraining), (ImagesForTesting, ImageLabelsForT
```

```
Loading /content/fruit 360/fruits-360_dataset/fruits-360/Train/
100%|██████████| 492/492 [00:00<00:00, 2723.33it/s]
100%|██████████| 475/475 [00:00<00:00, 2758.61it/s]
100%|██████████| 466/466 [00:00<00:00, 2587.19it/s]
100%|██████████| 490/490 [00:00<00:00, 2765.42it/s]
100%|██████████| 492/492 [00:00<00:00, 2828.66it/s]
100%|██████████| 492/492 [00:00<00:00, 3013.37it/s]
100%|██████████| 492/492 [00:00<00:00, 2748.89it/s]
100%|██████████| 462/462 [00:00<00:00, 3103.15it/s]
100%|██████████| 479/479 [00:00<00:00, 2639.95it/s]
100%|██████████| 738/738 [00:00<00:00, 2931.69it/s]
100%|██████████| 447/447 [00:00<00:00, 2819.33it/s]
100%|██████████| 492/492 [00:00<00:00, 1859.28it/s]
100%|██████████| 490/490 [00:00<00:00, 1458.71it/s]
100%|██████████| 490/490 [00:00<00:00, 1972.48it/s]
100%|██████████| 490/490 [00:00<00:00, 2636.56it/s]
100%|██████████| 492/492 [00:00<00:00, 4516.08it/s]
Loading /content/fruit 360/fruits-360_dataset/fruits-360/Test/
100%|██████████| 164/164 [00:00<00:00, 4297.68it/s]
100%|██████████| 157/157 [00:00<00:00, 4284.13it/s]
100%|██████████| 156/156 [00:00<00:00, 4692.96it/s]
100%|██████████| 166/166 [00:00<00:00, 5084.30it/s]
100%|██████████| 164/164 [00:00<00:00, 4672.02it/s]
100%|██████████| 165/165 [00:00<00:00, 4946.75it/s]
100%|██████████| 164/164 [00:00<00:00, 3520.31it/s]
100%|██████████| 154/154 [00:00<00:00, 1205.81it/s]
100%|██████████| 160/160 [00:00<00:00, 873.57it/s]
100%|██████████| 246/246 [00:00<00:00, 1316.27it/s]
100%|██████████| 151/151 [00:00<00:00, 4586.53it/s]
100%|██████████| 164/164 [00:00<00:00, 5185.18it/s]
100%|██████████| 166/166 [00:00<00:00, 4448.68it/s]
100%|██████████| 166/166 [00:00<00:00, 5124.15it/s]
100%|██████████| 166/166 [00:00<00:00, 5827.27it/s]
100%|██████████| 164/164 [00:00<00:00, 4327.72it/s]
```

In [ ]:
```
model = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(64, 64, 3)),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(16, activation="softmax"),
    ]
)
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 12544)             0         
                                                                 
 dropout_2 (Dropout)         (None, 12544)             0         
                                                                 
 dense_3 (Dense)             (None, 16)                200720    
                                                                 
=================================================================
Total params: 220,112
Trainable params: 220,112
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', met
```

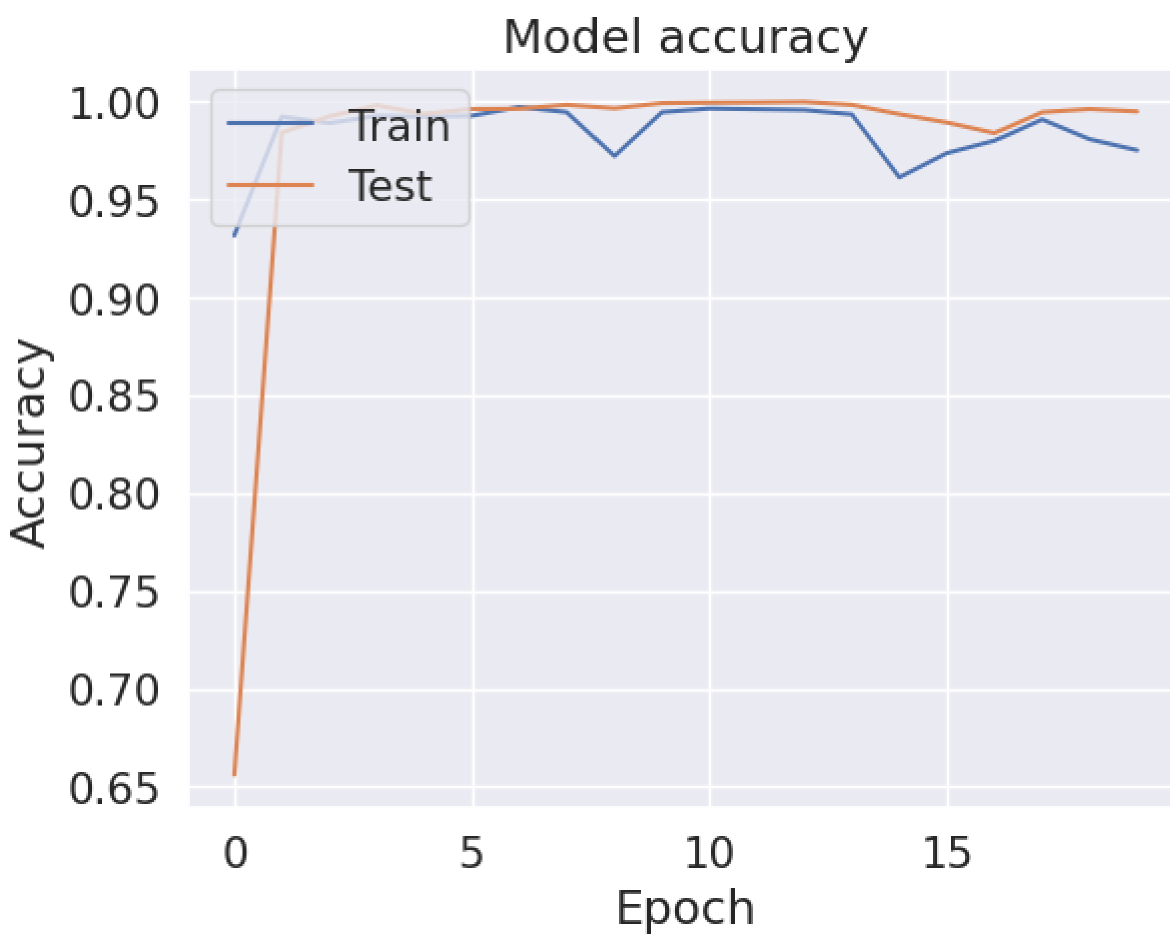
In [ ]:

```
history = model.fit(
    ImagesForTraining, ImageLabelsForTraining,
    batch_size=128,
    epochs=20,
    verbose=1,
    validation_data=(ImagesForTesting, ImageLabelsForTesting)
)
```

```
Epoch 1/20
63/63 [==============================] - 7s 25ms/step - loss: 14.4420 - accura
cy: 0.6561 - val_loss: 0.2019 - val_accuracy: 0.9315
Epoch 2/20
63/63 [==============================] - 1s 17ms/step - loss: 0.0502 - accurac
y: 0.9840 - val_loss: 0.0309 - val_accuracy: 0.9921
Epoch 3/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0256 - accurac
y: 0.9922 - val_loss: 0.0359 - val_accuracy: 0.9888
Epoch 4/20
63/63 [==============================] - 1s 20ms/step - loss: 0.0072 - accurac
y: 0.9979 - val_loss: 0.0195 - val_accuracy: 0.9925
Epoch 5/20
63/63 [==============================] - 1s 17ms/step - loss: 0.0214 - accurac
y: 0.9935 - val_loss: 0.0204 - val_accuracy: 0.9921
Epoch 6/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0124 - accurac
y: 0.9960 - val_loss: 0.0222 - val_accuracy: 0.9925
Epoch 7/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0126 - accurac
y: 0.9961 - val_loss: 0.0083 - val_accuracy: 0.9970
Epoch 8/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0081 - accurac
y: 0.9981 - val_loss: 0.0207 - val_accuracy: 0.9944
Epoch 9/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0111 - accurac
y: 0.9964 - val_loss: 0.2288 - val_accuracy: 0.9719
Epoch 10/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0030 - accurac
y: 0.9990 - val_loss: 0.0109 - val_accuracy: 0.9944
Epoch 11/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0035 - accurac
y: 0.9992 - val_loss: 0.0087 - val_accuracy: 0.9963
Epoch 12/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0014 - accurac
y: 0.9994 - val_loss: 0.0153 - val_accuracy: 0.9959
Epoch 13/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0017 - accurac
y: 0.9996 - val_loss: 0.0120 - val_accuracy: 0.9955
Epoch 14/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0051 - accurac
y: 0.9981 - val_loss: 0.0204 - val_accuracy: 0.9933
Epoch 15/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0238 - accurac
y: 0.9934 - val_loss: 0.1523 - val_accuracy: 0.9611
Epoch 16/20
63/63 [==============================] - 1s 17ms/step - loss: 0.0356 - accurac
y: 0.9891 - val_loss: 0.2192 - val_accuracy: 0.9734
Epoch 17/20
63/63 [==============================] - 1s 18ms/step - loss: 0.0698 - accurac
y: 0.9837 - val_loss: 0.1992 - val_accuracy: 0.9798
Epoch 18/20
63/63 [==============================] - 1s 17ms/step - loss: 0.0203 - accurac
y: 0.9944 - val_loss: 0.0229 - val_accuracy: 0.9906
Epoch 19/20
63/63 [==============================] - 1s 16ms/step - loss: 0.0198 - accurac
y: 0.9960 - val_loss: 0.0812 - val_accuracy: 0.9805
Epoch 20/20
63/63 [==============================] - 1s 15ms/step - loss: 0.0302 - accurac
y: 0.9947 - val_loss: 0.2466 - val_accuracy: 0.9749
```

In [ ]:
```
history.history.keys()
```
Out[ ]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



In [ ]:
```
score = model.evaluate(ImagesForTesting, ImageLabelsForTesting, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```
```
Test loss: 0.24655233323574066
Test accuracy: 0.97492516040802
```

Performance analysis of different approaches

In this project, I classified different types of fruits. As mentioned previously in the notebook the dataset originally contained 131 classes but i narrowed this down to 16. I used

TensorFlow and Keras to perform deep learning. I used a sequential model and CNN. 99%

The sequential model had a few layers. The input shape was 64 by 64 by 3(the 3 refers to 3 color channels) and the dense layers had 512 nodes each. The outer layer has a softmax activation function. The hidden layers have a relu activation function. It also uses the categorical_crossentropy loss function to compile the model.

The Convolutional Neural Network also uses the same input layer of 64x64x3 and the 3 refers to the 3 color channels. The first convolutional layer has 32 filters and uses the relu activation function. It extracts 32 different 3x3 features from the input image. The first max pooling layer reduces the spatial dimensions of the output from the convolutional layer by taking the maximum value within a 2x2 window. The second convolutional layer has 64 filters. The second max pooling layer further reduces the spatial dimensions. The flatten layer flattens the output from the second max pooling layer into a 1D vector, which is then passed to a fully connected layer.

The two neural networks had different accuracies. The first model had an accuracy of 96% and the second had an accuracy of 98%. I consider these accuracies to be quite high.

There are several reasons why CNNs can perform better than sequential models. CNNs are designed to take into account the spatial structure of input data. Also, it is able to detect local features like edges and textures. They are also able to detect pattens in an image regardless of position. They are also able to learn a small number of parameters and apply them across the entire image. This is called parameter sharing and reduces the risk of overfitting.