FRUIT CLASSIFICATION MODEL

ASSIGNMENT TWO

GROUP MEMBERS

STEP ONE (IMPORTING LIBRARIES, DATASET && INSPECTING THE DATASET

- ARINJUNA SARAH **2020/BCS/093/PS**

STEP TWO (PREPROCESSING DATA && SPLITTING THE DATA

- OKWAKUNDA GLORIA **2020/BCS/091/PS**
- KAYINZA MARIAM **2020/BCS/036/PS**

STEP THREE (DATA STANDARDIZATION, ARGUMENTATION, CACHING && SHUFFLING)

- AINAMANI CHRISTIAN **2020/BCS/001**
- MAYANJA ROBERT **2020/BCS/043/PS**

STEP FOUR (Building the Model, ARCHITECTURE && TRAINING)

- WESONGA BOB **2020/BCS/092/PS**
- SIEMBA ERNEST OOKO **2020/BCS/005**

STEP FIVE (EVALUATION && EXPORTING THE MODEL

- ATWANZIRE TIMOTHY IAN **2020/BCS/026/PS**

- KATUSHABE MOREEN **2020**/**BCS**/**034**/**PS**

https://www.kaggle.com/datasets/arnavmehta710a/fids30 (https://www.kaggle.com/datasets/arnavmehta710a/fids30) OR https://www.vicos.si/resources/fids30 (https://www.vicos.si/resources/fids30)

Importing Libraries

ARINJUNA SARAH

In [1]:

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
import os
```

```
2023-04-03 10:53:31.945909: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
```

Set of constants

- Batch Size - A number of samples processed before the model is updated
- Image Size - The Resolution
- Channels - Number of steps of the shifting of a convolutional filter over an input image
- Epochs - Total number of iterations of all the training data in one cycle for training the machine learning model
- Data_dir - Path of the dataset

In [2]:

```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=30
EPOCHS=30
data_dir = "FIDS30"
```

Importing the Dataset

- We will https://www.vicos.si/resources/fids30 (https://www.vicos.si/resources/fids30) data which can also be got from https://www.kaggle.com/datasets/arnavmehta710a/fids30 (https://www.kaggle.com/datasets/arnavmehta710a/fids30)
- We will use the "tf.keras.preprocessing.image_dataset_from_directory" inbuild func to load

the dataset

- It takes in
  - "seeds" @para for defining a starting point for the sequence
  - "shuffle" @para to aim to mix up data and can optionally retain logical relationships between columns
  - image_size && batch_size

In [3]:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 971 files belonging to 30 classes.

2023-04-03 10:53:36.872155: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
2023-04-03 10:53:36.874447: I tensorflow/core/common_runtime/process_u
til.cc:146] Creating new thread pool with default inter op setting: 2.
Tune using inter_op_parallelism_threads for best performance.
```

ClassNames - These are classes of our dataset

```
In [4]: class_names = dataset.class_names
        print(len(class_names))
        class_names

        30

Out[4]: ['acerolas',
         'apples',
         'apricots',
         'avocados',
         'bananas',
         'blackberries',
         'blueberries',
         'cantaloupes',
         'cherries',
         'coconuts',
         'figs',
         'grapefruits',
         'grapes',
         'guava',
         'kiwifruit',
         'lemons',
         'limes',
         'mangos',
         'olives',
         'oranges',
         'passionfruit',
         'peaches',
         'pears',
         'pineapples',
         'plums',
         'pomegranates',
         'raspberries',
         'strawberries',
         'tomatoes',
         'watermelons']
```

Preprocess the Dataset

Kayinza Mariam, Okwakunda Gloria & Katushabe Moreen

The shape of the first batch of images in a dataset

- As you can see above, each element in the dataset is a tuple. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels

```
In [5]: for image_batch, labels_batch in dataset.take(1):
            print(image_batch.shape)
            print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[ 5 11 27 21 24 19  3 22  6 14 11 19 17 24 11 21 22 12 21 17  4 22  5
25
  1  8 25  6  4 21 23 18]
```

Visualizing the Images

```
In [6]: plt.figure(figsize=(10, 10))
        for image_batch, labels_batch in dataset.take(1):
            for i in range(12):
                ax = plt.subplot(3, 4, i + 1)
                plt.imshow(image_batch[i].numpy().astype("uint8"))
                plt.title(class_names[labels_batch[i]])
                plt.axis("off")
```

Splitting the dataset

- It's good practice to use a validation split when developing your model.
- We will use 80% of the images for training and 20% for validation.

```
In [7]: len(dataset)
```

```
Out[7]: 31
```

```
In [8]: train_size = 0.8
        len(dataset)*train_size
```

```
Out[8]: 24.8
```

```
In [9]: train_ds = dataset.take(24)
        len(train_ds)
```

```
Out[9]: 24
```

```
In [10]: test_ds = dataset.skip(24)
         len(test_ds)
```

```
Out[10]: 7
```

```
In [11]: val_size=0.1
         len(dataset)*val_size
```

```
Out[11]: 3.1
```

```
In [12]: val_ds = test_ds.take(3)
         len(val_ds)
```

```
Out[12]: 3
```

```
In [13]: test_ds = test_ds.skip(3)
         len(test_ds)
```

```
Out[13]: 4
```

In [14]:

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

In [15]:

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [16]:

```
len(train_ds)
```

Out[16]:

```
24
```

In [17]:

```
len(val_ds)
```

Out[17]:

```
3
```

In [18]:

```
len(test_ds)
```

Out[18]:

```
4
```

DATA STANDARDIZATION, ARGUMENTATION, CACHING && SHUFFLING

Ainamani Christian && MAYANJA ROBERT

Cache, Shuffle, and Prefetch the Dataset

```
- "Cache" a module for language modelling which stores previous
hidden states in memory cells
- "Shuffle" a module to mixing up dataset
- "Prefetch" overlaps the preprocessing and model execution of
a training step.
```

In [19]:

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTO
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AU
```

Creating a Layer for Resizing and Normalization

In [20]:

```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

Data Augmentation

In [21]:

```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Applying Data Augmentation to Train Dataset

In [22]:

```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
WARNING:tensorflow:From /home/fedora/anaconda3/lib/python3.10/site-pac
kages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83:
Analyzer.lamba_check (from tensorflow.python.autograph.pyct.static_ana
lysis.liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement wh
ere they are used, or at least in the same block. https://github.com/t
ensorflow/tensorflow/issues/56089 (https://github.com/tensorflow/tenso
rflow/issues/56089)
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip ca
use there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause the
re is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause the
re is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUn
iformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTr
ansformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip ca
use there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause the
re is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause the
re is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUn
iformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTr
ansformV3 cause there is no registered converter for this op.
```

Building the Model

Wesonga Bob & Siemba Ernest Ooko

Model Architecture

In [23]:

```
num_classes = 30

model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(num_classes)
])
```

In [24]:

```
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

In [25]:

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=EPOCHS
)
```

```
Epoch 1/30

2023-04-03 10:53:54.937934: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:54:01.410207: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 17
of 10000
2023-04-03 10:54:01.412893: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:54:02.864622: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:54:03.299995: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.
2023-04-03 10:54:03.300049: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 1 o
f 1000
2023-04-03 10:54:03.300140: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.

24/24 [==============================] - ETA: 0s - loss: 3.3086 - accu
racy: 0.0768

2023-04-03 10:55:37.312737: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:55:42.202548: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:55:43.407733: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 10:55:43.667746: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 23
of 10000
2023-04-03 10:55:43.744196: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.
2023-04-03 10:55:43.744436: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 1 o
f 1000
2023-04-03 10:55:43.744552: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.

24/24 [==============================] - 117s 4s/step - loss: 3.3086 -
accuracy: 0.0768 - val_loss: 3.1436 - val_accuracy: 0.1042
Epoch 2/30
24/24 [==============================] - 91s 4s/step - loss: 2.7993 -
accuracy: 0.1706 - val_loss: 2.5101 - val_accuracy: 0.2083
Epoch 3/30
24/24 [==============================] - 90s 4s/step - loss: 2.3504 -
accuracy: 0.2461 - val_loss: 2.6380 - val_accuracy: 0.1771
Epoch 4/30
24/24 [==============================] - 92s 4s/step - loss: 2.1902 -
accuracy: 0.2878 - val_loss: 2.3530 - val_accuracy: 0.2188
```

```
Epoch 5/30
24/24 [==============================] - 90s 4s/step - loss: 1.9695 -
accuracy: 0.3529 - val_loss: 2.3430 - val_accuracy: 0.2292
Epoch 6/30
24/24 [==============================] - 91s 4s/step - loss: 1.8728 -
accuracy: 0.4102 - val_loss: 2.1488 - val_accuracy: 0.3438
Epoch 7/30
24/24 [==============================] - 88s 4s/step - loss: 1.7951 -
accuracy: 0.4089 - val_loss: 2.2537 - val_accuracy: 0.3125
Epoch 8/30
24/24 [==============================] - 89s 4s/step - loss: 1.5407 -
accuracy: 0.4935 - val_loss: 2.1685 - val_accuracy: 0.3021
Epoch 9/30
24/24 [==============================] - 88s 4s/step - loss: 1.5102 -
accuracy: 0.4857 - val_loss: 2.0344 - val_accuracy: 0.4167
Epoch 10/30
24/24 [==============================] - 90s 4s/step - loss: 1.3796 -
accuracy: 0.5495 - val_loss: 1.8846 - val_accuracy: 0.4375
Epoch 11/30
24/24 [==============================] - 90s 4s/step - loss: 1.3137 -
accuracy: 0.5664 - val_loss: 1.7198 - val_accuracy: 0.5104
Epoch 12/30
24/24 [==============================] - 89s 4s/step - loss: 1.1813 -
accuracy: 0.6263 - val_loss: 2.1887 - val_accuracy: 0.3854
Epoch 13/30
24/24 [==============================] - 90s 4s/step - loss: 1.0463 -
accuracy: 0.6536 - val_loss: 2.2246 - val_accuracy: 0.3958
Epoch 14/30
24/24 [==============================] - 89s 4s/step - loss: 1.0228 -
accuracy: 0.6693 - val_loss: 2.0088 - val_accuracy: 0.3646
Epoch 15/30
24/24 [==============================] - 88s 4s/step - loss: 0.9698 -
accuracy: 0.6810 - val_loss: 1.8004 - val_accuracy: 0.4479
Epoch 16/30
24/24 [==============================] - 89s 4s/step - loss: 0.9047 -
accuracy: 0.7148 - val_loss: 1.6077 - val_accuracy: 0.5208
Epoch 17/30
24/24 [==============================] - 88s 4s/step - loss: 0.8362 -
accuracy: 0.7383 - val_loss: 1.6431 - val_accuracy: 0.5208
Epoch 18/30
24/24 [==============================] - 90s 4s/step - loss: 0.7006 -
accuracy: 0.7656 - val_loss: 1.7465 - val_accuracy: 0.4896
Epoch 19/30
24/24 [==============================] - 90s 4s/step - loss: 0.6939 -
accuracy: 0.7786 - val_loss: 1.7895 - val_accuracy: 0.4792
Epoch 20/30
24/24 [==============================] - 88s 4s/step - loss: 0.6325 -
accuracy: 0.7747 - val_loss: 1.8770 - val_accuracy: 0.5312
Epoch 21/30
24/24 [==============================] - 90s 4s/step - loss: 0.6489 -
accuracy: 0.7839 - val_loss: 1.9455 - val_accuracy: 0.5104
Epoch 22/30
24/24 [==============================] - 89s 4s/step - loss: 0.5746 -
accuracy: 0.8164 - val_loss: 1.9166 - val_accuracy: 0.5312
Epoch 23/30
24/24 [==============================] - 89s 4s/step - loss: 0.5663 -
```

```
accuracy: 0.8177 - val_loss: 1.7853 - val_accuracy: 0.5312
Epoch 24/30
24/24 [==============================] - 90s 4s/step - loss: 0.5305 -
accuracy: 0.8125 - val_loss: 1.7998 - val_accuracy: 0.5000
Epoch 25/30
24/24 [==============================] - 95s 4s/step - loss: 0.4274 -
accuracy: 0.8620 - val_loss: 1.9640 - val_accuracy: 0.5104
Epoch 26/30
24/24 [==============================] - 112s 5s/step - loss: 0.4280 -
accuracy: 0.8620 - val_loss: 2.0358 - val_accuracy: 0.5208
Epoch 27/30
24/24 [==============================] - 95s 4s/step - loss: 0.4456 -
accuracy: 0.8594 - val_loss: 1.7088 - val_accuracy: 0.5938
Epoch 28/30
24/24 [==============================] - 90s 4s/step - loss: 0.3672 -
accuracy: 0.8841 - val_loss: 1.8127 - val_accuracy: 0.5208
Epoch 29/30
24/24 [==============================] - 105s 4s/step - loss: 0.3988 -
accuracy: 0.8633 - val_loss: 2.0113 - val_accuracy: 0.5417
Epoch 30/30
24/24 [==============================] - 91s 4s/step - loss: 0.3796 -
```

Evaluation

In [26]:
```
scores = model.evaluate(test_ds)
```

```
2023-04-03 11:41:02.114465: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 11:41:07.002962: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 11:41:08.190928: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 21
of 10000
2023-04-03 11:41:08.220648: W tensorflow/core/lib/png/png_io.cc:88] PN
G warning: iCCP: known incorrect sRGB profile
2023-04-03 11:41:08.521201: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.
2023-04-03 11:41:08.521285: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:392] Filling up shuffle buffer (this may take a while): 1 o
f 1000
2023-04-03 11:41:08.521357: I tensorflow/core/kernels/data/shuffle_dat
aset_op.cc:417] Shuffle buffer filled.

4/4 [==============================] - 15s 1s/step - loss: 2.1635 - ac
curacy: 0.5469
```

In [27]:
```
# Scores is just a list containing loss and accuracy value
scores
```

Out[27]:
```
[2.1634774208068848, 0.546875]
```

Plotting the Accuracy and Loss Curves

Atwanzire Timothy Ian && KATUSHABE MOREEN

```
In [28]: history
```

```
Out[28]: <keras.callbacks.History at 0x7f2c84420e50>
```

```
In [31]: history.params
```

```
Out[31]: {'verbose': 1, 'epochs': 30, 'steps': 24}
```

```
In [32]: history.history.keys()
```

```
Out[32]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

loss, accuracy, val loss etc are a python list containing values of loss, accuracy etc at the end of each epoch

```
In [33]: type(history.history['loss'])
```

```
Out[33]: list
```

```
In [34]: len(history.history['loss'])
```

```
Out[34]: 30
```

```
In [35]: history.history['loss'][:5] # show loss for first 5 epochs
```

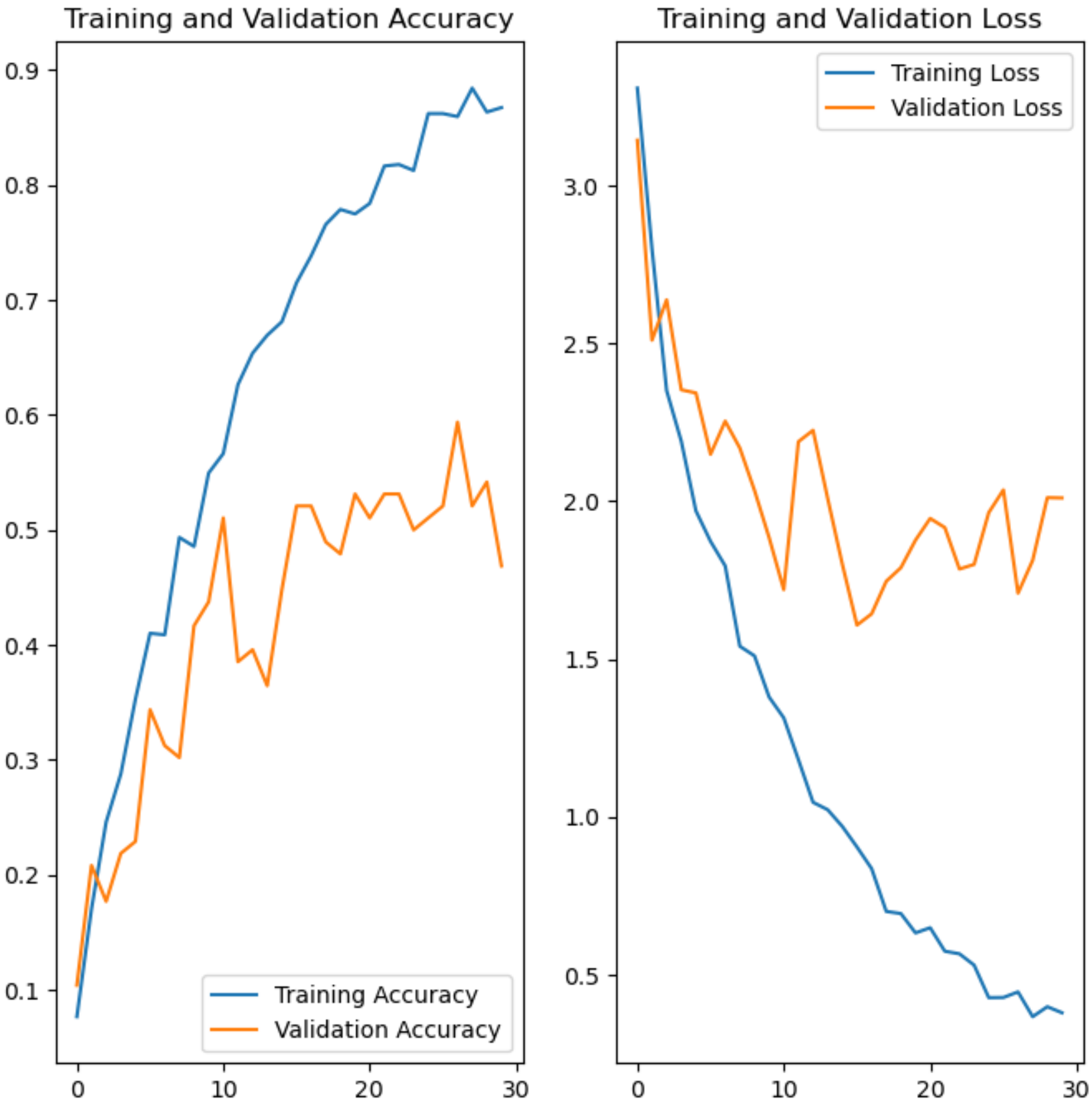
```
Out[35]: [3.3086416721343994,
          2.7993218898773193,
          2.350374460220337,
          2.1901614665985107,
          1.969476342201233]
```

```
In [36]: acc = history.history['accuracy']
         val_acc = history.history['val_accuracy']

         loss = history.history['loss']
         val_loss = history.history['val_loss']
```

```
In [37]: plt.figure(figsize=(8, 8))
         plt.subplot(1, 2, 1)
         plt.plot(range(EPOCHS), acc, label='Training Accuracy')
         plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
         plt.legend(loc='lower right')
         plt.title('Training and Validation Accuracy')

         plt.subplot(1, 2, 2)
         plt.plot(range(EPOCHS), loss, label='Training Loss')
         plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
         plt.legend(loc='upper right')
         plt.title('Training and Validation Loss')
         plt.show()
```



Run prediction on a sample image

In [38]:

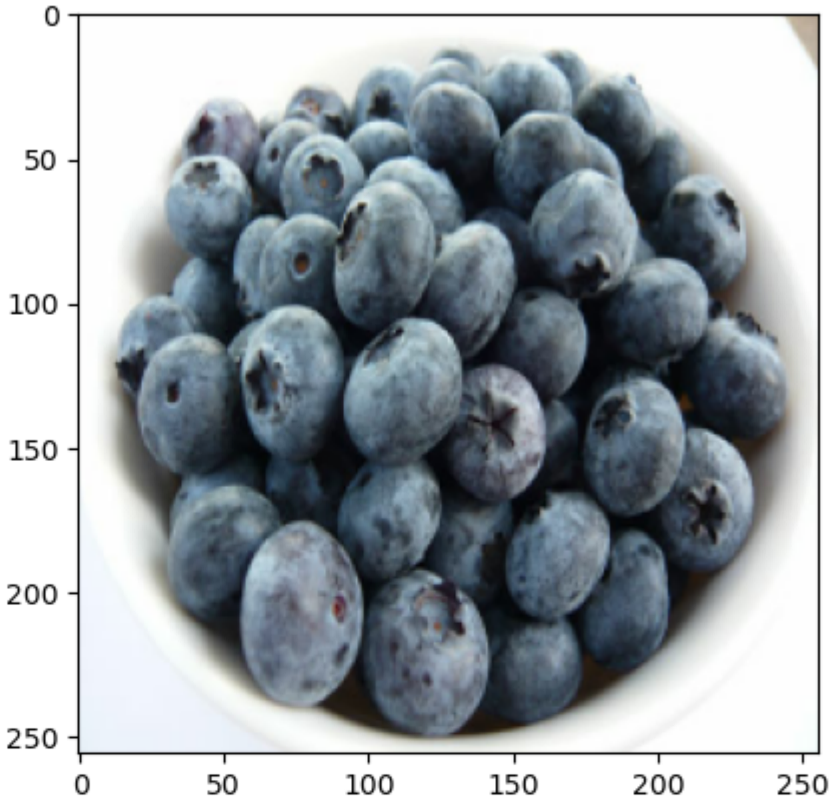
```python
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])
```

```
first image to predict
actual label: blueberries
1/1 [==============================] - 2s 2s/step
predicted label: blueberries
```



Write a function for inference

In [39]:

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].num
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

Now run inference on few sample images

In [40]:

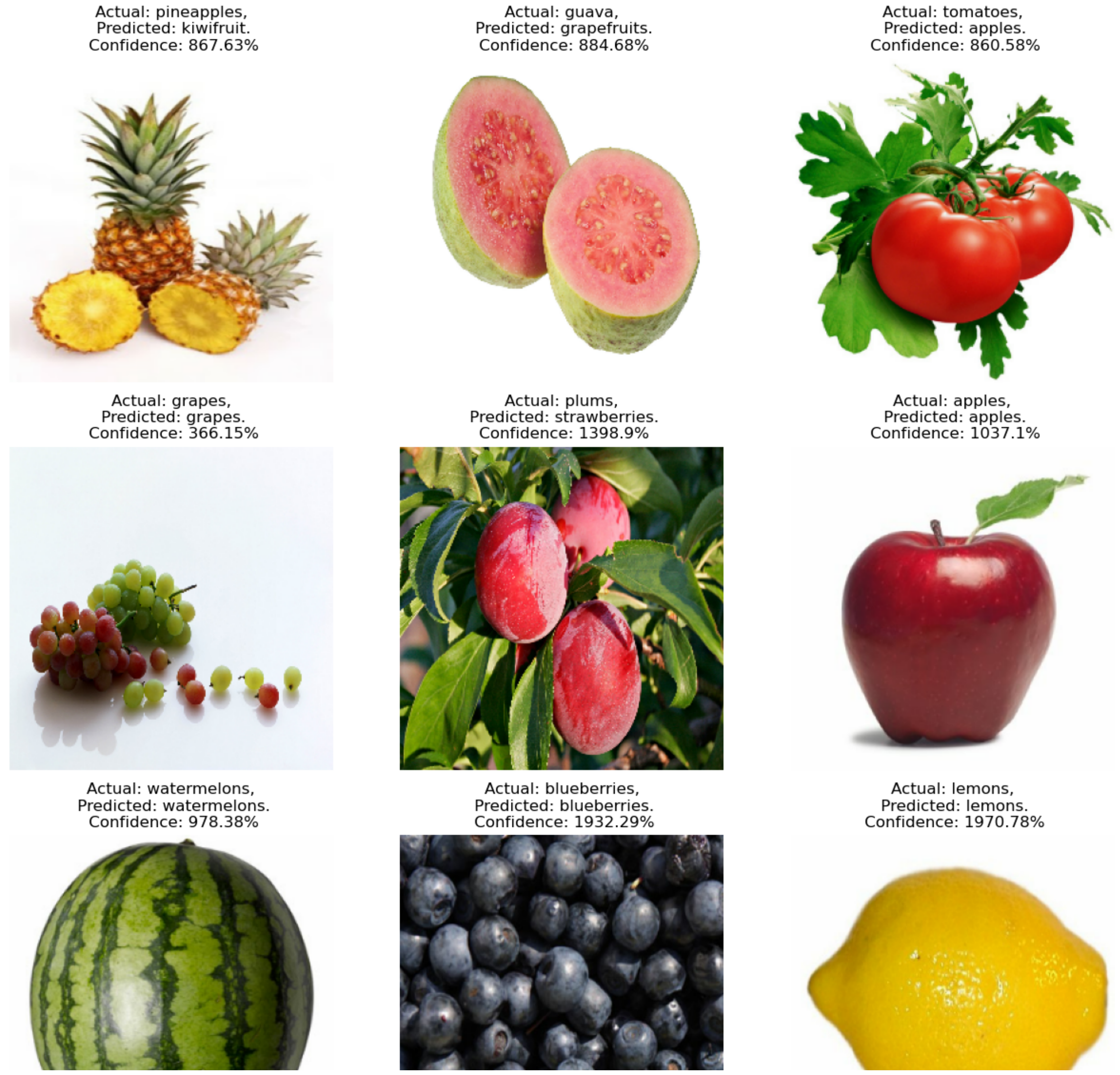
```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_cla

        plt.axis("off")
```

```
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 64ms/step
```



Actual: pineapples,
Predicted: kiwifruit.
Confidence: 867.63%

Actual: guava,
Predicted: grapefruits.
Confidence: 884.68%

Actual: tomatoes,
Predicted: apples.
Confidence: 860.58%

Actual: grapes,
Predicted: grapes.
Confidence: 366.15%

Actual: plums,
Predicted: strawberries.
Confidence: 1398.9%

Actual: apples,
Predicted: apples.
Confidence: 1037.1%

Actual: watermelons,
Predicted: watermelons.
Confidence: 978.38%

Actual: blueberries,
Predicted: blueberries.
Confidence: 1932.29%

Actual: lemons,
Predicted: lemons.
Confidence: 1970.78%

In [41]:

```python
import os
model_version=max([int(i) for i in os.listdir("./models") + [0]])+1
model.save(f"./models/{model_version}")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolutio
n_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _upd
ate_step_xla while saving (showing 4 of 4). These functions will not b
e directly callable after loading.

INFO:tensorflow:Assets written to: ./models/4/assets

INFO:tensorflow:Assets written to: ./models/4/assets
```

In [ ]: