Pro_Training

February 24, 2024

0.1 LOAD LIBRARIES

[17]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import os
```

[18]:
```
Image_Size = 256
BATCH_SIZE = 32
Channels = 3
EPOCHS =20
```

0.2 This Code Loads the Data Into Tensorflow Database

[19]:
```
Dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "Data",
    shuffle = True,
    image_size=(Image_Size,Image_Size),
    batch_size = BATCH_SIZE
)
```

```
Found 248 files belonging to 3 classes.
```

0.3 Analyse the DATA

[20]:
```
Class_names = Dataset.class_names
```

[21]:
```
Class_names
```

[21]:
```
['Blackpod', 'Frosty', 'Healthy']
```

[22]:
```
len(Dataset)
```

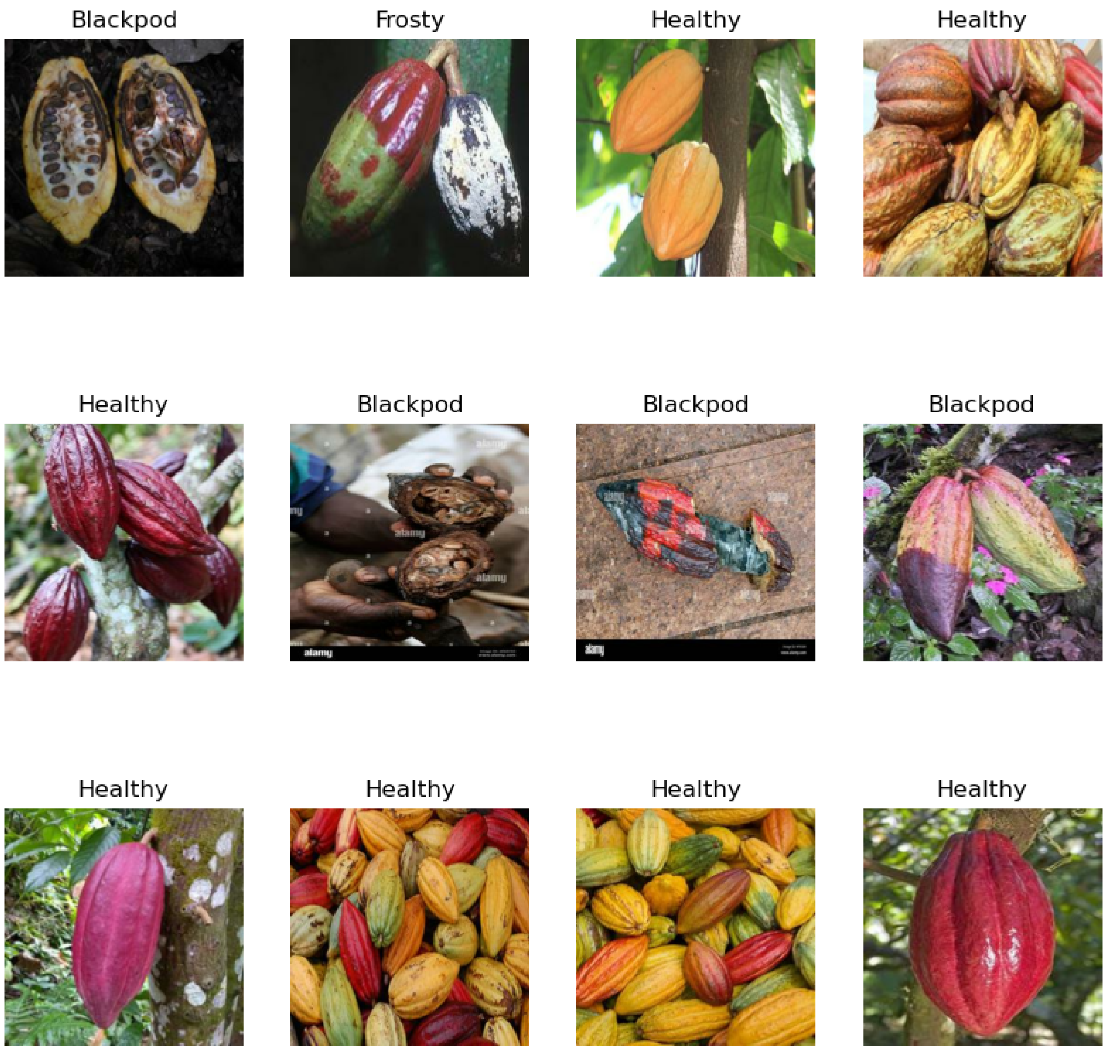
[22]:
```
8
```

[23]:
```
9*32
```

[23]: 288

[24]:
```
plt.figure(figsize=(10,10))
for image_batch, label_batch in Dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3,4,i+1)

        #print(image_batch.shape)
        #print(label_batch.numpy())
        #print(image_batch[0].numpy)## Changing tensor to a numpy
        #print(image_batch[0].shape)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.axis("off")
        plt.title(Class_names[label_batch[i]])
```



```
[25]: len(Dataset)
```

```
[25]: 8
```

0.4 Splitting the DATASET

```
[26]: #80%  ==>  training
      #20%  ==>  10% validation, 10% test to measure the accuracy of the model
```

0.4.1 Using Dataset.take to Split the DATA

```
[27]: train_size = 0.8
      len(Dataset)* train_size # getting the percentage of the train data size from␣
       ↪the whole data
```

```
[27]: 6.4
```

```
[28]: train_ds = Dataset.take(7)
      len(train_ds)## Taking the train size percentage from the Data
```

```
[28]: 7
```

```
[29]: test_ds = Dataset.skip(7)
      len(test_ds)## Skipping  the train size to get the test size
```

```
[29]: 1
```

```
[30]: val_size =0.1
      len(Dataset)*val_size## splitting the test size into Validation dataset and␣
       ↪test dataset
```

```
[30]: 0.8
```

```
[31]: val_ds= test_ds.take(1)
      len(val_ds)
```

```
[31]: 1
```

```
[32]: test_ds = test_ds.skip(1)
      len(test_ds)
```

```
[32]: 0
```

0.5 Splitting the DATASET

[33]:
```
def get_dataset_partitions_tf(ds, train_split = 0.8, val_split =0.1, test_split
↪=0.1, shuffle =True, shuffle_size =10000):
    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)## Taking the train size from the dataset
    val_ds = ds.skip(train_size).take(val_size)# Skip the train_size and the
↪remaining 20% take Val_size

    test_ds = val_ds= ds.skip(train_size).skip(val_size)## skip both train and
↪vals_size and the remaining is Test_size
    return train_ds, val_ds, test_ds
```

[34]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(Dataset)
```

[35]:
```
len(train_ds)
```

[35]: 6

[36]:
```
len(val_ds)
```

[36]: 2

[37]:
```
len(test_ds)
```

[37]: 2

0.6 Caching to improve the performance of the pipeline

Shuffle 1000 will shuffle the images

Prefetch to loads the next set of batch from the disk to improve performance

[38]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size =tf.data.
↪AUTOTUNE)

val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size =tf.data.AUTOTUNE)

test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size =tf.data.AUTOTUNE)
↪## Necessary for training performance
```

0.6.1 Preprocessing Resizing and Rescaling

[39]:
```
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(Image_Size,Image_Size),
    layers.experimental.preprocessing.Rescaling(1.0/255) ## Rescaling the␣
 ↪images to 255
])
```

0.7 Creating more samples due to the fewer images to maximize the variables for effective prediction

[40]:
```
data_augmentaion = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2), ## Randomflip and␣
 ↪some rotation to have a diverse forms of the data
    layers.experimental.preprocessing.RandomZoom(0.1),## to make the model a␣
 ↪robust one.
    layers.experimental.preprocessing.RandomContrast(0.1)
])
```

0.8 Build First Classifier (CNN)

[41]:
```
input_shape = (BATCH_SIZE,Image_Size,Image_Size,Channels)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    data_augmentaion,
    layers.Conv2D(32,(3,3), activation = 'relu',input_shape =input_shape),##␣
 ↪Need to have a lot of layers in order for the prediction to be inact
    layers.MaxPooling2D((2,2)),## this helps to scans over the image to pull␣
 ↪out the max values of the image

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64,(3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64,(3,3), activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),## flatten
    layers.Dense(64, activation = 'relu'),# and add a densed layer
```

```
    layers.Dense(n_classes, activation ='softmax')])# softmax activation␣
↪fucntion normalizes the probability of the classes

model.build(input_shape = input_shape) ## Force defining the Nero achitecture
```

[42]:
```
model.summary()### Module achitecture
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2  (32, 127, 127, 32)        0
 D)

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0
 g2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0
 g2D)

 flatten (Flatten)           (32, 2304)                0

 dense (Dense)               (32, 64)                  147520

 dense_1 (Dense)             (32, 3)                   195

=================================================================
```

```
Total params: 277891 (1.06 MB)
Trainable params: 277891 (1.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

0.9 Defining the optimizer, loss function and metrics

[43]:
```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy'])## accuracy is the metric used to track the training␣
 ↪process

#callback = keras.callbacks.EarlyStopping(monitor = 'val_loss',
                                       # patience = 3,
                                       # restore_best_weights = True)
```

0.10 Training the Network

[44]:
```
history = model.fit(
    train_ds,
    epochs =EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds
)
```

```
Epoch 1/20
6/6 [==============================] - 7s 875ms/step - loss: 13.2165 - accuracy:
0.3913 - val_loss: 1.4085 - val_accuracy: 0.2812
Epoch 2/20
6/6 [==============================] - 5s 852ms/step - loss: 1.2479 - accuracy:
0.3315 - val_loss: 1.0224 - val_accuracy: 0.4688
Epoch 3/20
6/6 [==============================] - 5s 875ms/step - loss: 0.8919 - accuracy:
0.5761 - val_loss: 0.9722 - val_accuracy: 0.5781
Epoch 4/20
6/6 [==============================] - 6s 978ms/step - loss: 0.8996 - accuracy:
0.5761 - val_loss: 0.9307 - val_accuracy: 0.5938
Epoch 5/20
6/6 [==============================] - 5s 915ms/step - loss: 0.8532 - accuracy:
0.6087 - val_loss: 0.8799 - val_accuracy: 0.5781
Epoch 6/20
6/6 [==============================] - 5s 903ms/step - loss: 0.7242 - accuracy:
0.6957 - val_loss: 0.8324 - val_accuracy: 0.6562
Epoch 7/20
6/6 [==============================] - 6s 963ms/step - loss: 0.7483 - accuracy:
```

```
0.6685 - val_loss: 0.7939 - val_accuracy: 0.6562
Epoch 8/20
6/6 [==============================] - 5s 922ms/step - loss: 0.6945 - accuracy:
0.7011 - val_loss: 0.9994 - val_accuracy: 0.5469
Epoch 9/20
6/6 [==============================] - 6s 1s/step - loss: 0.6763 - accuracy:
0.7011 - val_loss: 0.7517 - val_accuracy: 0.7031
Epoch 10/20
6/6 [==============================] - 6s 993ms/step - loss: 0.6432 - accuracy:
0.6957 - val_loss: 0.8828 - val_accuracy: 0.6250
Epoch 11/20
6/6 [==============================] - 6s 985ms/step - loss: 0.6038 - accuracy:
0.7391 - val_loss: 0.7323 - val_accuracy: 0.7344
Epoch 12/20
6/6 [==============================] - 6s 1s/step - loss: 0.5418 - accuracy:
0.7554 - val_loss: 0.6752 - val_accuracy: 0.7812
Epoch 13/20
6/6 [==============================] - 6s 976ms/step - loss: 0.5476 - accuracy:
0.7772 - val_loss: 1.0608 - val_accuracy: 0.5781
Epoch 14/20
6/6 [==============================] - 6s 969ms/step - loss: 0.6713 - accuracy:
0.7011 - val_loss: 0.6956 - val_accuracy: 0.7500
Epoch 15/20
6/6 [==============================] - 6s 920ms/step - loss: 0.6413 - accuracy:
0.6793 - val_loss: 1.1077 - val_accuracy: 0.5156
Epoch 16/20
6/6 [==============================] - 6s 931ms/step - loss: 0.7253 - accuracy:
0.6359 - val_loss: 0.7474 - val_accuracy: 0.6719
Epoch 17/20
6/6 [==============================] - 6s 952ms/step - loss: 0.5881 - accuracy:
0.7663 - val_loss: 0.6395 - val_accuracy: 0.7812
Epoch 18/20
6/6 [==============================] - 6s 939ms/step - loss: 0.5265 - accuracy:
0.7717 - val_loss: 0.6395 - val_accuracy: 0.8281
Epoch 19/20
6/6 [==============================] - 5s 887ms/step - loss: 0.4703 - accuracy:
0.8207 - val_loss: 0.8651 - val_accuracy: 0.6875
Epoch 20/20
6/6 [==============================] - 5s 916ms/step - loss: 0.4997 - accuracy:
0.7989 - val_loss: 0.6610 - val_accuracy: 0.7969
```

0.10.1 To define how well the model is performed with a data that hasn't been seen by the model in order to avoid any bias

[45]:
```
scores = model.evaluate(test_ds)## runing the model on the test_ds for the␣
↪first time (avoid bias)
```

```
2/2 [==============================] - 1s 260ms/step - loss: 0.3474 - accuracy:
```

0.8281

[46]:
```
scores
```

[46]:
```
[0.3474399745464325, 0.828125]
```

[47]:
```
history
```

[47]:
```
<keras.src.callbacks.History at 0x1485a2ed0>
```

[48]:
```
history.params
```

[48]:
```
{'verbose': 1, 'epochs': 20, 'steps': 6}
```

[49]:
```
history.history.keys()
```

[49]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

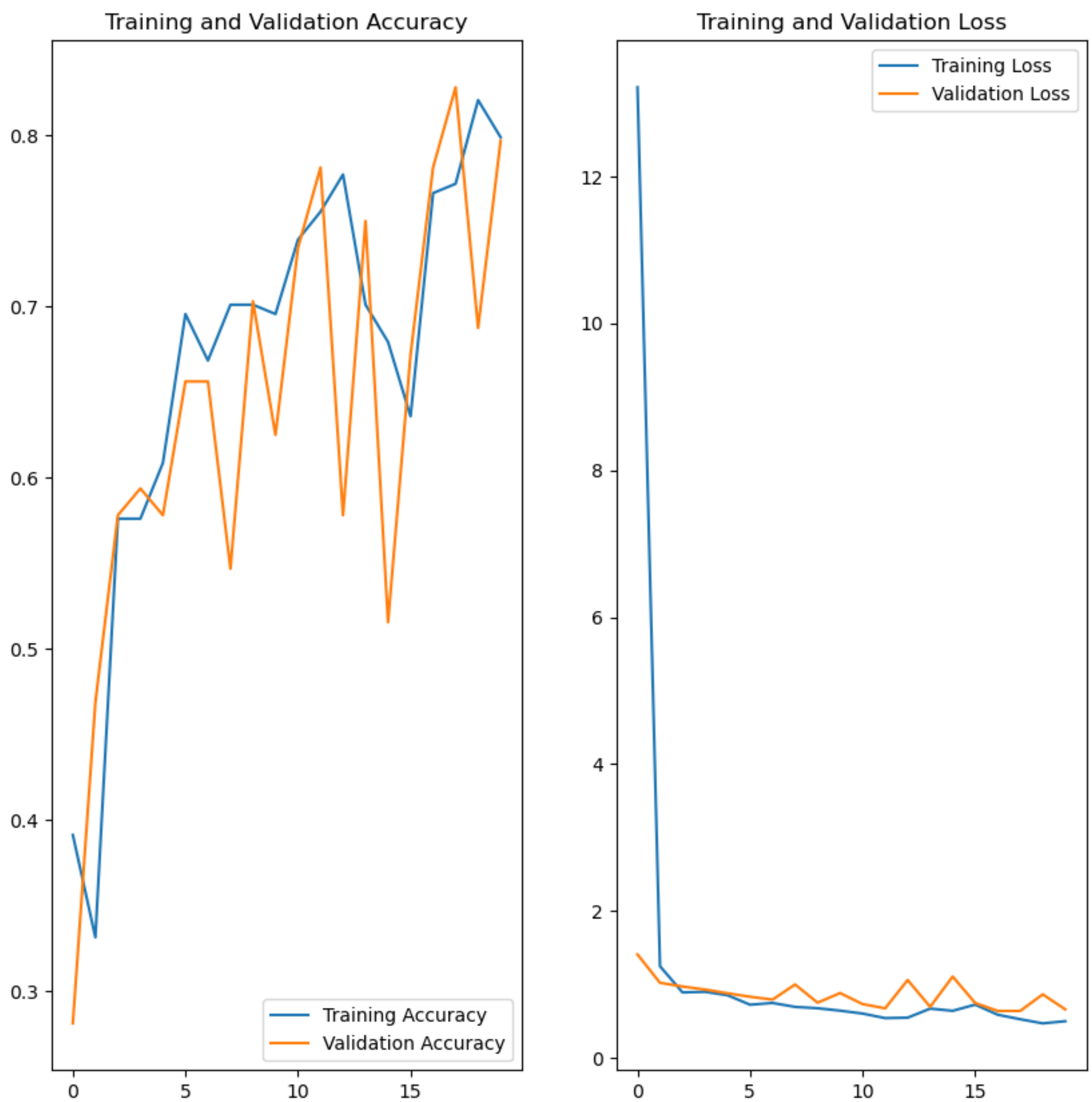
0.11 Plotting History

[50]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

[51]:
```
plt.figure(figsize=(10,10))
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') ## High accuracy was achieved

#plt.figure(figsize=(8,8))
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()
```



[52]:
```
np.argmax([1.3001359e-05, 1.9586462e-04, 9.9979120e-01])
```

[52]: 2

0.12 Making a Prediction

[58]:
```
plt.figure(figsize=(8,8))
for images_batch, labels_batch in test_ds.take(1):## taking one batch
    first_image = images_batch[4].numpy().astype('uint8')
    first_label = label_batch[4].numpy()

    print("First image to Predict")
```

```python
    plt.imshow(first_image)
    print("Actual Label:",Class_names[first_label])

    batch_prediction = model.predict(image_batch)
    print("Predicted Label:",Class_names[np.argmax(batch_prediction[4])])
    plt.axis('off')
```

```
First image to Predict
Actual Label: Healthy
1/1 [==============================] - 0s 229ms/step
Predicted Label: Healthy
```

0.13 Function Determining the Predicted_Class/Confidence_Level of the Model

[54]:
```
def predict(model,img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array,0)# create batch

    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
```

[59]:
```
plt.figure(figsize=(15,15))
for images, labels in test_ds.take(1):## one batch
    for i in range(6):## displaying only 9 images out if the batch
        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_Class}.\n␣
↪Confidence: {Confidence}%")

        plt.axis('off')
        plt.savefig('Fig')
```

```
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
```

0.14 Saving the Model

[ ]:
```
model_version = 1
model.save(f"../models/{model_version }")# model.. will take your from present directory to the new model directory
```

[ ]:
```
#import os
#model_version = max([int(i) for i in os.listdir("../models")+[0]])+1 # Changing a String to Integer
#model.save(f"../models/{model_version }")
```

[ ]:
```
```

[ ]:
```
```