In this blog post, we will begin a tutorial in Tensorflow related to image classification. The aim of the image calssification is to teach a machine learning algorithm to distinguish between pictures of dogs and pictures of cats.

Part 1:Load Packages and Obtain Data

Before we start coding, we will need to import necessary packages

```
import os
from tensorflow.keras import utils
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import datasets, layers, models
from scipy.signal import convolve2d
```

Next, we need to access the sample data, provided by the TensorFlow team that contains labeled images of cats and dogs. We will use the following codes to create TensorFlow Datasets for training, validation, and testing.

```

# location of data
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'

# download the data and extract it
path_to_zip = utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)

# construct paths
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

# parameters for datasets
BATCH_SIZE = 32
IMG_SIZE = (160, 160)
```

```
# construct train and validation datasets
train_dataset = utils.image_dataset_from_directory(train_dir,
                                                   shuffle=True,
                                                   batch_size=BATCH_SIZE,
                                                   image_size=IMG_SIZE)

validation_dataset = utils.image_dataset_from_directory(validation_dir,
                                                        shuffle=True,
                                                        batch_size=BATCH_SIZE,
                                                        image_size=IMG_SIZE)
class_names = train_dataset.class_names

# construct the test dataset by taking every 5th observation out of the validation dataset
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [==============================] - 4s 0us/step
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

After running the code, we will get such a output, whcih means we construct three datasets successfully.

Step1:Rapidly Reading Data

In this step, these codes could help us increase the speed of reading data.

```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

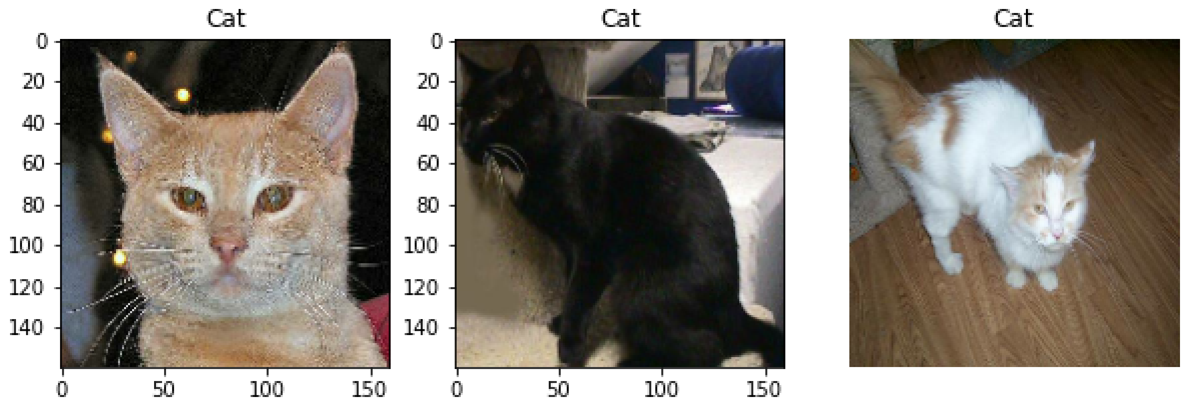
Step 2:Working with datasets

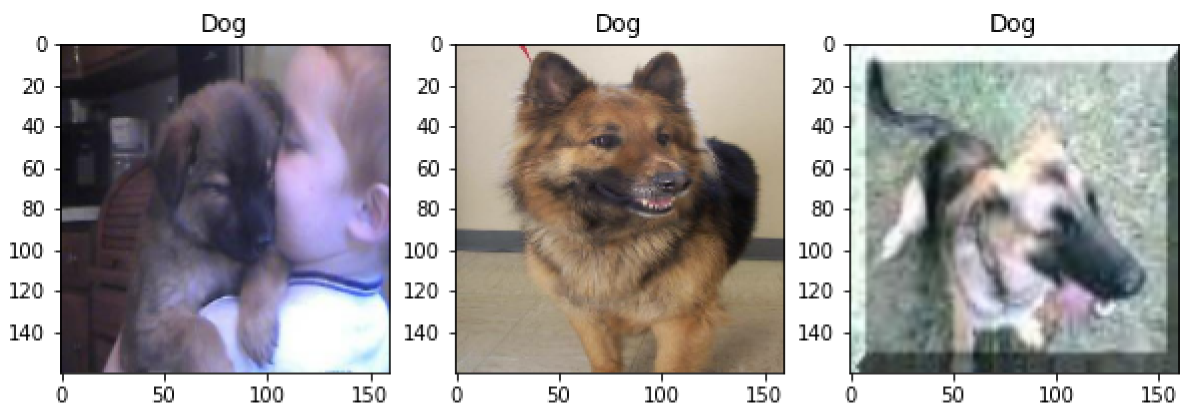
In this step, we will write a two-row visualization function to create a two row plots with total 6 images . In the first row, we will show three random pictures of cats. In the second row, showing three random pictures of dogs.

```
def two_row_visualization():
  plt.figure(figsize=(10, 10))
  #index for cats and dogs
  cats=0
  dogs=0
  # retrieve one batch (32 images with labels) from the training data
  #loop through the dataset to get images and labels
  for images, labels in train_dataset.take(1):
    #loop through 32 images
    for i in range(32):
      #choose random cats images
      if class_names[labels[i]]=="cats" and cats<3:
        ax = plt.subplot(2, 3, cats + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title("Cat")
        cats+=1
        #choose random dogs images
      elif class_names[labels[i]]=="dogs" and dogs<3:
        ax = plt.subplot(2, 3, dogs + 4)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title("Dog")
        dogs+=1
    if cats>3 and dogs >3:
        break

  plt.axis("off")
```

Here is the output:

```
ran_p=two_row_visualization()
```





Step 3:Check Label Frequencies

In this step, we will create an iterator called labels and then we convert it to a list to count the number of images in thetraining data with label 0 (corresponding to “cat”) and label 1 (corresponding to “dog”).

```
#numpy.ndarray
labels_iterator= train_dataset.unbatch().map(lambda image, label: label).as_numpy_iterator()
#convert to list
labels_list=list(labels_iterator)
#count the frequency
```

```
cats_f=labels_list.count(0)
dogs_f=labels_list.count(1)
print("Number of cat images in the training data:"+str(cats_f))
print("Number of dog images in the training data:"+str(dogs_f))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.8/dist-
packages/tensorflow/python/autograph/pyct/static_analysis/liveness.py:83: Analyzer.lamba_check (from
tensorflow.python.autograph.pyct.static_analysis.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 where they are used, or at least in the same block.
https://github.com/tensorflow/tensorflow/issues/56089

Number of cat images in the training data:1000
Number of dog images in the training data:1000
```

Since the baseline machine learning model is the model that always guesses the most frequent label, here we have same numbers of images for cats and dogs,which are 1000 and 1000 respectively. Thus, we can guess the accuracy is 0.5.

Part2: First Model

In this part, we will create our first model which named model1. We will create a tf.keras.Sequential model using some layers:

- Conv2D layers
- MaxPooling2D layers
- Flatten layer
- Dense layer
- Dropout layer

We can create a plot_accuracy() function to help us visualize the accuracy of our model.The x axis is the epoch and y axis is accuracy. In our graph, we can observe validation accuracy and training accuracy.

```
def plot_accuracy(history):
  acc = history.history['accuracy']
  val_acc = history.history['val_accuracy']
  plt.plot(acc, label='Training Accuracy')
  plt.plot(val_acc, label='Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
model1 = models.Sequential([
      layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),
      layers.MaxPooling2D((3, 3)),
      layers.Conv2D(32, (3, 3), activation='relu'),
      layers.MaxPooling2D((3, 3)),
      layers.Flatten(),
      layers.Dropout(0.1),
      layers.Dense(128, activation='relu'),
      layers.Dense(2)
])
```

Complie model

```
model1.compile(optimizer='adam',
              loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics = ['accuracy'])
```

Train the model

```
history1 = model1.fit(train_dataset,
                     epochs=20,
                     validation_data=validation_dataset)
```

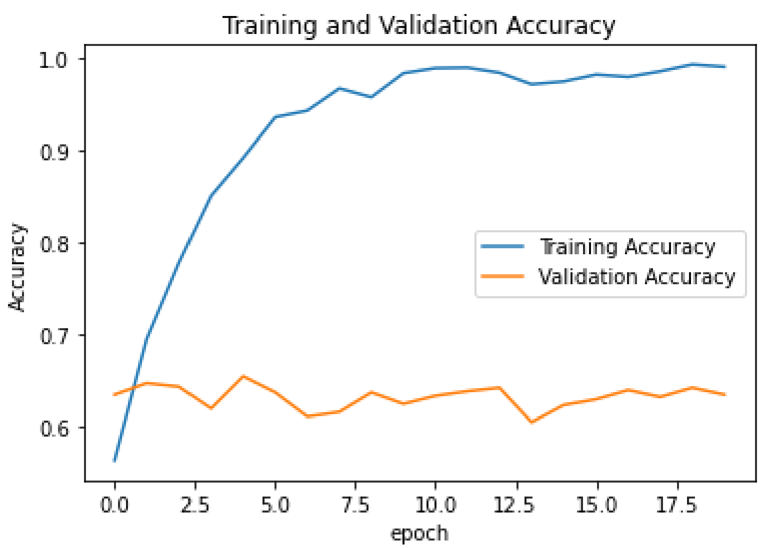
```
Epoch 1/20
63/63 [==============================] - 6s 59ms/step - loss: 9.9501 - accuracy: 0.5640 - val_loss: 0.6564 -
val_accuracy: 0.6349
Epoch 2/20
63/63 [==============================] - 3s 48ms/step - loss: 0.5842 - accuracy: 0.6950 - val_loss: 0.6774 -
```

```
val_accuracy: 0.6473
Epoch 3/20
63/63 [==============================] - 4s 57ms/step - loss: 0.4787 - accuracy: 0.7775 - val_loss: 0.7158 -
val_accuracy: 0.6436
Epoch 4/20
63/63 [==============================] - 4s 52ms/step - loss: 0.3456 - accuracy: 0.8505 - val_loss: 0.8179 -
val_accuracy: 0.6200
Epoch 5/20
63/63 [==============================] - 3s 48ms/step - loss: 0.2768 - accuracy: 0.8915 - val_loss: 1.0516 -
val_accuracy: 0.6547
Epoch 6/20
63/63 [==============================] - 3s 48ms/step - loss: 0.1761 - accuracy: 0.9355 - val_loss: 1.2554 -
val_accuracy: 0.6374
Epoch 7/20
63/63 [==============================] - 4s 52ms/step - loss: 0.1482 - accuracy: 0.9425 - val_loss: 1.1927 -
val_accuracy: 0.6114
Epoch 8/20
63/63 [==============================] - 3s 47ms/step - loss: 0.1027 - accuracy: 0.9665 - val_loss: 1.4451 -
val_accuracy: 0.6163
Epoch 9/20
63/63 [==============================] - 3s 48ms/step - loss: 0.1255 - accuracy: 0.9570 - val_loss: 1.4829 -
val_accuracy: 0.6374
Epoch 10/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0584 - accuracy: 0.9830 - val_loss: 1.7480 -
val_accuracy: 0.6250
Epoch 11/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0408 - accuracy: 0.9885 - val_loss: 1.6963 -
val_accuracy: 0.6337
Epoch 12/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0377 - accuracy: 0.9890 - val_loss: 2.0082 -
val_accuracy: 0.6386
Epoch 13/20
63/63 [==============================] - 3s 49ms/step - loss: 0.0493 - accuracy: 0.9835 - val_loss: 1.9963 -
val_accuracy: 0.6423
Epoch 14/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0968 - accuracy: 0.9710 - val_loss: 1.8765 -
val_accuracy: 0.6052
Epoch 15/20
```

```
63/63 [==============================] - 4s 66ms/step - loss: 0.0795 - accuracy: 0.9740 - val_loss: 2.2500 -
val_accuracy: 0.6238
Epoch 16/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0554 - accuracy: 0.9815 - val_loss: 1.7859 -
val_accuracy: 0.6300
Epoch 17/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0731 - accuracy: 0.9790 - val_loss: 2.0104 -
val_accuracy: 0.6399
Epoch 18/20
63/63 [==============================] - 5s 71ms/step - loss: 0.0622 - accuracy: 0.9850 - val_loss: 2.0048 -
val_accuracy: 0.6324
Epoch 19/20
63/63 [==============================] - 3s 48ms/step - loss: 0.0318 - accuracy: 0.9925 - val_loss: 2.0063 -
val_accuracy: 0.6423
Epoch 20/20
63/63 [==============================] - 5s 65ms/step - loss: 0.0298 - accuracy: 0.9900 - val_loss: 2.0624 -
val_accuracy: 0.6349
```

Plot the Accuracy

```
plot_accuracy(history1)
```



Summary of model1:

- The accuracy of my model stabilized between **61% and 65%** during training.
- Compared to the baseline,Model 1 is 13% better.
- Overfitting was observed in model 1.

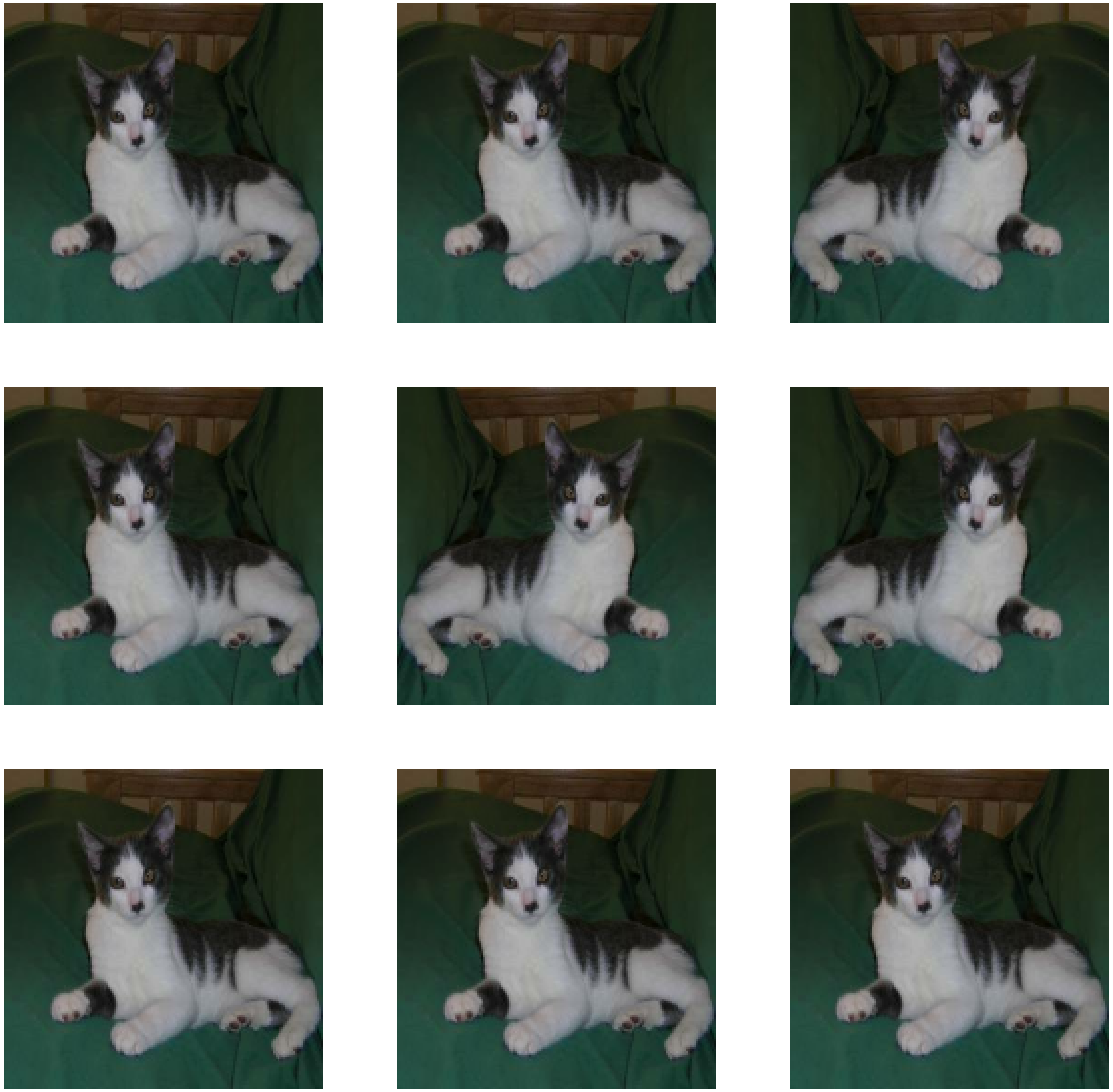
Part3: Model with Data Augmentation

Now we're going to add some data augmentation layers to our model. Data augmentation refers to the practice of including modified copies of the same image in the training set. We will add two layers here

- tf.keras.layers.RandomFlip()
- tf.keras.layers.RandomRotation()

Step1:create a tf.keras.layers.RandomFlip() layer

```
ranFlip= tf.keras.Sequential(tf.keras.layers.RandomFlip('horizontal'))
#flip horizontally
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  #print 9 pictures
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = ranFlip(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



Step2: create a tf.keras.layers.RandomRotation() layer

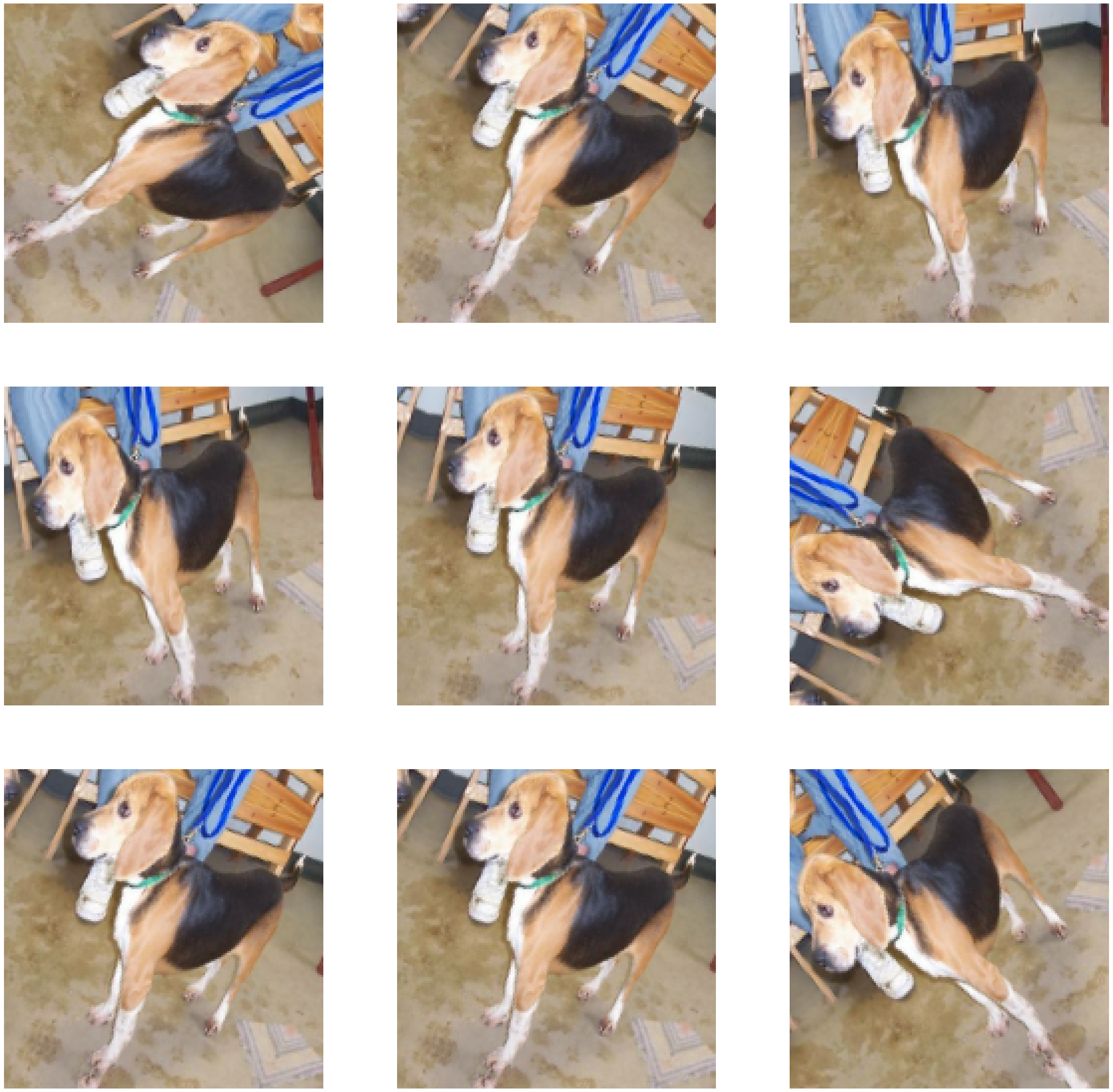
```
ranRota= tf.keras.Sequential(tf.keras.layers.RandomRotation(0.2))
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
```

```
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = ranRota(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
```

```
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:5 out of the last 5 calls to <function pfor.<locals>.f at 0x7f232a1e2670> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:6 out of the last 6 calls to <function pfor.<locals>.f at 0x7f23ac094c10> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
```

```
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
```



Second Model

Now we will create a new tf.keras.models.Sequential model called model2 in which the first two layers are augmentation layers

- RandomFlip()
- RandomRotation()

```
model2 = tf.keras.models.Sequential([
    layers.RandomFlip('horizontal'),
    layers.RandomRotation(0.2),
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((3, 3)),
    layers.Flatten(),
    layers.Dropout(0.1),
    layers.Dense(128, activation='relu'),
    layers.Dense(2) # number of classes in  dataset
])
```

Complie the Model

```
sce = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model2.compile(optimizer='adam',
             loss = sce,
              metrics = ['accuracy'])
```

Train the Model

```
history2 = model2.fit(train_dataset,
                     epochs=20,
                     validation_data=validation_dataset)
```

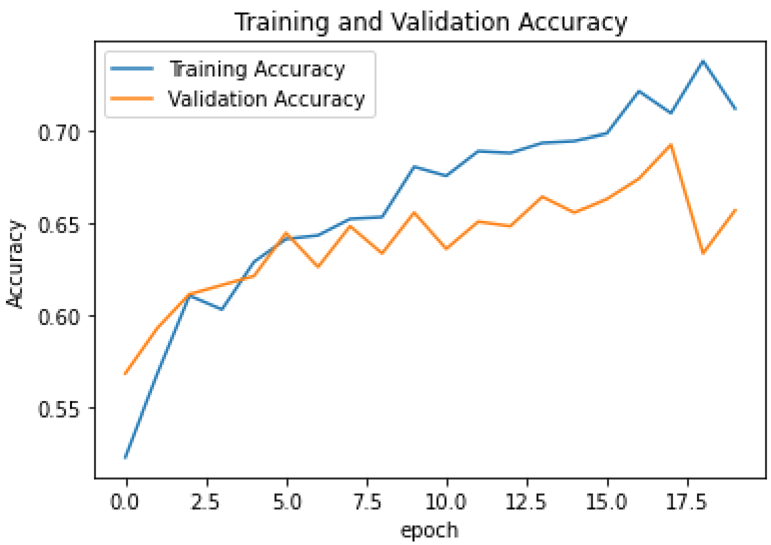
```
Epoch 1/20

WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for
this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter
```

```
for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for
this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter
for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for
this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter
for this op.
63/63 [==============================] - 12s 112ms/step - loss: 12.1113 - accuracy: 0.5225 - val_loss: 0.6846 -
val_accuracy: 0.5681
Epoch 2/20
63/63 [==============================] - 8s 125ms/step - loss: 0.6817 - accuracy: 0.5680 - val_loss: 0.6756 -
val_accuracy: 0.5928
Epoch 3/20
63/63 [==============================] - 8s 126ms/step - loss: 0.6590 - accuracy: 0.6105 - val_loss: 0.6628 -
val_accuracy: 0.6114
Epoch 4/20
63/63 [==============================] - 8s 128ms/step - loss: 0.6641 - accuracy: 0.6030 - val_loss: 0.6517 -
val_accuracy: 0.6163
Epoch 5/20
63/63 [==============================] - 7s 111ms/step - loss: 0.6483 - accuracy: 0.6290 - val_loss: 0.6559 -
val_accuracy: 0.6213
Epoch 6/20
63/63 [==============================] - 7s 110ms/step - loss: 0.6312 - accuracy: 0.6415 - val_loss: 0.6584 -
val_accuracy: 0.6448
Epoch 7/20
63/63 [==============================] - 10s 158ms/step - loss: 0.6347 - accuracy: 0.6435 - val_loss: 0.6386 -
val_accuracy: 0.6262
Epoch 8/20
63/63 [==============================] - 8s 125ms/step - loss: 0.6287 - accuracy: 0.6525 - val_loss: 0.6313 -
val_accuracy: 0.6485
```

```
Epoch 9/20
63/63 [==============================] - 7s 110ms/step - loss: 0.6130 - accuracy: 0.6535 - val_loss: 0.6302 -
val_accuracy: 0.6337
Epoch 10/20
63/63 [==============================] - 7s 115ms/step - loss: 0.5989 - accuracy: 0.6810 - val_loss: 0.6404 -
val_accuracy: 0.6559
Epoch 11/20
63/63 [==============================] - 8s 125ms/step - loss: 0.5979 - accuracy: 0.6760 - val_loss: 0.6377 -
val_accuracy: 0.6361
Epoch 12/20
63/63 [==============================] - 8s 126ms/step - loss: 0.5798 - accuracy: 0.6895 - val_loss: 0.6337 -
val_accuracy: 0.6510
Epoch 13/20
63/63 [==============================] - 7s 110ms/step - loss: 0.5821 - accuracy: 0.6885 - val_loss: 0.6400 -
val_accuracy: 0.6485
Epoch 14/20
63/63 [==============================] - 8s 127ms/step - loss: 0.5740 - accuracy: 0.6940 - val_loss: 0.6184 -
val_accuracy: 0.6646
Epoch 15/20
63/63 [==============================] - 8s 128ms/step - loss: 0.5685 - accuracy: 0.6950 - val_loss: 0.6128 -
val_accuracy: 0.6559
Epoch 16/20
63/63 [==============================] - 7s 111ms/step - loss: 0.5626 - accuracy: 0.6990 - val_loss: 0.6196 -
val_accuracy: 0.6634
Epoch 17/20
63/63 [==============================] - 8s 115ms/step - loss: 0.5541 - accuracy: 0.7220 - val_loss: 0.6222 -
val_accuracy: 0.6745
Epoch 18/20
63/63 [==============================] - 8s 126ms/step - loss: 0.5585 - accuracy: 0.7100 - val_loss: 0.6052 -
val_accuracy: 0.6931
Epoch 19/20
63/63 [==============================] - 7s 111ms/step - loss: 0.5307 - accuracy: 0.7385 - val_loss: 1.4120 -
val_accuracy: 0.6337
Epoch 20/20
63/63 [==============================] - 8s 123ms/step - loss: 0.5560 - accuracy: 0.7125 - val_loss: 0.6461 -
val_accuracy: 0.6572
```

Plot the Accuracy

```
plot_accuracy(history2)
```



Summary of model2:

- The accuracy of my model stabilized between **57% and 69%** during training.
- Compared to Model1,Model 2 has similar average validation accuracy.However,model 2 shows more inconsistent overall accuracy.
- Overfitting was observed in the beginning and end epoches in model 2.

Part4: Data Preprocessing

In this part, we will create a model 3 which has a preprocessor layer as the very first layer, before the data augmentation layers.

create a preprocessing layer

```
i = tf.keras.Input(shape=(160, 160, 3))
x = tf.keras.applications.mobilenet_v2.preprocess_input(i)
preprocessor = tf.keras.Model(inputs = [i], outputs = [x])
```

```
model3 = tf.keras.models.Sequential([
    preprocessor,
    layers.RandomFlip('horizontal'),
    layers.RandomRotation(0.2),
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(160, 160, 3)),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((3, 3)),

    layers.Flatten(),
    layers.Dropout(0.1),
    layers.Dense(128, activation='relu'),
    layers.Dense(2) # number of classes in  dataset
])
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
```

Complie the Model

```
sce = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model3.compile(optimizer='adam',
             loss = sce,
              metrics = ['accuracy'])
```

Train the Model

```
history3 = model3.fit(train_dataset,
                     epochs=20,
```

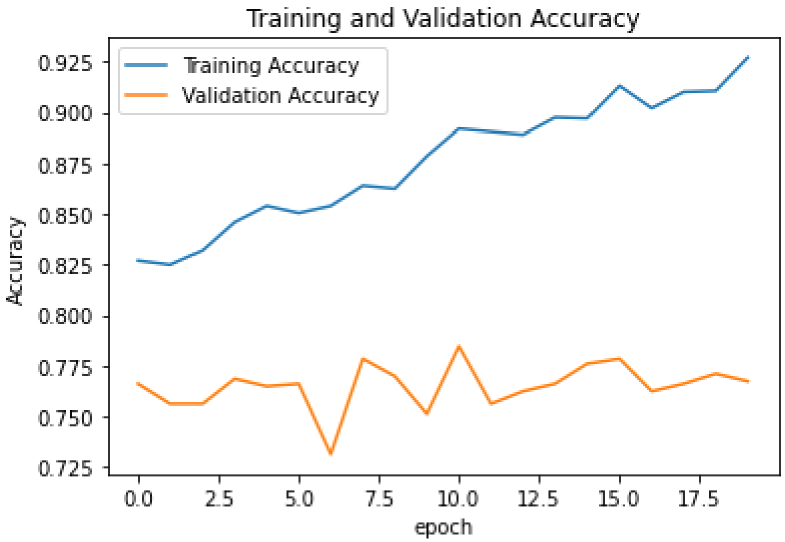
```
                      validation_data=validation_dataset)
```

```
Epoch 1/20
63/63 [==============================] - 7s 110ms/step - loss: 0.3672 - accuracy: 0.8270 - val_loss: 0.4967 -
val_accuracy: 0.7661
Epoch 2/20
63/63 [==============================] - 7s 111ms/step - loss: 0.3867 - accuracy: 0.8250 - val_loss: 0.5174 -
val_accuracy: 0.7562
Epoch 3/20
63/63 [==============================] - 8s 127ms/step - loss: 0.3796 - accuracy: 0.8320 - val_loss: 0.5068 -
val_accuracy: 0.7562
Epoch 4/20
63/63 [==============================] - 7s 110ms/step - loss: 0.3433 - accuracy: 0.8460 - val_loss: 0.5017 -
val_accuracy: 0.7686
Epoch 5/20
63/63 [==============================] - 8s 123ms/step - loss: 0.3288 - accuracy: 0.8540 - val_loss: 0.4800 -
val_accuracy: 0.7649
Epoch 6/20
63/63 [==============================] - 8s 126ms/step - loss: 0.3309 - accuracy: 0.8505 - val_loss: 0.4939 -
val_accuracy: 0.7661
Epoch 7/20
63/63 [==============================] - 8s 125ms/step - loss: 0.3461 - accuracy: 0.8540 - val_loss: 0.5499 -
val_accuracy: 0.7314
Epoch 8/20
63/63 [==============================] - 7s 110ms/step - loss: 0.3273 - accuracy: 0.8640 - val_loss: 0.4815 -
val_accuracy: 0.7785
Epoch 9/20
63/63 [==============================] - 7s 110ms/step - loss: 0.3134 - accuracy: 0.8625 - val_loss: 0.5029 -
val_accuracy: 0.7698
Epoch 10/20
63/63 [==============================] - 8s 126ms/step - loss: 0.2855 - accuracy: 0.8785 - val_loss: 0.5340 -
val_accuracy: 0.7512
Epoch 11/20
63/63 [==============================] - 8s 124ms/step - loss: 0.2656 - accuracy: 0.8920 - val_loss: 0.5483 -
val_accuracy: 0.7847
Epoch 12/20
63/63 [==============================] - 9s 146ms/step - loss: 0.2640 - accuracy: 0.8905 - val_loss: 0.6208 -
val_accuracy: 0.7562
```

```
Epoch 13/20
63/63 [==============================] - 7s 111ms/step - loss: 0.2577 - accuracy: 0.8890 - val_loss: 0.5905 -
val_accuracy: 0.7624
Epoch 14/20
63/63 [==============================] - 8s 125ms/step - loss: 0.2474 - accuracy: 0.8975 - val_loss: 0.5951 -
val_accuracy: 0.7661
Epoch 15/20
63/63 [==============================] - 7s 112ms/step - loss: 0.2333 - accuracy: 0.8970 - val_loss: 0.5880 -
val_accuracy: 0.7760
Epoch 16/20
63/63 [==============================] - 7s 111ms/step - loss: 0.2120 - accuracy: 0.9130 - val_loss: 0.5969 -
val_accuracy: 0.7785
Epoch 17/20
63/63 [==============================] - 8s 126ms/step - loss: 0.2364 - accuracy: 0.9020 - val_loss: 0.6172 -
val_accuracy: 0.7624
Epoch 18/20
63/63 [==============================] - 8s 126ms/step - loss: 0.2187 - accuracy: 0.9100 - val_loss: 0.6649 -
val_accuracy: 0.7661
Epoch 19/20
63/63 [==============================] - 7s 110ms/step - loss: 0.2227 - accuracy: 0.9105 - val_loss: 0.5889 -
val_accuracy: 0.7710
Epoch 20/20
63/63 [==============================] - 8s 125ms/step - loss: 0.1903 - accuracy: 0.9270 - val_loss: 0.6668 -
val_accuracy: 0.7673
```

Plot the Accuracy

```
plot_accuracy(history3)
```



Summary of model3:

- The accuracy of my model stabilized between **73% and 78%** during training.
- Model 3 is 22.5% better than model1.
- Overfitting was observed in model 3.

Part5: Transfer Learning

In this part,we will create a model 4.First,we need to access a pre-existing “base model”, incorporate it into a full model for our current task, and then train that model.

```
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
base_model.trainable = False

i = tf.keras.Input(shape=IMG_SHAPE)
x = base_model(i, training = False)
base_model_layer = tf.keras.Model(inputs = [i], outputs = [x])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
9406464/9406464 [==============================] - 1s 0us/step
```

Now, create a model called model4 that uses MobileNetV2 and the following layers

- The preprocessor layer from Part §4.
- The data augmentation layers from Part §3.
- The base_model_layer constructed above.
- A Dense(2) layer at the very end to actually perform the classification.

```
model4 = tf.keras.models.Sequential([
    preprocessor,
    layers.RandomFlip('horizontal'),
    layers.RandomRotation(0.2),
    base_model_layer,
    layers.GlobalMaxPooling2D(),
    layers.Dense(2) # number of classes in  dataset
])
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for
this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter
for this op.
```

Check the Complexity

According to the summary, we need to train 2562 parameters.

```
model4.summary()
```

```
Model: "sequential_12"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 model (Functional)          (None, 160, 160, 3)       0

 random_flip_6 (RandomFlip)  (None, 160, 160, 3)       0

 random_rotation_5 (RandomRo  (None, 160, 160, 3)      0
 tation)

 model_1 (Functional)        (None, 5, 5, 1280)        2257984

 global_max_pooling2d (Globa  (None, 1280)             0
 lMaxPooling2D)

 dense_22 (Dense)            (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

Complie and Train the Model

```
sce = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model4.compile(optimizer='adam',
             loss = sce,
              metrics = ['accuracy'])
history4 = model4.fit(train_dataset,
                    epochs=20,
                    validation_data=validation_dataset)
```

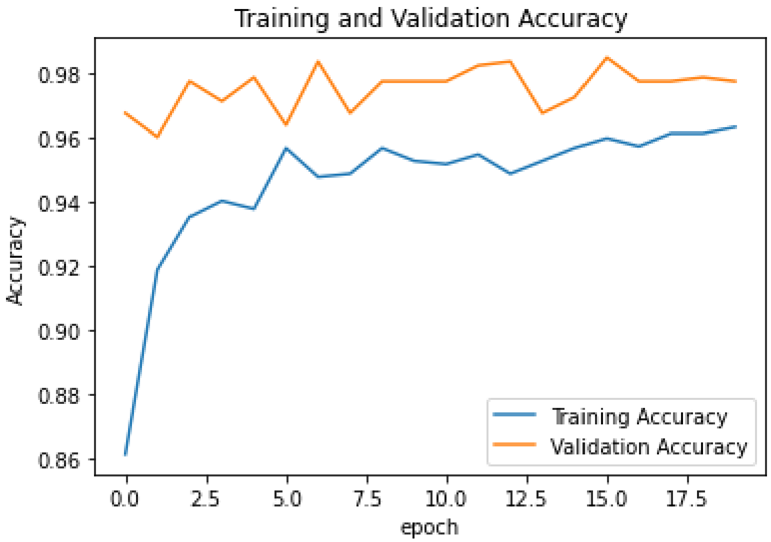
```
Epoch 1/20

WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for
```

```
this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
63/63 [==============================] - 16s 164ms/step - loss: 0.4795 - accuracy: 0.8615 - val_loss: 0.0972 - val_accuracy: 0.9678
Epoch 2/20
63/63 [==============================] - 9s 132ms/step - loss: 0.2678 - accuracy: 0.9190 - val_loss: 0.1255 - val_accuracy: 0.9604
Epoch 3/20
63/63 [==============================] - 9s 144ms/step - loss: 0.2040 - accuracy: 0.9355 - val_loss: 0.0679 - val_accuracy: 0.9777
Epoch 4/20
63/63 [==============================] - 9s 139ms/step - loss: 0.2101 - accuracy: 0.9405 - val_loss: 0.0680 - val_accuracy: 0.9715
Epoch 5/20
63/63 [==============================] - 9s 139ms/step - loss: 0.1959 - accuracy: 0.9380 - val_loss: 0.0621 - val_accuracy: 0.9790
Epoch 6/20
63/63 [==============================] - 9s 140ms/step - loss: 0.1453 - accuracy: 0.9570 - val_loss: 0.0871 - val_accuracy: 0.9641
Epoch 7/20
63/63 [==============================] - 8s 127ms/step - loss: 0.1651 - accuracy: 0.9480 - val_loss: 0.0394 - val_accuracy: 0.9839
Epoch 8/20
63/63 [==============================] - 9s 140ms/step - loss: 0.1678 - accuracy: 0.9490 - val_loss: 0.0970 - val_accuracy: 0.9678
Epoch 9/20
63/63 [==============================] - 9s 139ms/step - loss: 0.1559 - accuracy: 0.9570 - val_loss: 0.0607 - val_accuracy: 0.9777
Epoch 10/20
63/63 [==============================] - 9s 142ms/step - loss: 0.1682 - accuracy: 0.9530 - val_loss: 0.0628 -
```

```
val_accuracy: 0.9777
Epoch 11/20
63/63 [==============================] - 8s 126ms/step - loss: 0.1432 - accuracy: 0.9520 - val_loss: 0.0682 -
val_accuracy: 0.9777
Epoch 12/20
63/63 [==============================] - 9s 142ms/step - loss: 0.1319 - accuracy: 0.9550 - val_loss: 0.0613 -
val_accuracy: 0.9827
Epoch 13/20
63/63 [==============================] - 9s 139ms/step - loss: 0.1582 - accuracy: 0.9490 - val_loss: 0.0461 -
val_accuracy: 0.9839
Epoch 14/20
63/63 [==============================] - 8s 127ms/step - loss: 0.1280 - accuracy: 0.9530 - val_loss: 0.1154 -
val_accuracy: 0.9678
Epoch 15/20
63/63 [==============================] - 9s 135ms/step - loss: 0.1318 - accuracy: 0.9570 - val_loss: 0.0669 -
val_accuracy: 0.9728
Epoch 16/20
63/63 [==============================] - 9s 138ms/step - loss: 0.1086 - accuracy: 0.9600 - val_loss: 0.0392 -
val_accuracy: 0.9851
Epoch 17/20
63/63 [==============================] - 9s 140ms/step - loss: 0.1149 - accuracy: 0.9575 - val_loss: 0.0687 -
val_accuracy: 0.9777
Epoch 18/20
63/63 [==============================] - 8s 124ms/step - loss: 0.1135 - accuracy: 0.9615 - val_loss: 0.0658 -
val_accuracy: 0.9777
Epoch 19/20
63/63 [==============================] - 9s 133ms/step - loss: 0.1067 - accuracy: 0.9615 - val_loss: 0.0706 -
val_accuracy: 0.9790
Epoch 20/20
63/63 [==============================] - 9s 140ms/step - loss: 0.1000 - accuracy: 0.9635 - val_loss: 0.0708 -
val_accuracy: 0.9777
```

Plot the Accuracy

```
plot_accuracy(history4)
```



Summary of model4:

- The accuracy of my model stabilized between ** 97% and 99% ** during training.
- Model 4 is 48% better than model1.
- Overfitting was not observed in model 4.

Part6: Score on Test Data

```
model1.evaluate(test_dataset)
model2.evaluate(test_dataset)
model3.evaluate(test_dataset)
model4.evaluate(test_dataset)
```

```
6/6 [==============================] - 1s 61ms/step - loss: 2.0109 - accuracy: 0.6146
6/6 [==============================] - 1s 58ms/step - loss: 0.6598 - accuracy: 0.6458
6/6 [==============================] - 0s 31ms/step - loss: 0.5727 - accuracy: 0.7865
6/6 [==============================] - 0s 38ms/step - loss: 0.0871 - accuracy: 0.9896
```

```
[0.0871458426117897, 0.9895833134651184]
```

According to the output, model 4 is the most performant model, which has accuracy 98.96%.