1. a) Import Packages

```
import numpy as np
import pandas as pd
import os
import cv2
import PIL
import tensorflow as tf
import matplotlib.pyplot as mlp
import warnings
import seaborn as sns
from matplotlib.image import imread
import pathlib
from tensorflow import keras
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

1. b) Import Data (Images)

```
# Loading my kaggle profile and downloading the zip file of the images

os.environ["KAGGLE_USERNAME"] = "umar2354"

os.environ["KAGGLE_KEY"] = "c9114a316a26a66568f37bc28aca46fc"

# Download images of cats and dogs

!kaggle datasets download -d chetankv/dogs-cats-images
```

```
Downloading dogs-cats-images.zip to /content
 99% 432M/435M [00:11<00:00, 40.5MB/s]
100% 435M/435M [00:11<00:00, 39.7MB/s]
```

```
# Unzip Files

from zipfile import ZipFile
file_name = "/content/dogs-cats-images.zip"
with ZipFile(file_name, "r") as zip:
  zip.extractall()
  print("done")
```

```
done
```

```
# Checking for file paths, making sure everything loads properly

warnings.filterwarnings("ignore")
data_dir_list = os.listdir("/content/dataset")
print(data_dir_list)
path, dirs, files = next(os.walk("/content/dataset"))
file_count = len(files)
```

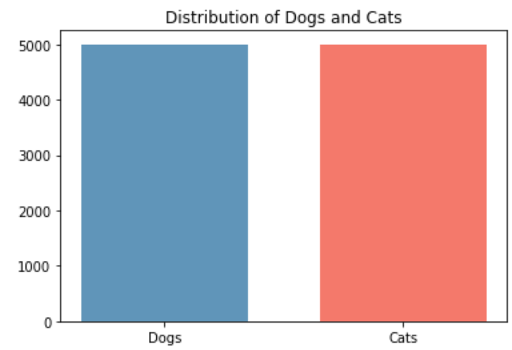
```
['test_set', 'training_set']
```

```
###==============================================###
###  DATASET IS ALREADY SPLIT INTO TRAIN AND TEST  ###
###==============================================###

# Copying paths for train and test

trainDog = "/content/dataset/training_set/dogs"

trainCat = "/content/dataset/training_set/cats"

testDog = "/content/dataset/test_set/dogs"
```

1. c) Describe Dataset

```
# Lengths of dogs and cats from each train and test set

trainDogLen = len(os.listdir("/content/dataset/training_set/dogs"))
trainCatLen = len(os.listdir("/content/dataset/training_set/cats"))
testDogLen = len(os.listdir("/content/dataset/test_set/dogs"))
testCatLen = len(os.listdir("/content/dataset/test_set/dogs"))

dogLen = trainDogLen + testDogLen

catLen = trainCatLen + testCatLen

# Creating Bar graph for distribution

fig, graph = mlp.subplots()
graph.set_title("Distribution of Dogs and Cats")
graph.bar([1, 2], [dogLen, catLen], width = 0.7,
          tick_label = ["Dogs", "Cats"],
          color = ["#6095b9", "#f4796b"],
          align = "center")
```

```
<BarContainer object of 2 artists>
```




```
# Description of dataset

print(
"This is a Dataset with 10,000 images of dogs and cats, 5000 dogs and 5000 cats.\n",
"The data is split into an 80/20 train test split. The model should be able to \n",
"predict the difference between the image of a dog and a cat.\n")
```

```
This is a Dataset with 10,000 images of dogs and cats, 5000 dogs and 5000 cats.
 The data is split into an 80/20 train test split. The model should be able to
 predict the difference between the image of a dog and a cat.
```

2. Sequential Model

```
# Setting standard width and height for images

imgW = 256
imgH = 256
batch = 16


# Fitting images, locating where train data is and idedntifying classes

trainDir = "/content/dataset/training_set"

train_datagen = ImageDataGenerator(rescale = 1/255.0,
                                   rotation_range=30,
                                   zoom_range=0.4,
                                   horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(trainDir,
                                                    batch_size=batch,
```

```
                                                  class_mode="categorical",
                                                  target_size=(imgH, imgW))
```

```
Found 8000 images belonging to 2 classes.
```

```
# Fitting images, locating where test data is and idedntifying classes

testDir = "/content/dataset/test_set"

test_datagen = ImageDataGenerator(rescale = 1/255.0,
                                  rotation_range = 30,
                                  zoom_range = 0.4,
                                  horizontal_flip = True)

test_generator = test_datagen.flow_from_directory(testDir,
                                                  batch_size = batch,
                                                  class_mode = "categorical",
                                                  target_size = (imgH, imgW))
```

```
Found 2000 images belonging to 2 classes.
```

```
# Saving best model

callbacks = EarlyStopping(monitor = "test_loss", patience=5, verbose = 1, mode = "auto")
best_model_file = "/content/CNN_aug_best_weights.h5"
best_model = ModelCheckpoint(best_model_file, monitor = "test_acc", verbose = 1, save_best_only = True)


# Initializing Sequential model

model = Sequential([
    Conv2D(16, (3, 3), activation="relu", input_shape=(imgH, imgW, 3)), MaxPooling2D(2, 2),
    Conv2D(32, (3, 3), activation="relu"), MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation="relu"),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation="relu"),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Conv2D(256, (3, 3), activation="relu"),
    Conv2D(256, (3, 3), activation="relu"),
    Conv2D(256, (3, 3), activation="relu"),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation="relu"),
    Dense(512, activation="relu"),
    Dense(2, activation="softmax")
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 58, 58, 64)        36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 29, 29, 64)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 27, 27, 128)       73856     
                                                                 
 conv2d_5 (Conv2D)           (None, 25, 25, 128)       147584    
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 12, 12, 128)      0         
 2D)                                                             
                                                                 
 conv2d_6 (Conv2D)           (None, 10, 10, 256)       295168    
```

```
 conv2d_7 (Conv2D)           (None, 8, 8, 256)         590080

 conv2d_8 (Conv2D)           (None, 6, 6, 256)         590080

 max_pooling2d_4 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 2)                 1026

=================================================================
Total params: 3,201,122
Trainable params: 3,201,122
Non-trainable params: 0
_________________________________________________________________
```

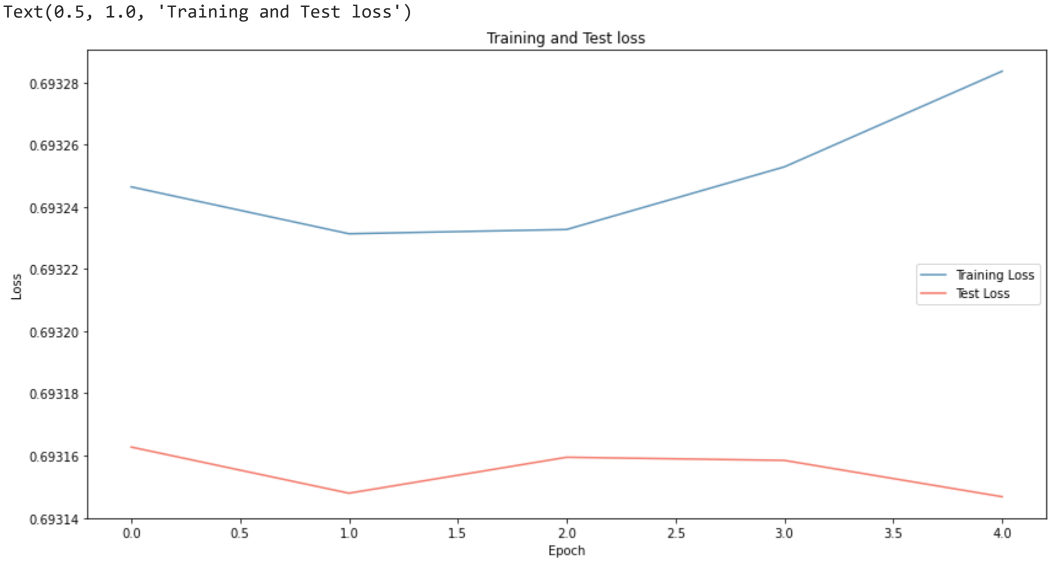
```
# Getting model ready for accuracy

model.compile(optimizer = "Adam",
              loss = "categorical_crossentropy",
              metrics = ["accuracy"])


# Running model

history = model.fit_generator(train_generator,
                              epochs = 5,
                              verbose = 1,
                              validation_data = test_generator,
                              callbacks = [best_model]
                              )
```

```
Epoch 1/5
500/500 [==============================] - ETA: 0s - loss: 0.6940 - accuracy: 0.4985WARNING:tensorflow:Can save best model only with tes
500/500 [==============================] - 647s 1s/step - loss: 0.6940 - accuracy: 0.4985 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 2/5
500/500 [==============================] - ETA: 0s - loss: 0.6933 - accuracy: 0.4925WARNING:tensorflow:Can save best model only with tes
500/500 [==============================] - 638s 1s/step - loss: 0.6933 - accuracy: 0.4925 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 3/5
500/500 [==============================] - ETA: 0s - loss: 0.6933 - accuracy: 0.4972WARNING:tensorflow:Can save best model only with tes
500/500 [==============================] - 634s 1s/step - loss: 0.6933 - accuracy: 0.4972 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 4/5
500/500 [==============================] - ETA: 0s - loss: 0.6934 - accuracy: 0.4857WARNING:tensorflow:Can save best model only with tes
500/500 [==============================] - 631s 1s/step - loss: 0.6934 - accuracy: 0.4857 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 5/5
500/500 [==============================] - ETA: 0s - loss: 0.6932 - accuracy: 0.5008WARNING:tensorflow:Can save best model only with tes
500/500 [==============================] - 630s 1s/step - loss: 0.6932 - accuracy: 0.5008 - val_loss: 0.6932 - val_accuracy: 0.5000
```

```
# Outputting epochs onto a line graph to show the progression of accuracy of the model

acc = history.history["accuracy"]
test_acc = history.history["val_accuracy"]
loss = history.history["loss"]
test_loss = history.history["val_loss"]

epochs = range(len(acc))

fig = mlp.figure(figsize = (14,7))
mlp.plot(epochs, acc, "#6095b9", label = "Training Accuracy")
mlp.plot(epochs, test_acc, "#f4796b", label = "Test Accuracy")
mlp.xlabel("Epoch")
mlp.ylabel("Accuracy")
mlp.title("Training and Test accuracy")
mlp.legend(loc = "lower right")
mlp.show()
```


```
# Outputting Train and Test Loss

fig2 = mlp.figure(figsize=(14,7))
mlp.plot(epochs, loss, "#6095b9", label = "Training Loss")
mlp.plot(epochs, test_loss, "#f4796b", label = "Test Loss")
mlp.legend(loc = 'right')
mlp.xlabel('Epoch')
mlp.ylabel('Loss')
mlp.title('Training and Test loss')
```

```
Text(0.5, 1.0, 'Training and Test loss')
```



3. Applying Convolutional Neural Networks (CNN)

Now we'll the convulutinal neural network architechture to process our input images. These neural networks have excellent performance in image and video recognition, semantic parsing, and phrase identification.

```
# Importing Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.preprocessing.image import ImageDataGenerator
```

```
# Initialize CNN
classifier = Sequential()
```

```
# Orienting images using Keras ImageDataGenerator

train_generator #trainset
test_generator  #testset
```

```
<keras.preprocessing.image.DirectoryIterator at 0x7fa1572b48e0>
```

Applying Convolution, Activation, Pooling to output Final Layer

Convolution involves multiplying weights by input. Multiplication is conducted between input data and filter or kernel weights. ANN learns complicated data patterns with the activation function. Activation functions add nonlinearity to neural networks. Pooling offers spatial variance, allowing the system to recognize objects with different appearances. Pooling reduces network parameters and calculations so it decreases network size to prevent overfitting.

```
# Import the Sequential model and layers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(256, 256, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding = 'same'))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('sigmoid'))

model.compile(loss = 'binary_crossentropy',
              optimizer = 'Adam',
              metrics = ['accuracy'])

batch_size = 16
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 254, 254, 32)      896

 activation_10 (Activation)  (None, 254, 254, 32)      0

 max_pooling2d_6 (MaxPooling  (None, 127, 127, 32)     0
 2D)

 conv2d_7 (Conv2D)           (None, 125, 125, 64)      18496

 activation_11 (Activation)  (None, 125, 125, 64)      0

 max_pooling2d_7 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_8 (Conv2D)           (None, 60, 60, 128)       73856

 activation_12 (Activation)  (None, 60, 60, 128)       0

 max_pooling2d_8 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 flatten_2 (Flatten)         (None, 115200)            0
```

```
 dense_4 (Dense)             (None, 64)                7372864

 activation_13 (Activation)  (None, 64)                0

 dropout_2 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 2)                 130

 activation_14 (Activation)  (None, 2)                 0

=================================================================
Total params: 7,466,242
Trainable params: 7,466,242
Non-trainable params: 0
_________________________________________________________________
```

Fitting Model to Training Set

```
# Fitting model to Training Set
model.fit_generator(train_generator,
                    epochs = 5,
                    validation_data = test_generator,
                    verbose = 1)

# Evaluating model performance on Testing data
loss, accuracy = model.evaluate(test_generator)

print("\nModel's Evaluation Metrics: ")
print("---------------------------")
print("Accuracy: {} \nLoss: {}".format(accuracy, loss))
```

```
Epoch 1/5
500/500 [==============================] - 155s 309ms/step - loss: 0.6963 - accuracy: 0.5307 - val_loss: 0.6854 - val_accuracy: 0.5250
Epoch 2/5
500/500 [==============================] - 153s 306ms/step - loss: 0.6628 - accuracy: 0.5855 - val_loss: 0.6333 - val_accuracy: 0.6120
Epoch 3/5
500/500 [==============================] - 153s 307ms/step - loss: 0.6462 - accuracy: 0.6175 - val_loss: 0.6308 - val_accuracy: 0.6625
Epoch 4/5
500/500 [==============================] - 153s 307ms/step - loss: 0.6284 - accuracy: 0.6475 - val_loss: 0.6127 - val_accuracy: 0.6700
Epoch 5/5
500/500 [==============================] - 153s 307ms/step - loss: 0.6260 - accuracy: 0.6549 - val_loss: 0.5778 - val_accuracy: 0.6955
125/125 [==============================] - 30s 237ms/step - loss: 0.5743 - accuracy: 0.7015

Model's Evaluation Metrics:
---------------------------
Accuracy: 0.7014999985694885
Loss: 0.5743228793144226
```

4. Transfer Learning

```
# Using Resnet50 model for pretrained model

resnet_model = Sequential()

# Getting the model ready for our previously selected image dimensions and classes

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(imgH, imgW, 3),
                   pooling="avg",classes=2,
                   weights="imagenet")
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)


# Outputting model summary

resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation = "relu"))
resnet_model.add(Dense(2, activation = "softmax"))
resnet_model.summary()
```

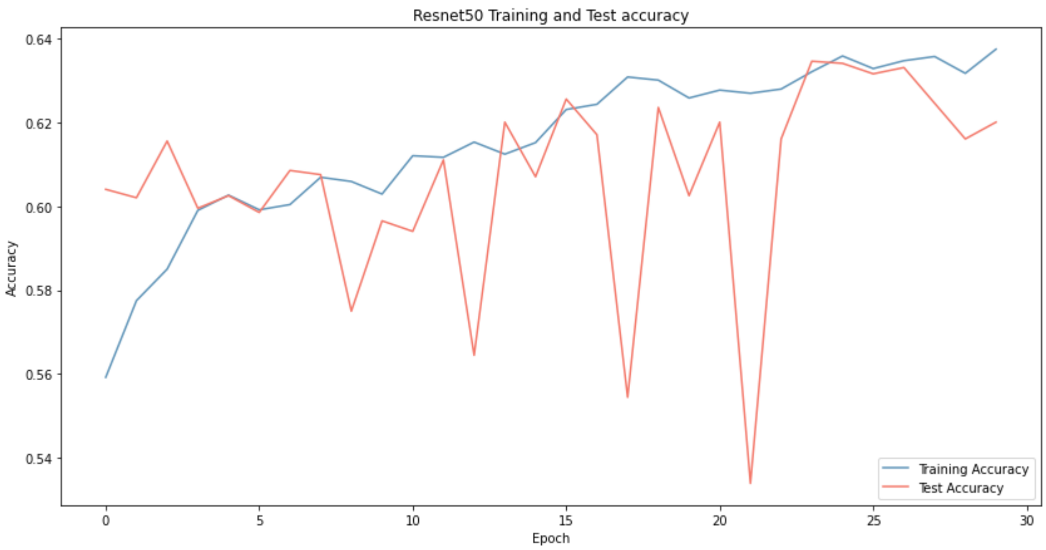
```
Model: "sequential_4"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten_5 (Flatten)         (None, 2048)              0

 dense_11 (Dense)            (None, 512)               1049088

 dense_12 (Dense)            (None, 2)                 1026

=================================================================
Total params: 24,637,826
Trainable params: 1,050,114
Non-trainable params: 23,587,712
_________________________________________________________________
```

```
# Running Resnet50 Model

resnet_model.compile(optimizer = Adam(lr = 0.001),
                     loss = "categorical_crossentropy", metrics = ["accuracy"])

historyTL = resnet_model.fit(train_generator, validation_data = test_generator, epochs = 30)
```

```
Epoch 1/30
500/500 [==============================] - 179s 350ms/step - loss: 0.7222 - accuracy: 0.5592 - val_loss: 0.6626 - val_accuracy: 0.604
Epoch 2/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6787 - accuracy: 0.5775 - val_loss: 0.6634 - val_accuracy: 0.602
Epoch 3/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6693 - accuracy: 0.5850 - val_loss: 0.6640 - val_accuracy: 0.615
Epoch 4/30
500/500 [==============================] - 173s 346ms/step - loss: 0.6645 - accuracy: 0.5990 - val_loss: 0.6626 - val_accuracy: 0.599
Epoch 5/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6650 - accuracy: 0.6026 - val_loss: 0.6624 - val_accuracy: 0.602
Epoch 6/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6626 - accuracy: 0.5991 - val_loss: 0.6655 - val_accuracy: 0.598
Epoch 7/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6635 - accuracy: 0.6004 - val_loss: 0.6576 - val_accuracy: 0.608
Epoch 8/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6600 - accuracy: 0.6069 - val_loss: 0.6580 - val_accuracy: 0.607
Epoch 9/30
500/500 [==============================] - 173s 347ms/step - loss: 0.6606 - accuracy: 0.6059 - val_loss: 0.6805 - val_accuracy: 0.575
Epoch 10/30
500/500 [==============================] - 173s 346ms/step - loss: 0.6645 - accuracy: 0.6029 - val_loss: 0.6659 - val_accuracy: 0.596
Epoch 11/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6545 - accuracy: 0.6120 - val_loss: 0.6695 - val_accuracy: 0.594
Epoch 12/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6575 - accuracy: 0.6116 - val_loss: 0.6605 - val_accuracy: 0.611
Epoch 13/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6551 - accuracy: 0.6152 - val_loss: 0.7004 - val_accuracy: 0.564
Epoch 14/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6523 - accuracy: 0.6124 - val_loss: 0.6581 - val_accuracy: 0.620
Epoch 15/30
500/500 [==============================] - 175s 350ms/step - loss: 0.6534 - accuracy: 0.6151 - val_loss: 0.6611 - val_accuracy: 0.607
Epoch 16/30
500/500 [==============================] - 173s 345ms/step - loss: 0.6476 - accuracy: 0.6230 - val_loss: 0.6517 - val_accuracy: 0.625
Epoch 17/30
500/500 [==============================] - 173s 347ms/step - loss: 0.6455 - accuracy: 0.6242 - val_loss: 0.6533 - val_accuracy: 0.617
Epoch 18/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6437 - accuracy: 0.6308 - val_loss: 0.6988 - val_accuracy: 0.554
Epoch 19/30
500/500 [==============================] - 173s 347ms/step - loss: 0.6448 - accuracy: 0.6300 - val_loss: 0.6530 - val_accuracy: 0.623
Epoch 20/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6452 - accuracy: 0.6258 - val_loss: 0.6528 - val_accuracy: 0.602
Epoch 21/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6438 - accuracy: 0.6276 - val_loss: 0.6502 - val_accuracy: 0.620
Epoch 22/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6455 - accuracy: 0.6269 - val_loss: 0.7154 - val_accuracy: 0.534
Epoch 23/30
500/500 [==============================] - 174s 349ms/step - loss: 0.6450 - accuracy: 0.6279 - val_loss: 0.6623 - val_accuracy: 0.616
Epoch 24/30
500/500 [==============================] - 173s 347ms/step - loss: 0.6405 - accuracy: 0.6320 - val_loss: 0.6412 - val_accuracy: 0.634
Epoch 25/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6408 - accuracy: 0.6357 - val_loss: 0.6416 - val_accuracy: 0.634
Epoch 26/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6391 - accuracy: 0.6327 - val_loss: 0.6427 - val_accuracy: 0.631
Epoch 27/30
500/500 [==============================] - 174s 347ms/step - loss: 0.6397 - accuracy: 0.6346 - val_loss: 0.6395 - val_accuracy: 0.633
Epoch 28/30
500/500 [==============================] - 174s 348ms/step - loss: 0.6381 - accuracy: 0.6356 - val_loss: 0.6381 - val_accuracy: 0.624
Epoch 29/30
```

```
# Outputting epochs onto a line graph to show the progression of accuracy of the model

accTL = historyTL.history["accuracy"]
test_accTL = historyTL.history["val_accuracy"]
lossTL = historyTL.history["loss"]
test_lossTL = historyTL.history["val_loss"]

epochs = range(len(accTL))

fig = mlp.figure(figsize = (14,7))
mlp.plot(epochs, accTL, "#6095b9", label = "Training Accuracy")
mlp.plot(epochs, test_accTL, "#f4796b", label = "Test Accuracy")
mlp.xlabel("Epoch")
mlp.ylabel("Accuracy")
mlp.title("Resnet50 Training and Test accuracy")
mlp.legend(loc = "lower right")
mlp.show()
```




```
# Outputting Train and Test Loss

fig2 = mlp.figure(figsize=(14,7))
mlp.plot(epochs, lossTL, "#6095b9", label = "Training Loss")
mlp.plot(epochs, test_lossTL, "#f4796b", label = "Test Loss")
mlp.legend(loc = 'right')
mlp.xlabel('Epoch')
mlp.ylabel('Loss')
mlp.title('Resnet50 Training and Test loss')
```

```
Text(0.5, 1.0, 'Resnet50 Training and Test loss')
```

5. Summary of Approaches

The initial model was a sequential model, which is a very simple and straightforward architecture in which the layers are arranged sequentially, but is confined to single-input, single-output layer stacks. In terms of test accuracy, this model produced a steady output; however, training accuracy results were lower. While the training loss seemed to peak on the fourth epoch, resulting in a training loss of 0.69328, the test loss began high, just over 0.69316, then decreased on the fourth epoch, yielding output between 0.69314 and 0.69316.

We utilized the convolution neural network architecture in subsequent evaluations. This architecture is used primarily for image classification, and its variants are also scalable for huge datasets. In the CNN model description, it is evident that the output of each Conv2D and MaxPooling2D layer is a 3D form tensor (height, width, channels). As you move deeper into the network, the width and height proportions begin to diminish. The first argument determines the quantity of output channels for each Conv2D layer. Typically, it is preferable to increase the number of output channels in each Conv2D layer as the width and height decrease. In addition, to complete the model, the final output tensor from the convolutional base is fed into one or more Dense layers for classification. Dense layers accept vectors (1D) as input, but the current output is a 3D tensor. The dataset contains two output classes, hence a final Dense layer with two outputs is used. The training accuracy of this model was 0.70, while the test accuracy was 0.69. In addition, the training loss was 0.57, whereas the test loss was also 0.57.

In our transfer learning model, Resnet50 was used as a pre-trained model. Resnet50 is a CNN model variant consisting of 50 layers. The skip connection is the most innovative aspect of ResNet. As you may be aware, without changes, deep neural networks frequently exhibit vanishing gradients. However, the architecture enables the network to learn the identity function, allowing it to send the input via the block without traversing the other weight layers. This enables you to stack additional layers and construct a deeper network, mitigating the vanishing gradient by allowing your network to bypass training levels it deems less important. On the 30th epoch, this design yields a training accuracy of 0.64 and a test accuracy of 0.62. In addition, the training loss output reduced from 0.72 to 0.64, and the test loss output decreased from 0.66 to 0.65.