```
# Weeek4 - Homework : Chest X-Ray Image Classification (Normal or Pneumonia) using Transfer Learning
#1- Download Chest X-Ray Image Data From Kaggle
#2- Plot Normal and Pneumonia Chest X-Ray Images
#3- Chest X-Ray Image Preprocessing/ Augmentation/ Transformation for Model Training, Validation and Testing Dataset
#4- Create InceptionV3 Base CNN Model
#5- Create a Sequential Model for traing & prediction. Add base model, pooling layer & fully connected Dense layers
#6- Compile the Sequential Model and Display Model Summary
#7- Fit and Train the Sequential Model and Display Results
#8- Evaluate the Sequential Model Performance and Show Results
#9- Test the pre-trained Sequential Model and Show Results
# Work below is based on Anjana Tiha github project at https://github.com/anjanatiha/Pneumonia-Detection-from-Chest-X-Ray-Images-with-D
import os #for OS dependent functionalities
from os import listdir, makedirs, getcwd, remove
import numpy as np # data presprocessing
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt # visualize the data
import matplotlib.image as mimg
import imgaug.augmenters as iaa
import math, json
import datetime
import random
import re
import cv2
from PIL import Image  #Python Imaging Library
from pathlib import Path
from glob import glob # find pathnames for images matching specified pattern
import keras
# Inception-v3 is convolution neural network trained on million+ images from ImageNet db.
# The network is 48 layers deep and can classify images into 1000 object categories
# The Inception-v3 model expects color images to have the square shape 299×299.
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.image import ImageDataGenerator #to perform image augmentation on the fly & easy way
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.utils import plot_model
from keras import optimizers
from keras.optimizers import Adam, SGD , RMSprop
from keras import backend as K
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
import torch # to save model etc on google colab drive
from google.colab import drive # to save model etc on google colab drive

#!pip install tensorflow --upgrade
#!pip install keras --upgrade

print("Tensorflow version " + tf.__version__)

print("Keras version " + keras.__version__)

print("Numpy version " + np.__version__)


#################### Download Chest X-Ray Image Data From Kaggle ##############################

print("\n" + '\033[1m \033[4mDownload Chest X-Ray Data From Kaggle\033[0m' + "\n")
# Setup the Kaggle environemt for image download
#os.environ['KAGGLE_USERNAME'] = 'XXXX'
#os.environ['KAGGLE_KEY'] = 'XXXXXXX'
# Dowmload the Kaggle Chest X-Ray dataset
#!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia
# unzip the chest x-ray images
#!unzip chest-xray-pneumonia.zip
#!unzip chest_xray.zip

#################### Plot Normal and Pneumonia Chest X-Ray Images ##############################

print("\n" + '\033[1m \033[4mPlot Normal and Pneumonia Chest X-Ray Images\033[0m' + "\n")

# Define and configure file directories for train, validate and test image dataset
training_dir = './chest_xray/train'
validate_dir = './chest_xray/val'
test_dir = './chest_xray/test'

# Plot an image of Normal Chest Xray
plt.figure(1, figsize = (12 , 6))
plt.subplot(1 , 2 , 1)
print('Number of Normal Images: ' + str(len(glob(training_dir+"/NORMAL/*.jpeg"))))
img = np.asarray(plt.imread(glob(training_dir+"/NORMAL/*.jpeg")[0]))
plt.title('Normal Chest X-Ray')
plt.imshow(img)

# Plot an image of Pneumonia Chest Xray
plt.subplot(1 , 2 , 2)
img = np.asarray(plt.imread(glob(training_dir+"/PNEUMONIA/*.jpeg")[0]))
```

```
print('Number of Pneumonia Images: '  + str(len(glob(training_dir+"/PNEUMONIA/*.jpeg"))))
print("")
plt.title('Pneumonia Chest X-Ray')
plt.imshow(img)
plt.show()

######### Chest X-Ray Image Preprocessing/ Augmentation/ Transformation for Model Training, Validation and Testing Dataset ##########

print("\n" + '\033[1m \033[4mImage Preprocessing/ Augmentation/ Transformation for Training, Validation and Testing Dataset\033[0m' + "

# Define Constants used
# Rescale the pixel values (between 0 and 255) to [0, 1] interval
# Neural networks perform better with normalize data.
# rescale normalizes the image pixel values to have zero mean & standard deviation of 1.
rescale = 1./255 # Scale the image between 0 and 1
target_size = (150, 150)  # by keras doc, conv2d input shape should be (height, width) == (row, colum) sequence
batch_size=32
img_height=150
img_width=150

# Use ImageDataGenerator() so Python generators that automatically turn image files into preprocessed tensors that can be fed directly
#1. Decode the JPEG content to RGB grids of pixels.
#2. Convert these into floating-point tensors.
#3. Rescale the pixel values (between 0 and 255) to the [0, 1] interval ( neural networks perform better with normalize data).
#4. Helps us easily augment images.

# Create Training Image Data Generator and augument the training images to augment our data-set and improve generalization.
training_datagen = ImageDataGenerator(
    rescale=rescale,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

print('Training Data Generation')
# this is the augmentation configuration we will use for training
training_generator = training_datagen.flow_from_directory(
    training_dir,
    target_size=target_size,
    class_mode='binary',
    batch_size=32
)
print('Training classes are' + str(training_generator.class_indices) + '\n')


# Create Vaildation Image Data Generator
# No image data augumentation for validation image data generation, only rescale
validation_datagen = ImageDataGenerator(rescale=rescale)
print('\n' + 'Validation Data Generation')
# this is the augmentation configuration we will use for validation
validation_generator = validation_datagen.flow_from_directory(
    validate_dir,
    target_size=target_size,
    class_mode='binary',
    batch_size=32
)
print('Validation classes are' + str(validation_generator.class_indices) + '\n')

# Create Testing Image Data Generator
test_datagen = ImageDataGenerator(rescale=rescale)
print('\n' + 'Test Data Generation')
# this is the augmentation configuration we will use for testing
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=target_size,
    class_mode='binary',
    batch_size=1
)
print('Test classes are' + str(test_generator.class_indices))

######################## Create InceptionV3 Base CNN Model ###################################

print("\n" + '\033[1m \033[4mCreating Pre-Trained InceptionV3 Base Model For Transfer Learning\033[0m' + "\n")

# A Convolutional Neural Network (CNN) architecture has three main parts:
# A convolutional layer that extracts features from a source image. Convolution helps with blurring, sharpening, edge detection, noise
# A pooling layer that reduces the image dimensionality without losing important features or patterns.
# A fully connected layer also known as the dense layer, in which the results of the convolutional layers are fed through one or more n

# Create the base pre-trained InceptionV3 CNN model for image processing
# “include_top” argument set to False so fully-connected output layers of the model
# used to make predictions is not loaded, allowing a new output layer to be added and trained.
# A model without a top will output activations from the last convolutional or pooling layer directly.
# Your input should be input_shape=(img_rows, img_cols, 3 channels RGB)  or (batchSize, height, width, feature/channels)
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

######################## Create a Sequential Model for traing & prediction. Add base model, pooling layer & fully connected Dense layer
# Initializing the network using the Sequential Class
```

```
# The Sequential model (a linear stack of layers) is a prediction model which is trained with a set of training sequences.
# Once trained, the sequential model is used to perform sequence predictions.
# Sequential class which is a linear stack of Layers where after creating you can define all of the layers in the constructor
model = Sequential()
# Add the convolution layer by adding the pre-trained InceptionV3 base model output
model.add(base_model)
# Add a pooling layer by adding global spatial average pooling layer
model.add(keras.layers.GlobalAveragePooling2D())
# Add a fully-connected layer;
model.add(Dense(1024, activation='relu'))  # rectified linear unit
# Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent ove
model.add(Dropout(0.5))
# Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close
model.add(keras.layers.BatchNormalization())
# Model output arrays will be of shape (*, 8)
model.add(Dense(8, activation='softmax'))
# Add logistic layer
# Using 1 class for binary classification with activation sigmoid
# Model output arrays will be of shape (*, 1)
model.add(Dense(1, activation='sigmoid'))

######################### Compile the Model and Display Model Summary ####################

print('Compiling The Model' + "\n")

model.compile(loss= 'binary_crossentropy',
              optimizer='rmsprop',          # root mean square propagation
              metrics=['accuracy'])

print('Displaying The Model Summary Information below' + "\n")
model.summary()
print("")

######################### Fit and Train the Sequential Model and Display Results ###################################

print("\n" + '\033[1m \033[4mStarting to Fit and Train The Model\033[0m' + "\n")

# Train the model on data generated batch-by-batch by a Python generator
# Train the model for defined epochs for steps_per_epoch times in each epoch
# An epoch usually means one iteration over all of the training data.
# For instance for 20,000 images and a batch size of 100, the epoch should contain 20,000 / 100 = 200 steps.

steps_per_epoch=len(training_generator)
print('steps_per_epoch is ' + str(steps_per_epoch) + '\n')
validation_steps=len(validation_generator)
print('validation_steps is ' + str(validation_steps) + '\n')

history = model.fit_generator(
    training_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=5,
    verbose=1,
    validation_data=validation_generator,
    validation_steps=validation_steps,
)

# Save the Weights and the Model
print("")
drive.mount('/content/gdrive')   # if drive is dismounted then will need to forget previous authentication key and reauthenticate
model_save_name = 'ChestXRayClassifier.pt'
model_path_filename = F"/content/gdrive/My Drive/{model_save_name}"
print('\n' + 'Saving the trained model at %s ' % path)
torch.save(model, model_path_filename)   #torch.save(model.state_dict(), path) gives error

# Plot the Model and its layers
path = F"/content/gdrive/My Drive"
plot_model(model, to_file='chest-xray-classification-model.png')
model_img = Image.open("chest-xray-classification-model.png")
plt.figure(1, figsize = (16 , 16))
plt.title('Trained Model')
plt.imshow(model_img)

print('\n' "Completed Model Fitting and Training" '\n')

######################### Evaluate Model Performance ##############################

print("\n" + '\033[1m \033[4mPerform and Plot Model Performance Evaluation\033[0m' + "\n")

xlabel = 'Epoch'
legends = ['Training', 'Validation']

ylim_pad = [0.01, 0.1]

plt.figure(figsize=(15, 5))

# Plot training & validation Accuracy values

y1 = history.history['acc']
y2 = history.history['val_acc']

```

```
min_y = min(min(y1), min(y2))-ylim_pad[0]
max_y = max(max(y1), max(y2))+ylim_pad[0]

plt.subplot(121)

plt.plot(y1)
plt.plot(y2)

plt.title('Model Accuracy', fontsize=17)
plt.xlabel(xlabel, fontsize=15)
plt.ylabel('Accuracy', fontsize=15)
plt.ylim(min_y, max_y)
plt.legend(legends, loc='upper left')
plt.grid()

# Plot training & validation loss values

y1 = history.history['loss']
y2 = history.history['val_loss']

min_y = min(min(y1), min(y2))-ylim_pad[1]
max_y = max(max(y1), max(y2))+ylim_pad[1]

plt.subplot(122)

plt.plot(y1)
plt.plot(y2)

plt.title('Model Loss', fontsize=17)
plt.xlabel(xlabel, fontsize=15)
plt.ylabel('Loss', fontsize=15)
plt.ylim(min_y, max_y)
plt.legend(legends, loc='upper left')
plt.grid()

plt.show()

######################### Test the pre-trained Sequential Model and Show Results ##############################

print("\n" + '\033[1m \033[4mTesting The Trained Model\033[0m' + "\n")

test_steps=len(test_generator)
results = model.evaluate_generator(generator=test_generator, steps=test_steps, verbose=1)
print('\n' + 'Trained Model Testing Results: Loss and Accuracy: ' + str(results))

history = model.fit_generator(
    training_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=5,
    verbose=1,
    validation_data=validation_generator,
    validation_steps=validation_steps,
)

# Save the Weights and the Model
print("")
drive.mount('/content/gdrive')   # if drive is dismounted then will need to forget previous authentication key and reauthenticate
model_save_name = 'ChestXRayClassifier.pt'
model_path_filename = F"/content/gdrive/My Drive/{model_save_name}"
print('\n' + 'Saving the trained model at %s ' % path)
torch.save(model, model_path_filename)   #torch.save(model.state_dict(), path) gives error

# Plot the Model and its layers
path = F"/content/gdrive/My Drive"
plot_model(model, to_file='chest-xray-classification-model.png')
model_img = Image.open("chest-xray-classification-model.png")
plt.figure(1, figsize = (16 , 16))
plt.title('Trained Model')
plt.imshow(model_img)

print('\n' "Completed Model Fitting and Training" '\n')

######################### Evaluate Model Performance ##############################

print("\n" + '\033[1m \033[4mPerform and Plot Model Performance Evaluation\033[0m' + "\n")

xlabel = 'Epoch'
legends = ['Training', 'Validation']

ylim_pad = [0.01, 0.1]

plt.figure(figsize=(15, 5))

# Plot training & validation Accuracy values

y1 = history.history['acc']
y2 = history.history['val_acc']

min_y = min(min(y1), min(y2))-ylim_pad[0]
max_y = max(max(y1), max(y2))+ylim_pad[0]
```

```
359
360 plt.subplot(121)
361
362 plt.plot(y1)
363 plt.plot(y2)
364
365 plt.title('Model Accuracy', fontsize=17)
366 plt.xlabel(xlabel, fontsize=15)
367 plt.ylabel('Accuracy', fontsize=15)
368 plt.ylim(min_y, max_y)
369 plt.legend(legends, loc='upper left')
370 plt.grid()
371
372 # Plot training & validation loss values
373
374 y1 = history.history['loss']
375 y2 = history.history['val_loss']
376
377 min_y = min(min(y1), min(y2))-ylim_pad[1]
378 max_y = max(max(y1), max(y2))+ylim_pad[1]
379
380 plt.subplot(122)
381
382 plt.plot(y1)
383 plt.plot(y2)
384
385 plt.title('Model Loss', fontsize=17)
386 plt.xlabel(xlabel, fontsize=15)
387 plt.ylabel('Loss', fontsize=15)
388 plt.ylim(min_y, max_y)
389 plt.legend(legends, loc='upper left')
390 plt.grid()
391
392 plt.show()
393
394 ######################## Test the trained Model ##################################
395
396 print("\n" + '\033[1m \033[4mTesting The Trained Model\033[0m' + "\n")
397
398 test_steps=len(test_generator)
399 results = model.evaluate_generator(generator=test_generator, steps=test_steps, verbose=1)
400 print('\n' + 'Trained Model Testing Results: Loss and Accuracy: ' + str(results))
```

Tensorflow version 1.14.0
Keras version 2.2.5
Numpy version 1.16.5

Download Chest X-Ray Data From Kaggle

Plot Normal and Pneumonia Chest X-Ray Images

Number of Normal Images: 1341
Number of Pneumonia Images: 3875

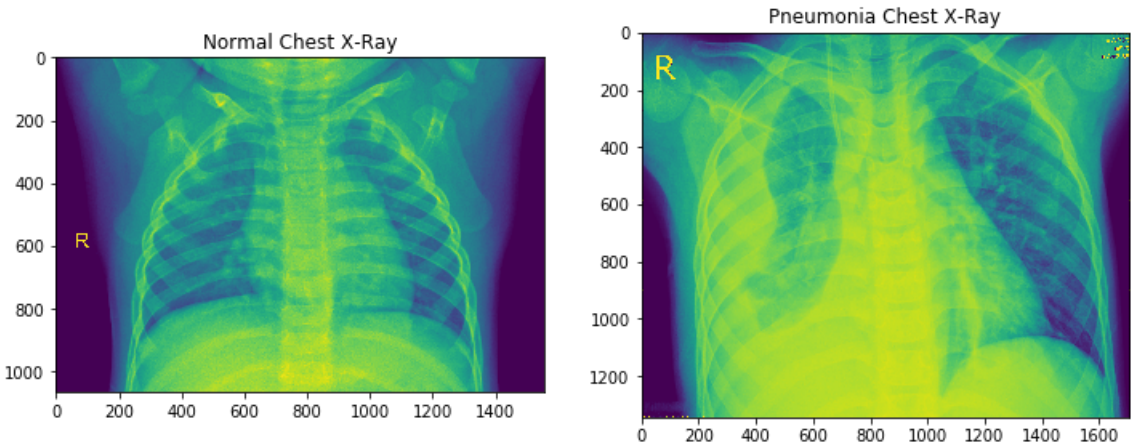


Image Preprocessing/ Augmentation/ Transformation for Training, Validation and Testing Dataset

Training Data Generation
Found 5216 images belonging to 2 classes.
Training classes are{'NORMAL': 0, 'PNEUMONIA': 1}

Validation Data Generation
Found 16 images belonging to 2 classes.
Validation classes are{'NORMAL': 0, 'PNEUMONIA': 1}

Test Data Generation
Found 624 images belonging to 2 classes.
Test classes are{'NORMAL': 0, 'PNEUMONIA': 1}

Creating Pre-Trained InceptionV3 Base Model For Transfer Learning

Compiling The Model

Displaying The Model Summary Information below

Model: "sequential_17"

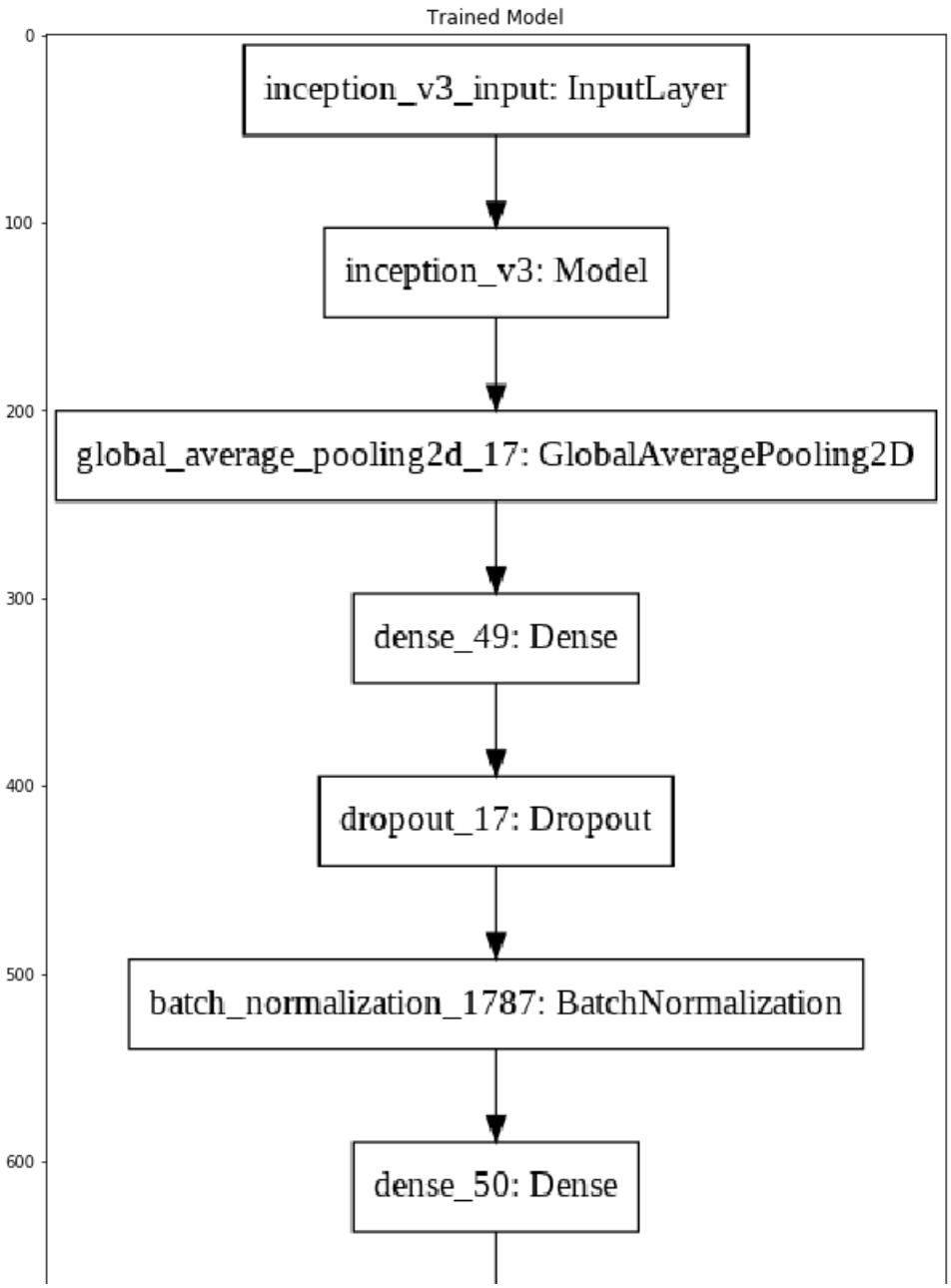
Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, 3, 3, 2048)	21802784
global_average_pooling2d_17	(None, 2048)	0
dense_49 (Dense)	(None, 1024)	2098176
dropout_17 (Dropout)	(None, 1024)	0
batch_normalization_1787 (Ba	(None, 1024)	4096
dense_50 (Dense)	(None, 8)	8200
dense_51 (Dense)	(None, 1)	9

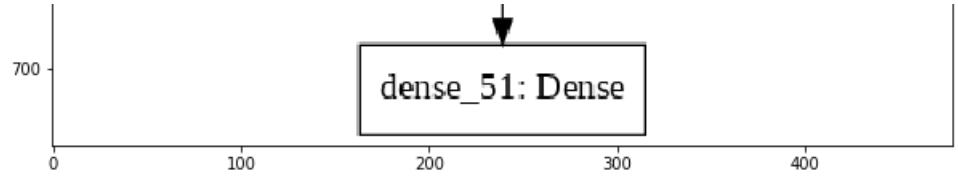
Total params: 23,913,265
Trainable params: 23,876,785
Non-trainable params: 36,480

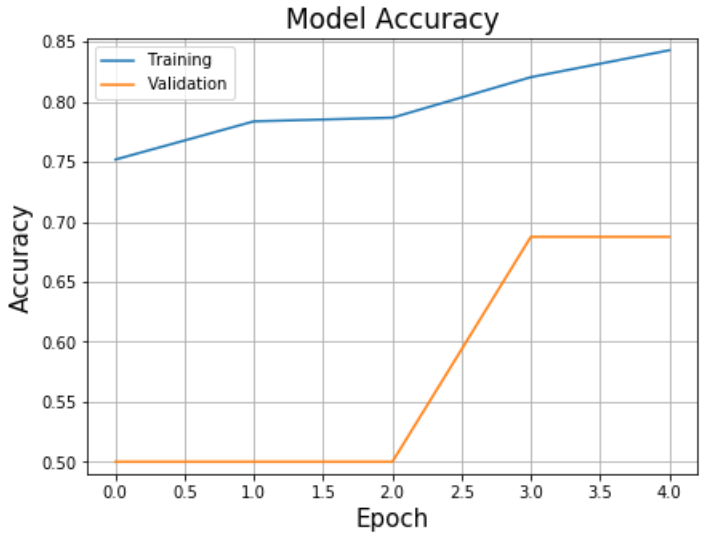
Starting to Fit and Train The Model

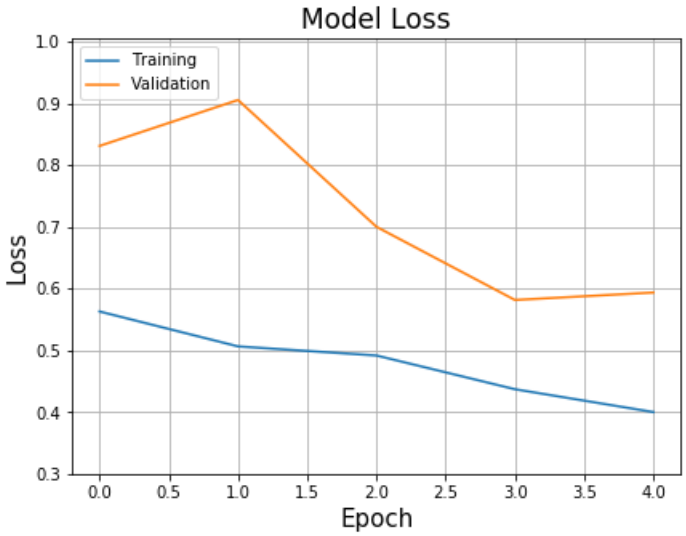
```
steps_per_epoch is 163

validation_steps is 1

Epoch 1/5
163/163 [==============================] - 836s 5s/step - loss: 0.5630 - acc: 0.7519 - val_loss: 0.8313 - val_acc: 0.5000
Epoch 2/5
163/163 [==============================] - 760s 5s/step - loss: 0.5065 - acc: 0.7837 - val_loss: 0.9058 - val_acc: 0.5000
Epoch 3/5
163/163 [==============================] - 756s 5s/step - loss: 0.4917 - acc: 0.7868 - val_loss: 0.7002 - val_acc: 0.5000
Epoch 4/5
163/163 [==============================] - 759s 5s/step - loss: 0.4371 - acc: 0.8206 - val_loss: 0.5817 - val_acc: 0.6875
Epoch 5/5
163/163 [==============================] - 765s 5s/step - loss: 0.4001 - acc: 0.8430 - val_loss: 0.5938 - val_acc: 0.6875

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=T

Saving the trained model at /content/gdrive/My Drive

Completed Model Fitting and Training
```

Perform and Plot Model Performance Evaluation









Testing The Trained Model

```
624/624 [==============================] - 60s 96ms/step

Trained Model Testing Results: Loss and Accuracy: [0.5722941191723714, 0.6634615384615384]
Epoch 1/5
 26/163 [===>..........................] - ETA: 11:26 - loss: 0.4076 - acc: 0.8269
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-22-a73bc0487cf3> in <module>()
    307     verbose=1,
    308     validation_data=validation_generator,
--> 309     validation_steps=validation_steps,
    310 )
    311

                                    6 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/client/session.py in __call__(self, *args, **kwargs)
   1456         ret = tf_session.TF_SessionRunCallable(self._session._session,
   1457                                                self._handle, args,
-> 1458                                                run_metadata_ptr)
   1459         if run_metadata:
   1460           proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
1
```

Double-click (or enter) to edit