DEEP LEARNING ASSIGNMENT

Computer Vision With Transfer Learning

Akash.E
2018103005
CSE - 'P' Batch
11/05/2021

CS-6005 DEEP LEARNING ASSIGNMENT - 4

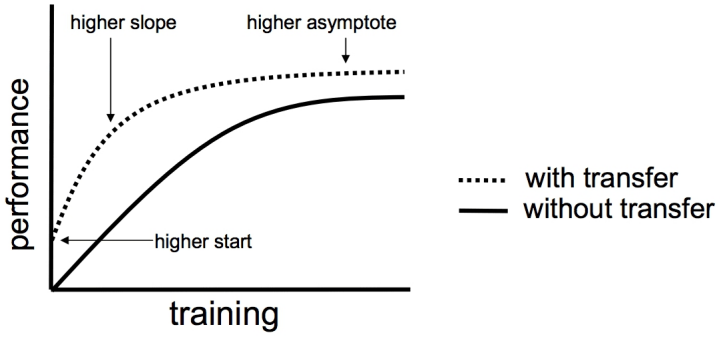
AKASH.E - (2018103005)

COMPUTER VISION WITH TRANSFER LEARNING

1. INTRODUCTION:

In this project we try to classify Chest X-Ray(CXR) images as normal or pneunomia affected using computer vision and transfer learning.

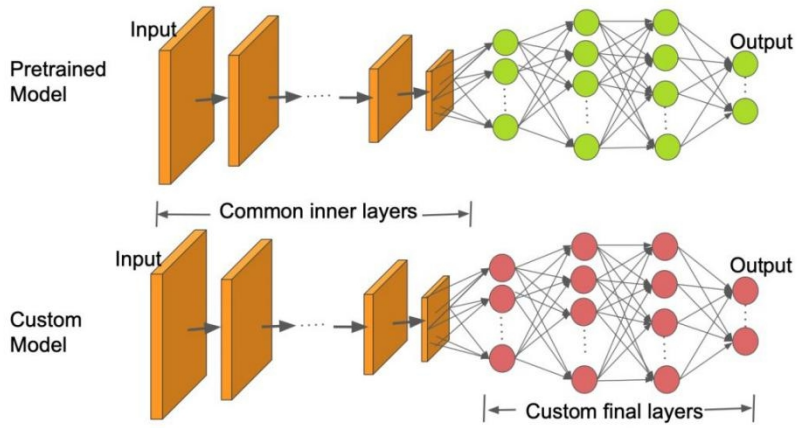
Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. Transfer learning is an optimization that allows rapid progress or improved performance when modeling the second task.



You can use transfer learning on your own predictive modeling problems.Two common approaches are as follows:

1. Develop Model Approach
2. Pre-trained Model Approach

In the develop model approach, we develop a source model first and then use that model to accomplish another task.But, in the pre-trained model approach we use the already pre-trained models that are available for use. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.



2. VGG-19 :

VGG-19 is a trained Convolutional Neural Network, from Visual Geometry Group, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The column E refers to VGG-19 architecture. The VGG-19 was trained on the ImageNet challenge (ILSVRC) 1000-class classification task. The network takes a (224, 224, 3) RBG image as the input.conv3-64 means 64 (3, 3) square filters. Note that all the conv layers in VGG-19 use (3, 3) filters and that the number of filters increases in powers of two (64, 128, 256, 512)

Architecture Description :

- A fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- Spatial padding was used to preserve the spatial resolution of the image.
- Max pooling was performed over a 2 * 2 pixel windows with sride 2.
- This was followed by Rectified linear unit(ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
- Implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way ILSVRC classification and the final layer is a softmax function.

The layers with trainable weights are only the convolution layers and the Fully Connected layers. maxpool layer is used to reduce the size of the input image where softmax is used to make the final decision.

3. DATASET :

The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans.

4. MODULE LIST :

1. Fetching the data and visualizing it.
2. Data Augmentation using Image Data Generator.
3. Import VGG-19 and Model Building.
4. Training the model.
5. Visualizing Results.

5. MODULE EXPLANATION :

1. Fetching the data and visualizing it :

In this module we import the necessary libraries and fetch the dataset from kaggle and visualize how it looks.

[2]
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import random
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img
```

[3]
```
os.listdir("/kaggle/input/chest-xray-pneumonia/chest_xray/train")
```

```
['PNEUMONIA', 'NORMAL']
```

[4]
```
train_dir = "/kaggle/input/chest-xray-pneumonia/chest_xray/train"
test_dir = "/kaggle/input/chest-xray-pneumonia/chest_xray/test"
valid_dir = "/kaggle/input/chest-xray-pneumonia/chest_xray/val"
```

[5]
```
alltype = ["NORMAL","PNEUMONIA"]
normal = random.sample(os.listdir(train_dir+"/NORMAL"),5)
pneumonia = random.sample(os.listdir(train_dir+"/PNEUMONIA"),5)
```

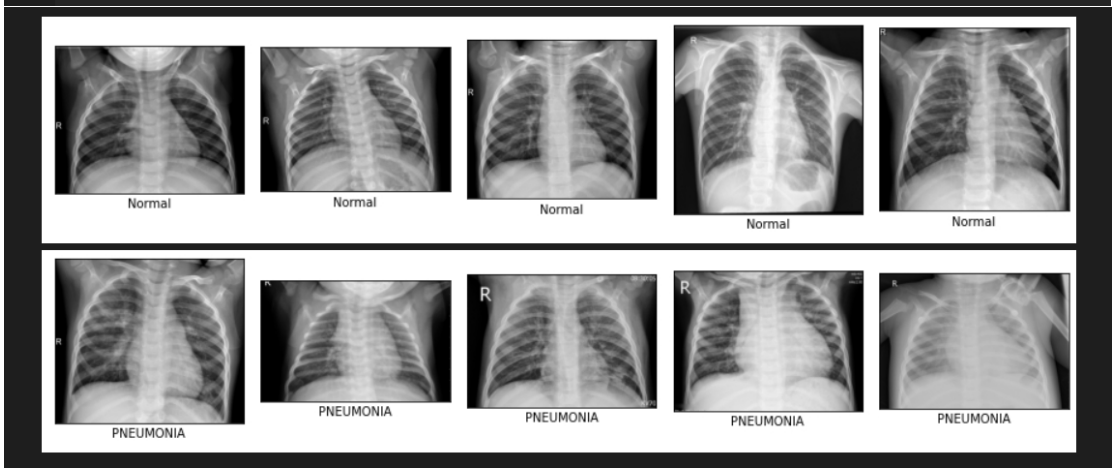
[6]
```
from tqdm import tqdm
import cv2
x_train =[]
y_train = []
#os.chdir = "/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL"
for i in tqdm(os.listdir("/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL")):
    img = cv2.imread("/kaggle/input/chest-xray-pneumonia/chest_xray/train/NORMAL" +"/"+i)
    img = cv2.resize(img,(256,256))
    x_train.append(img)
    y_train.append("Normal")
```

```
100%|██████████| 1341/1341 [00:46<00:00, 29.09it/s]
```

[7]
```
for i in tqdm(os.listdir("/kaggle/input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA")):
    img = cv2.imread("/kaggle/input/chest-xray-pneumonia/chest_xray/train/PNEUMONIA" +"/"+i)
    img = cv2.resize(img,(256,256))
    x_train.append(img)
    y_train.append("PNEUMONIA")
```

```
100%|██████████| 3875/3875 [00:58<00:00, 66.60it/s]
```

```
[8]
fig = plt.figure(figsize=(12,6))
fig.set_size_inches(12,12)

for i,image in enumerate(normal):
    plt.subplot(1,5,i+1)
    img = load_img(train_dir+"/NORMAL"+"/"+image)
    plt.imshow(img)
    plt.xlabel("Normal")
    plt.xticks([])
    plt.yticks([])
plt.tight_layout();

plt.figure(figsize=(12,6))
fig.set_size_inches(12,12)

for i,image in enumerate(pneumonia):
    plt.subplot(1,5,i+1)
    img = load_img(train_dir+"/PNEUMONIA"+"/"+image)
    plt.imshow(img)
    plt.xlabel("PNEUMONIA")
    plt.xticks([])
    plt.yticks([])
plt.tight_layout();
```




```
[9]
print("Total images in Train directory: {}".format(len(os.listdir(train_dir+"/NORMAL")) + len(os.listdir(train_dir+"/PNEUMONIA"))))
print("Total images in Test directory: {}".format(len(os.listdir(test_dir+"/NORMAL")) + len(os.listdir(test_dir+"/PNEUMONIA"))))
print("Total images in Validation directory: {}".format(len(os.listdir(valid_dir+"/NORMAL")) + len(os.listdir(valid_dir+"/PNEUMONIA"))))

Total images in Train directory: 5216
Total images in Test directory: 624
Total images in Validation directory: 16
```

2. Data Augmentation using Image Data Generator :

Data Augmentation involves a suite of techniques that boost the size and value of training datasets such that better Deep Learning models can be built using them.

Here we have a small number of images in our training dataset,so image data augmentation can helps us to increase the models performance by increasing the number of images by doing some operation i.e Zooming,shear,flip etc

```
[10]
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=(1./255),shear_range = 0.2,zoom_range=0.3,
                                   horizontal_flip=True)#fill_mode='nearest'
test_datagen = ImageDataGenerator(rescale = (1./255))

train_data = train_datagen.flow_from_directory(directory = train_dir,target_size=(224,224),
                                               class_mode = "categorical",batch_size=32)
test_data = test_datagen.flow_from_directory(directory = test_dir,target_size=(224,224),
                                             class_mode = "categorical",batch_size=32)

Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

3. Import VGG-19 and Model Building :

Now we import our pre-trained VGG-19 model using keras library and we set the 'include_top' to false. This will remove the top layers because we want to customize them.First we freeze the weights of all trainable layers.

Now we add the custom layer for predicting the output for our model. We set the output layer size to 2 since there are only two classes to classify. Then we use 'Adam' optimizer and use 'Categorical Cross Entropy' for compiling the model.

```
[14]
from tensorflow.keras.applications.vgg19 import VGG19
vgg = VGG19(weights = "imagenet",include_top = False,input_shape=(224,224,3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 3s 0us/step

[15]
for layer in vgg.layers:
    layer.trainable = False

[16]
from tensorflow.keras.layers import Flatten,Dense,Dropout,BatchNormalization
x = vgg.output
x = Flatten()(x)
#x = Dense(units=4096, activation='relu')(x)
#x = BatchNormalization()(x)
#x = Dropout(0.5)(x)
predictions = Dense(2,activation= "softmax")(x)

[17]
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

model = Model(inputs = vgg.input,outputs = predictions)
model.compile(optimizer="adam",loss = "categorical_crossentropy",metrics =["accuracy"])#Adam(1e-4)
```

4. Training the Model :

The model is trained with the following parameters :

Hyperparameter	Value
Optimizer	Adam
Loss	Categorical Cross Entropy
Learning Rate	0.01
Epoch	15
Batch Size	32

We also use early stopping and model checkpoint to avoid overfitting.

```
[18]
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

checkpoint = ModelCheckpoint("vgg19.h5",monitor = "val_accuracy",save_best_only=True,
                             save_weights_only=False,mode='auto',verbose=1,period=1)

earlystop = EarlyStopping(monitor="val_acc",patience=5,verbose=1)

[19]
history = model.fit_generator(generator = train_data,validation_data = test_data,
                              epochs = 15,verbose = 1,callbacks=[checkpoint,earlystop])
```

```
Epoch 1/15
163/163 [==============================] - ETA: 0s - loss: 0.3221 - accuracy: 0.8823
Epoch 00001: val_accuracy improved from -inf to 0.85096, saving model to vgg19.h5
163/163 [==============================] - 113s 695ms/step - loss: 0.3221 - accuracy: 0.8823 - val_loss: 0.4750 - val_accuracy: 0.8510
Epoch 2/15
163/163 [==============================] - ETA: 0s - loss: 0.1715 - accuracy: 0.9344
Epoch 00002: val_accuracy improved from 0.85096 to 0.88301, saving model to vgg19.h5
163/163 [==============================] - 112s 688ms/step - loss: 0.1715 - accuracy: 0.9344 - val_loss: 0.2905 - val_accuracy: 0.8830
Epoch 3/15
163/163 [==============================] - ETA: 0s - loss: 0.1301 - accuracy: 0.9500
Epoch 00003: val_accuracy did not improve from 0.88301
163/163 [==============================] - 113s 691ms/step - loss: 0.1301 - accuracy: 0.9500 - val_loss: 0.4904 - val_accuracy: 0.8494
Epoch 4/15
163/163 [==============================] - ETA: 0s - loss: 0.1237 - accuracy: 0.9519
Epoch 00004: val_accuracy improved from 0.88301 to 0.90064, saving model to vgg19.h5
163/163 [==============================] - 110s 672ms/step - loss: 0.1237 - accuracy: 0.9519 - val_loss: 0.2690 - val_accuracy: 0.9006
Epoch 5/15
163/163 [==============================] - ETA: 0s - loss: 0.1141 - accuracy: 0.9584
Epoch 00005: val_accuracy did not improve from 0.90064
163/163 [==============================] - 110s 675ms/step - loss: 0.1141 - accuracy: 0.9584 - val_loss: 0.5008 - val_accuracy: 0.8686
Epoch 6/15
163/163 [==============================] - ETA: 0s - loss: 0.1243 - accuracy: 0.9544
Epoch 00006: val_accuracy improved from 0.90064 to 0.90705, saving model to vgg19.h5
163/163 [==============================] - 110s 674ms/step - loss: 0.1243 - accuracy: 0.9544 - val_loss: 0.3131 - val_accuracy: 0.9071
Epoch 7/15
163/163 [==============================] - ETA: 0s - loss: 0.1400 - accuracy: 0.9498
Epoch 00007: val_accuracy did not improve from 0.90705
163/163 [==============================] - 112s 686ms/step - loss: 0.1400 - accuracy: 0.9498 - val_loss: 0.2893 - val_accuracy: 0.8990
Epoch 8/15
163/163 [==============================] - ETA: 0s - loss: 0.1081 - accuracy: 0.9607
Epoch 00008: val_accuracy did not improve from 0.90705
163/163 [==============================] - 113s 695ms/step - loss: 0.1081 - accuracy: 0.9607 - val_loss: 0.4285 - val_accuracy: 0.8782
Epoch 9/15
163/163 [==============================] - ETA: 0s - loss: 0.1068 - accuracy: 0.9603
Epoch 00009: val_accuracy did not improve from 0.90705
163/163 [==============================] - 112s 686ms/step - loss: 0.1068 - accuracy: 0.9603 - val_loss: 0.5533 - val_accuracy: 0.8606
Epoch 10/15
163/163 [==============================] - ETA: 0s - loss: 0.1219 - accuracy: 0.9567
Epoch 00010: val_accuracy did not improve from 0.90705
163/163 [==============================] - 113s 694ms/step - loss: 0.1219 - accuracy: 0.9567 - val_loss: 0.3196 - val_accuracy: 0.8894
Epoch 11/15
163/163 [==============================] - ETA: 0s - loss: 0.1058 - accuracy: 0.9622
```

```
Epoch 12/15
163/163 [==============================] - ETA: 0s - loss: 0.1162 - accuracy: 0.9569
Epoch 00012: val_accuracy did not improve from 0.90705
163/163 [==============================] - 114s 697ms/step - loss: 0.1162 - accuracy: 0.9569 - val_loss: 0.5889 - val_accuracy: 0.8798
Epoch 13/15
163/163 [==============================] - ETA: 0s - loss: 0.1179 - accuracy: 0.9590
Epoch 00013: val_accuracy did not improve from 0.90705
163/163 [==============================] - 115s 706ms/step - loss: 0.1179 - accuracy: 0.9590 - val_loss: 0.3825 - val_accuracy: 0.9071
Epoch 14/15
163/163 [==============================] - ETA: 0s - loss: 0.1040 - accuracy: 0.9597
Epoch 00014: val_accuracy improved from 0.90705 to 0.91346, saving model to vgg19.h5
163/163 [==============================] - 115s 705ms/step - loss: 0.1040 - accuracy: 0.9597 - val_loss: 0.3475 - val_accuracy: 0.9135
Epoch 15/15
163/163 [==============================] - ETA: 0s - loss: 0.1501 - accuracy: 0.9492
Epoch 00015: val_accuracy did not improve from 0.91346
163/163 [==============================] - 115s 707ms/step - loss: 0.1501 - accuracy: 0.9492 - val_loss: 0.3803 - val_accuracy: 0.8830
```

We can see that our training accuracy has finally reached 94% . Now that the training is complete we are ready to see how it performs on the test data.

5. Visualizing results :

We fetch the testing data and we evaluate our model.

[21]
```
x_test =[]
y_test = []
for i in tqdm(os.listdir("/kaggle/input/chest-xray-pneumonia/chest_xray/test/NORMAL")):
    img = cv2.imread("/kaggle/input/chest-xray-pneumonia/chest_xray/test/NORMAL" +"/"+i)
    img = cv2.resize(img,(224,224))
    x_test.append(img)
    y_test.append("Normal")

for i in tqdm(os.listdir("/kaggle/input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA")):
    img = cv2.imread("/kaggle/input/chest-xray-pneumonia/chest_xray/test/PNEUMONIA" +"/"+i)
    img = cv2.resize(img,(224,224))
    x_test.append(img)
    y_test.append("PNEUMONIA")
```
```
100%|██████████| 234/234 [00:05<00:00, 42.60it/s]
100%|██████████| 390/390 [00:03<00:00, 119.67it/s]
```

[45]
```
x_test = np.array(x_test)
y_test = np.array(y_test)

y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred,axis=1)
```

[46]
```
y_pred[:15]
```
```
array([1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0])
```

[47]
```
unique,counts = np.unique(y_pred,return_counts=True)
print(unique,counts)
```
```
[0 1] [274 350]
```

[48]
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_test = le.fit_transform(y_test)
```

[49]
```
unique,counts = np.unique(y_test,return_counts=True)
print(unique,counts)
```
```
[0 1] [234 390]
```

Classification report on testing results :

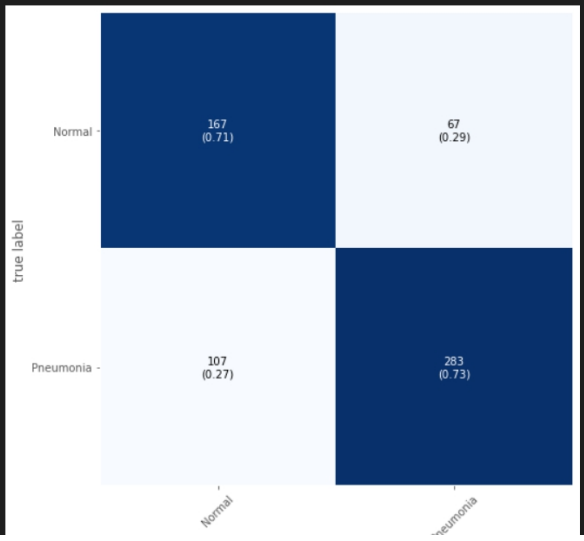
```
from sklearn.metrics import accuracy_score,classification_report
print(classification_report(y_test,y_pred))
```

```
Classification report :
              precision    recall  f1-score   support

           0       0.91      0.86      0.89        73
           1       0.88      0.92      0.90        77

    accuracy                           0.89       150
   macro avg       0.89      0.89      0.89       150
weighted avg       0.89      0.89      0.89       150
```

From the above report we can conclude that our model has a **testing accuracy of 89%** which is really good.

Confusion matrix :

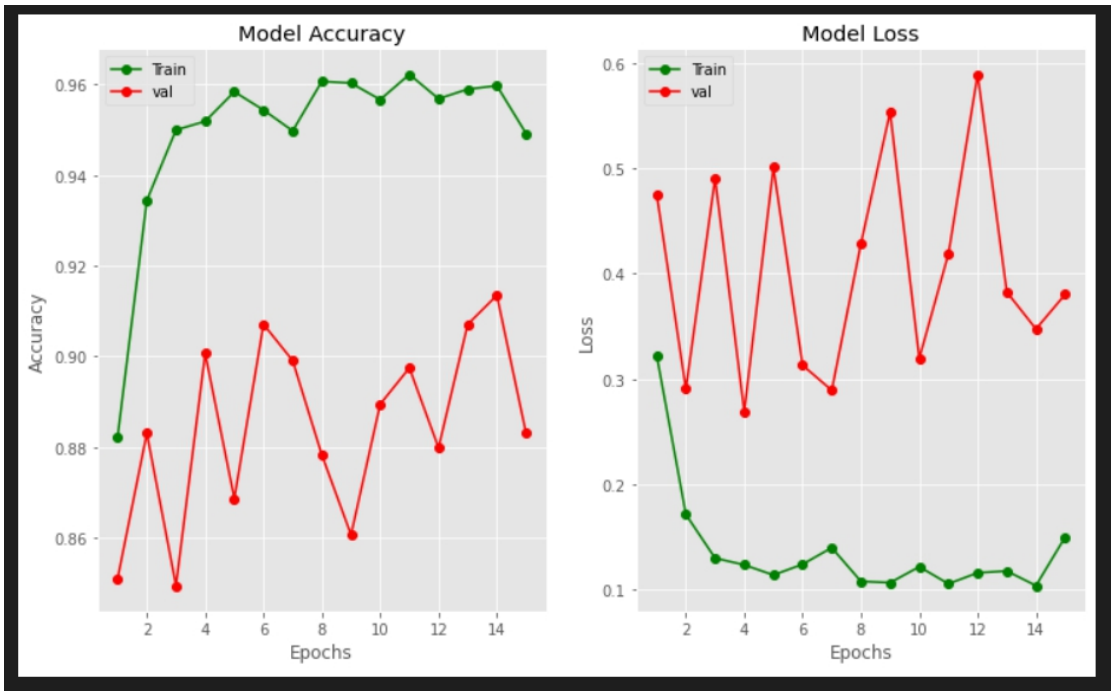
```
[51]
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(conf_mat = cm,figsize=(8,8),class_names = ["Normal","Pneumonia"],show_normed=True)

(<Figure size 576x576 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7ff5e0809850>)
```



Plotting the results :

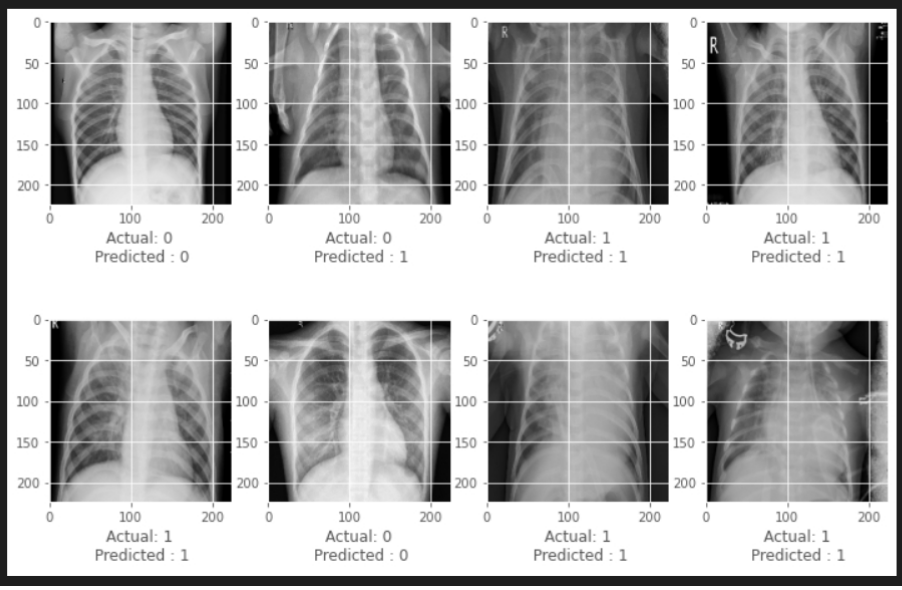
[52]

```
plt.style.use("ggplot")
fig = plt.figure(figsize=(12,7))
epochs = range(1,16)
plt.subplot(1,2,1)
plt.plot(epochs,history.history["accuracy"],"go-")
plt.plot(epochs,history.history["val_accuracy"],"ro-")
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Train","val"],loc = "upper left")
#plt.show()

#fig = plt.figure(figsize=(12,8))
plt.subplot(1,2,2)
plt.plot(epochs,history.history["loss"],"go-")
plt.plot(epochs,history.history["val_loss"],"ro-")
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Train","val"],loc = "upper left")
plt.show()
```



Predicted vs Actual Results :

```
[53]
plt.figure(figsize=(12,8))
#plt.title("0 for Normal & 1 is for Pneumonia")
for i in range(8):
    sample_index = np.random.randint(1,len(y_test))
    plt.subplot(2,4,i+1)
    plt.imshow(x_test[sample_index])
    plt.xlabel("Actual: {}\n Predicted : {}".format(y_test[sample_index],y_pred[sample_index]))
```



6. CONCLUSION :

Thus we used the VGG-19 model to classify the CXR images into Normal and Pneumonia affected using transfer learning. We also got good training and testing accuracies while doing so.