MOUNTING GOOGLE DRIVE

```
import pandas as pd,numpy as np, matplotlib.pyplot as plt,seaborn as sns
import tensorflow as tf
import os
import cv2
from google.colab import drive
drive.mount('/content/drive')
```

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

SETTING FILE LOCATIONS

```
from PIL import Image
train_fold_normal=('/content/drive/MyDrive/project/train_data/NORMAL')
train_fold_pneumonia=('/content/drive/MyDrive/project/train_data/PNEUMONIA')
test_fold_normal=('/content/drive/MyDrive/project/test_data/NORMAL')
test_fold_pneumonia=('/content/drive/MyDrive/project/test_data/PNEUMONIA')
val_fold_normal=('/content/drive/MyDrive/project/val_data/NORMAL')
val_fold_pneumonia=('/content/drive/MyDrive/project/val_data/PNEUMONIA')
```

LOADING DATASETS

```
x_images=[]
y_labels=[]
for file in os.listdir(train_fold_normal):
  img=Image.open(os.path.join(train_fold_normal,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_images.append(img)
  y_labels.append(0)


for file in os.listdir(train_fold_pneumonia):
  img=Image.open(os.path.join(train_fold_pneumonia,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_images.append(img)
  y_labels.append(1)


for file in os.listdir(test_fold_normal):
  img=Image.open(os.path.join(test_fold_normal,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_images.append(img)
  y_labels.append(0)


for file in os.listdir(test_fold_pneumonia):
  img=Image.open(os.path.join(test_fold_pneumonia,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_images.append(img)
  y_labels.append(1)


x_valimages=[]
y_vallabels=[]
for file in os.listdir(val_fold_normal):
  img=Image.open(os.path.join(val_fold_normal,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_valimages.append(img)
  y_vallabels.append(0)
for file in os.listdir(val_fold_pneumonia):
  img=Image.open(os.path.join(val_fold_pneumonia,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  x_valimages.append(img)
  y_vallabels.append(1)
```

CONVERTING THEM INTO ARRAY AND PRINTING ITS SHAPE

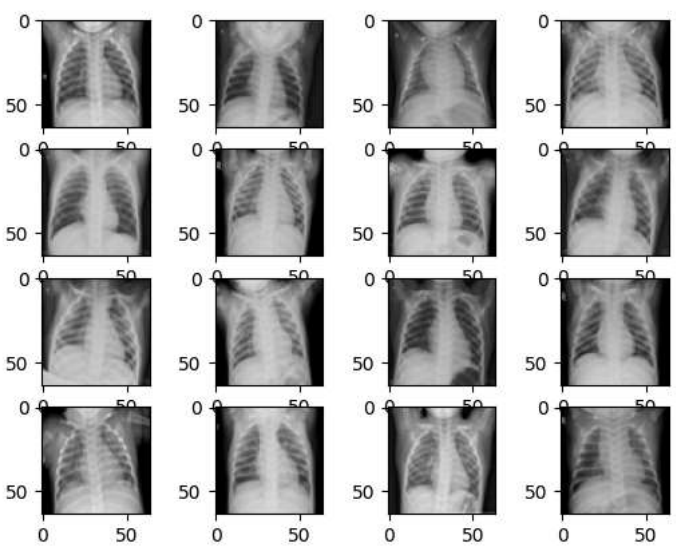
```
x_images=np.array(x_images)
y_labels=np.array(y_labels)
x_valimages=np.array(x_valimages)
y_vallabels=np.array(y_vallabels)
```

```
print("xray images1",x_images.shape)
print("y labels for the xrays",y_labels.shape)
print('validation images',x_valimages.shape)
print('validation labels',y_vallabels.shape)
```

```
xray images1 (5897, 64, 64, 3)
y labels for the xrays (5897,)
validation images (16, 64, 64, 3)
validation labels (16,)
```

VERIFYING THE XRAY IMAGES ARE RIGHT

```
for i in range(16):
  plt.subplot(4,4,i+1)
  plt.imshow(x_images[i])
```



```
y_labels[1:2500]
```

```
array([0, 0, 0, ..., 1, 1, 1])
```

SPLITING XRAYS AND ITS LABELS INTO TRAIN AND TEST DATA SETS

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_images,y_labels,test_size=0.2,random_state=42)
print('x train',x_train.shape)
print('y train',y_train.shape)
print('x test',x_test.shape)
print('y test',y_test.shape)
```

```
x train (4717, 64, 64, 3)
y train (4717,)
x test (1180, 64, 64, 3)
y test (1180,)
```

FLATTENING THE IMAGES

```
trainimagesflat=x_train.reshape(len(x_train),-1)
testimagesflat=x_test.reshape(len(x_test),-1)
valimagesflat=x_valimages.reshape(len(y_valimages),-1)
print("flattend train data shape",trainimagesflat.shape)
print("flattend test data shape",testimagesflat.shape)
```

```
print("flattend val data shape",valimagesflat.shape)
```

```
flattend train data shape (4717, 12288)
flattend test data shape (1180, 12288)
flattend val data shape (16, 12288)
```

RESHAPING INTO CONVINIENT FILES

```
trainimage=x_train.reshape(len(x_train),-1).astype('float32')/255
testimage=x_test.reshape(len(x_test),-1).astype('float32')/255
valimages=x_valimages.reshape(len(x_valimages),-1).astype('float32')/255
trainimage=trainimage.reshape(-1,64,64,3)
testimage=testimage.reshape(-1,64,64,3)
valimages=valimages.reshape(-1,64,64,3)
print("reshaped train data shape",trainimage.shape)
print("reshaped test data shape",testimage.shape)
print("reshaped val data shape",valimages.shape)
```

```
reshaped train data shape (4717, 64, 64, 3)
reshaped test data shape (1180, 64, 64, 3)
reshaped val data shape (16, 64, 64, 3)
```

```
trainimage.shape
```

```
(4717, 64, 64, 3)
```

CLASSYFYING THE LABELS INTO A BINARY DATASET

```
from tensorflow.keras.utils import to_categorical
trainlabels=to_categorical(y_train,num_classes=2)
testlabels=to_categorical(y_test,num_classes=2)
vallabels=to_categorical(y_vallabels,num_classes=2)
trainlabels.shape,vallabels.shape
```

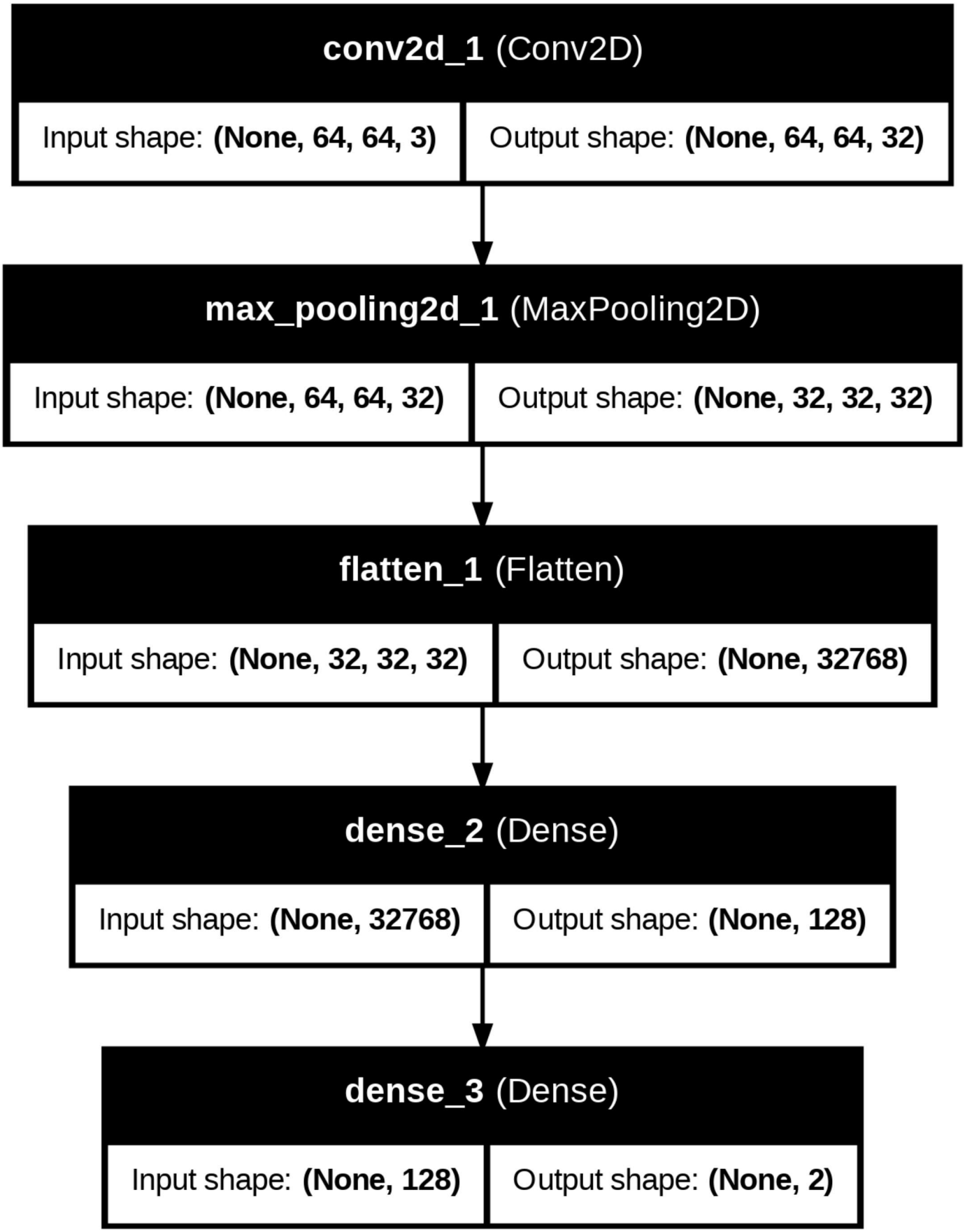
```
((4717, 2), (16, 2))
```

CONVOLUTIONAL NEURAL NETWORK OPERATIONS(CNN)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
model=Sequential()
model.add(layers.Conv2D(32,(3,3),strides=(1,1),padding='same',activation='relu',input_shape=(64,64,3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Flatten(input_shape=(64,64,3)))
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(2,activation='sigmoid'))
```

PLOTTING THE MODELS INPUT AND OUTPUTS

```
from tensorflow.keras.utils import plot_model
plot_model(model,show_shapes=True,show_layer_names=True)
```


conv2d_1 (Conv2D)
Input shape: (None, 64, 64, 3)
Output shape: (None, 64, 64, 32)
max_pooling2d_1 (MaxPooling2D)
Input shape: (None, 64, 64, 32)
Output shape: (None, 32, 32, 32)
flatten_1 (Flatten)
Input shape: (None, 32, 32, 32)
Output shape: (None, 32768)
dense_2 (Dense)
Input shape: (None, 32768)
Output shape: (None, 128)
dense_3 (Dense)
Input shape: (None, 128)
Output shape: (None, 2)

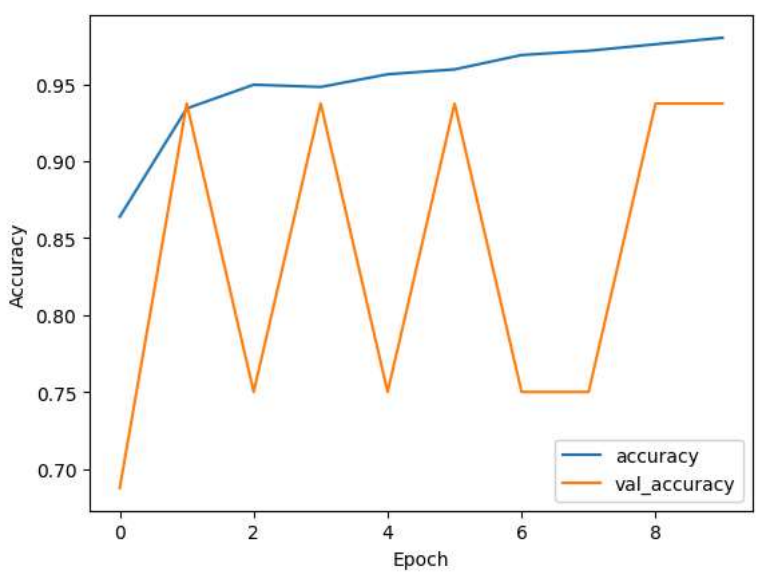
TESTING THE LOSS AND ACCURACY OF THE MODEL

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history=model.fit(trainimage,trainlabels,epochs=10,batch_size=32,validation_data=(valimages,vallabels))
test_loss,test_accuracy=model.evaluate(testimage,testlabels)
print('test loss',test_loss)
print('test accuracy',test_accuracy)
```

```
Epoch 1/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 23s 142ms/step - accuracy: 0.7931 - loss: 0.6301 - val_accuracy: 0.6875 - val_loss: 0.7185
Epoch 2/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 41s 147ms/step - accuracy: 0.9341 - loss: 0.1766 - val_accuracy: 0.9375 - val_loss: 0.3013
Epoch 3/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 40s 143ms/step - accuracy: 0.9546 - loss: 0.1314 - val_accuracy: 0.7500 - val_loss: 0.5284
Epoch 4/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 39s 131ms/step - accuracy: 0.9514 - loss: 0.1179 - val_accuracy: 0.9375 - val_loss: 0.2643
Epoch 5/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 21s 143ms/step - accuracy: 0.9548 - loss: 0.1119 - val_accuracy: 0.7500 - val_loss: 0.3990
Epoch 6/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 40s 139ms/step - accuracy: 0.9594 - loss: 0.1036 - val_accuracy: 0.9375 - val_loss: 0.1664
Epoch 7/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 42s 145ms/step - accuracy: 0.9733 - loss: 0.0819 - val_accuracy: 0.7500 - val_loss: 0.3795
Epoch 8/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 38s 128ms/step - accuracy: 0.9736 - loss: 0.0727 - val_accuracy: 0.7500 - val_loss: 0.5194
Epoch 9/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 22s 141ms/step - accuracy: 0.9762 - loss: 0.0639 - val_accuracy: 0.9375 - val_loss: 0.2551
Epoch 10/10
148/148 ━━━━━━━━━━━━━━━━━━━━ 39s 130ms/step - accuracy: 0.9834 - loss: 0.0531 - val_accuracy: 0.9375 - val_loss: 0.1586
37/37 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9283 - loss: 0.1899
test loss 0.18404045701026917
test accuracy 0.9347457885742188
```

PLOTTING GRAPH ON EPOCH AND ACCURACY

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
testimage.shape
```

```
(1180, 64, 64, 3)
```

TEST ACCURACY AND TEST LOSS

```
test_loss,test_accuracy=model.evaluate(testimage,testlabels)
print('test loss',test_loss)
print('test accuracy',test_accuracy)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9283 - loss: 0.1899
test loss 0.18404045701026917
test accuracy 0.9347457885742188
```

MODEL EVALUATION

```
val_images1=('/content/drive/MyDrive/project/val_data/NORMAL')
val_images2=('/content/drive/MyDrive/project/val_data/PNEUMONIA')
val_image=[]
val_label=[]
for file in os.listdir(val_images1):
  img=Image.open(os.path.join(val_images1,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  val_image.append(img)
  val_label.append(0)
for file in os.listdir(val_images2):
  img=Image.open(os.path.join(val_images2,file)).convert('RGB')
  img=img.resize((64,64))
  img=np.array(img)
  val_image.append(img)
  val_label.append(1)
val_image=np.array(val_image)
val_label=np.array(val_label)
val_image=val_image.reshape(len(val_image),64,64,3).astype('float32')/255
val_label=to_categorical(val_label,num_classes=2)
valloss,valaccuracy=model.evaluate(val_image,val_label)
print('validation loss',valloss)
print('validation accuracy',valaccuracy)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 0.9375 - loss: 0.1586
validation loss 0.15860748291015625
validation accuracy 0.9375
```

```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

y_pred = model.predict(x_test)
y_pred_binary = np.argmax(y_pred, axis=1)
if y_test.ndim == 2:
    y_test = np.argmax(y_test, axis=1)

print("Classification Report")
print(classification_report(y_test, y_pred_binary))
cm = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(cm)
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step
Classification Report
              precision    recall  f1-score   support

           0       0.64      0.98      0.78       353
           1       0.99      0.77      0.87       827

    accuracy                           0.83      1180
   macro avg       0.82      0.87      0.82      1180
weighted avg       0.89      0.83      0.84      1180

Confusion Matrix:
[[345   8]
 [190 637]]
```

LABELLING

```
prediction=model.predict(val_image)
pred_class=np.argmax(prediction,axis=1)
```

```
class_labels=['NORMAL','PNEUMONIA']
predicted_label=[class_labels[i] for i in pred_class]
val_label=np.argmax(val_label,axis=1)
true_label=[class_labels[i] for i in val_label]
print('predicted classses index',predicted_label)
print('true labels',true_label)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
predicted classses index ['NORMAL', 'PNEUMONIA', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', '
true labels ['NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNE
```

```
img_to_pred=np.expand_dims(val_image[2],axis=0)
prediction=model.predict(img_to_pred)
pred_class=np.argmax(prediction)
plt.imshow(val_image[2].reshape(64,64,3),cmap='gray')
plt.title("True label: " + str(val_label[0]))
plt.title("Predicted label: " + str(pred_class))
plt.axis('off')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```