```
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
```

```
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.layers import Dropout
from keras.layers import BatchNormalization
import keras
from keras import backend as K
import matplotlib.pyplot as plt
import sklearn


path_normal = '/content/drive/MyDrive/Deep learning demo project/Normal/'
path_pneumonia = '/content/drive/MyDrive/Deep learning demo project/pneumonia/'


##Import necessary libraries
import numpy as np
import PIL
import cv2
import os
data1 = list()
data2 = list()
x = list()


##Class-1 images##

for image in os.walk(path_normal):
  data1.append(image[2])

for i in range(len(data1[0])):
  str_complete = path_normal + data1[0][i]
  img = cv2.imread(str_complete)
  img = cv2.resize(img, (224, 224))
  x.append(img)
  print(i)#Ensure all images are loaded
```

```
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
```

```
print(img.shape)
```

```
(224, 224, 3)
```

```
##Class-2 images##

for image in os.walk(path_pneumonia):
  data2.append(image[2])

for i in range(len(data2[0])):
  str_complete = path_pneumonia + data2[0][i]
  img = cv2.imread(str_complete)
  img = cv2.resize(img, (224, 224))
  x.append(img)#Ensure all images are loaded
  print(i)
```

```
687
688
689
690
691
692
693
694
695
696
697
698
699
```

```
data_x = np.asarray(x)
```

```
data_x.shape
```

```
(1400, 224, 224, 3)
```

```
x=data_x
```

```
y = np.zeros(1400)
y[:700] =1
y[700:1400]=2
```

```
from sklearn.model_selection import train_test_split
```

```
##Dataset Split##
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
#y = to_categorical(y)
x_train, x_test, y_train, y_test = train_test_split(data_x, y, test_size=0.2, random_state=1)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=1/8, random_state=1)

y_tr_one_hot  = np.zeros((np.array(y_train).shape[0],2))

for i in range(np.array(y_train).shape[0]):
  label = y_train[i]-1
  y_tr_one_hot[i][int(label)] = 1


y_val_one_hot  = np.zeros((np.array(y_val).shape[0],2))

for i in range(np.array(y_val).shape[0]):
  label = y_val[i]-1
  y_val_one_hot[i][int(label)] = 1


y_te_one_hot  = np.zeros((np.array(y_test).shape[0],2))

for i in range(np.array(y_test).shape[0]):
  label = y_test[i]-1
  y_te_one_hot[i][int(label)] = 1


from keras.models import load_model
from keras.layers import Lambda
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense


model = tf.keras.applications.DenseNet201(include_top=False,input_shape=(224,224,3))
# mark loaded layers as not trainable
for layer in model.layers:
  layer.trainable = False
# add new classifier layers
flat1 = Flatten()(model.layers[-1].output)
#x=Dense(1024,activation='relu')(flat1) # FC layer 1
#x=Dense(64,activation='relu')(x)   # FC layer 2
output = Dense(2, activation='softmax')(flat1)
model = Model(inputs=model.inputs, outputs=output)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_tr_one_hot, validation_data=(x_val, y_val_one_hot), epochs=5, batch_size=200,verbose=1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_or
74836368/74836368 [==============================] - 1s 0us/step
```

```
Epoch 1/5
5/5 [==============================] - 438s 86s/step - loss: 13.3887 - accuracy: 0.6684 - val_loss: 6.8950 - val_accuracy
Epoch 2/5
5/5 [==============================] - 400s 84s/step - loss: 5.7344 - accuracy: 0.8276 - val_loss: 2.2513 - val_accuracy:
Epoch 3/5
5/5 [==============================] - 358s 73s/step - loss: 1.2322 - accuracy: 0.9582 - val_loss: 1.7628 - val_accuracy:
Epoch 4/5
5/5 [==============================] - 392s 82s/step - loss: 1.4610 - accuracy: 0.9633 - val_loss: 1.8438 - val_accuracy:
Epoch 5/5
5/5 [==============================] - 395s 83s/step - loss: 1.0438 - accuracy: 0.9684 - val_loss: 1.1649 - val_accuracy:
<keras.callbacks.History at 0x7f0713b83250>
```

```
import sklearn
from sklearn.metrics import confusion_matrix
```

```
test_loss, test_acc = model.evaluate(np.array(x_test), np.array(y_te_one_hot), verbose=0)
print(test_acc)
##Evaluating Sensitivity, Accuracy and Kappa scores
y_prob = model.predict(x_test)
Y_pred = y_prob.argmax(axis=-1)
```

```
0.9928571581840515
9/9 [==============================] - 85s 9s/step
```

```
cm1 = confusion_matrix(y_test-1,Y_pred)
print("confusion matrix \n",cm1)
```

```
confusion matrix
 [[136   1]
 [  1 142]]
```

```
from sklearn.metrics import classification_report
```

```
import pandas as pd
```

```
print(pd.DataFrame(classification_report(y_test-1,Y_pred,output_dict=True)).T)
Kappa=sklearn.metrics.cohen_kappa_score(y_test-1,Y_pred)
print('Kappa=',Kappa)
```

```
                precision    recall  f1-score     support
0.0              0.992701  0.992701  0.992701  137.000000
1.0              0.993007  0.993007  0.993007  143.000000
accuracy         0.992857  0.992857  0.992857    0.992857
macro avg        0.992854  0.992854  0.992854  280.000000
weighted avg     0.992857  0.992857  0.992857  280.000000
Kappa= 0.9857077229340003
```

✓ 0s completed at 15:54