Team- Atleast_we_tried

Members- Kandregula Abhinav, Adwait Dhamane

Code, begins.....

Importing necessary libraries

```
import cv2
import pandas as pd
import numpy as np
from csv import *
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import os

from keras.layers import concatenate
from keras.layers import Input
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import MaxPooling2D, Flatten,Conv2D,
Dense,BatchNormalization,GlobalAveragePooling2D,Dropout
from keras.applications.densenet import DenseNet121
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.applications.vgg16 import VGG16
from keras.applications.resnet_v2 import ResNet50V2
from keras.applications.inception_v3 import InceptionV3
from keras.models import load_model
from sklearn.model_selection import StratifiedKFold
#from keras import vis.utilis
```

Defining paths

```
train_csv = r"path to train_images"
train_img = r"path to train_folder"
test_img = r"path to test_images"
lis = (os.listdir(train_img))
lis2 = os.listdir(test_img)
```

Defining 2 functions for image preprocessing

```
def image_equalization(image):

    R, G, B = cv2.split(image)
    output1_R = cv2.equalizeHist(R)
    output1_G = cv2.equalizeHist(G)
    output1_B = cv2.equalizeHist(B)
    equ = cv2.merge((output1_R, output1_G, output1_B))

    return equ
```

```
def preprocess_image(img_arr):

    img_arr = img_arr.astype('float16')
    img_arr /= 255
    if len(img_arr.shape) > 2:
        img_arr = img_arr[:, :, 0]
    img_arr = np.stack((img_arr, img_arr, img_arr), axis=-1)
    img_arr = cv2.resize(img_arr, (224, 224))
    return img_arr
```

Data Loading

```
y  = []
y1 = []
df = pd.read_csv(train_csv)
for i in range(len(lis)):
    if df.iloc[i][0] in lis:
        if df.iloc[i][1] == "No Finding":
            y.append(0)
        else:
            y.append(1)

        img_arr = cv2.imread(train_img+chr(92)+df.iloc[i][0])

        resized_img = cv2.resize(img_arr, (224, 224),interpolation =
cv2.INTER_CUBIC)
        eq_img = image_equalization(resized_img)

        y1.append(eq_img.astype(np.float16))

X_train, X_val, y_train, y_val = train_test_split(y1, y,
test_size=0.2, random_state=42)

X_train = np.array(X_train)
X_val = np.array(X_val)

y_train = np.array(y_train)
y_val = np.array(y_val)

X_train/=255
X_val/=255

X_test = []
for item in lis2:
    img = cv2.imread(test_img+chr(92)+item)
    resized_img = cv2.resize(img, (224, 224), interpolation =
cv2.INTER_CUBIC)
    eq_img = image_equalization(resized_img)
    X_test.append(eq_img.astype(np.float16))
```

```
X_test = np.array(X_test)
X_test/=255
```

Building the model

```
model_1 = load_model("path to cheXnet weights")
#model_1.summary()

input_layer = Input(shape = (224, 224, 3))

opt1 = model_1.layers[-2].output

for layer in model_1.layers:
    layer.trainable =  False

x = BatchNormalization()(opt1)
x = Dense(256,activation = "relu")(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128,activation = "relu")(x)
x = Dropout(0.5)(x)
x = Dense(1, activation = "sigmoid")(x)
stacked_model = keras.Model(inputs = model_1.input, outputs = x)
stacked_model.compile(optimizer = 'adam', loss =
'binary_crossentropy', metrics = ['accuracy'])
```

Defining folds for 5 folds cross validation

```
folds =
list(StratifiedKFold(n_splits=5,shuffle=True,random_state=1).split(X_t
rain,y_train))
```

Running the 5 fold cross validation over X_train, y_train

```
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
for j, (train_idx,val_idx) in enumerate(folds):
    print('\nFold',j)
    X_train_cv = X_train[train_idx]
    y_train_cv = y_train[train_idx]
    X_val_cv = X_train[val_idx]
    y_val_cv = y_train[val_idx]

    early_stop = EarlyStopping(monitor='val_accuracy', patience=3,
verbose=1)
    mcp_save = ModelCheckpoint('savelocation for
weights',save_best_only=True,monitor='val_accuracy',mode='max')
    stacked_model.fit(X_train_cv, y_train_cv,epochs=10,
validation_data=(X_val_cv, y_val_cv), callbacks=[early_stop,
mcp_save])
```

```
model_1_wts = load_model("path to weights")
opt = model_1_wts.predict(X_test)

opt_val = model_1_wts.predict(X_val)
```

Snippet to get the best treshold value that minimizes the loss

```
import numpy as np
accu = []

pred_probs = opt_val
ground_truth = y_val

thresholds = np.arange(0, 1, 0.01)

best_threshold = 0
best_accuracy = 0

for threshold in thresholds:

    pred_labels = (pred_probs >= threshold).astype(int)

    accuracy = np.mean(pred_labels == ground_truth)
    accu.append((threshold,accuracy))

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_threshold = threshold

print(f"Best threshold: {best_threshold:.2f}")
print(f"Best accuracy: {best_accuracy:.2f}")
```

Generation of submission dataframe

```
lis3 = []
for i in range(opt.shape[0]):
    if opt[i] < best_threshold:
        lis3.append(0)
    else:
        lis3.append(1)
df2 = pd.DataFrame()
df2["Image Index"] = lis2
df2["Finding Labels"] = lis3
df2.to_csv("final_opt.csv", index=False)
```