ECG paper record digitization and diagnosis using deep learning

Siddharth Mishra, Gaurav Khatwani, Rupali Patil, Darshan Sapariya, Vruddhi Shah, Darsh Parmar, Sharath Dinesh, Prathamesh Daphal, Ninad Mehendale

S1 : Code for PDF to JPG conversion

Code

Comments

```
from pdf2image import convert_from_path
import os
import sys
import shutil
```

Importing required libraries

```
outputdir = "cd4_pdf3/"
count = 1
def convert(file, outputdir):
    global count
```

Specifies the directory where the output will be saved

```
    if not os.path.exists(outputdir):
        os.makedirs(outputdir)
```

Checking if the output directory exists in the location

```
    pages = convert_from_path(file)
```

Converting all the pages of pdfs into list of images

```
    for page in pages:
        myfile = outputdir + 'image' + str(count) + '.jpg'
        count = count + 1
```

Iterating through all pages of provided pdf

```
        page.save(myfile, "JPEG")
        print(myfile)
    print(file)
```

Storing the images in the output directory

```
args = sys.argv
if len(args) > 1:
    file = args[1]

pdfs = os.listdir(file)
j = 0
for i in range(len(pdfs)):
```

```
    if pdfs[i].endswith('.pdf'):
        j = j + 1
        convert(file + pdfs[i], outputdir)
    else:
        if not os.path.exists(outputdir):
            os.makedirs(outputdir)
```

Checking if the file is in pdf format, if yes, it will convert it into images

```
            shutil.copy(file + pdfs[i], outputdir)
```

If the file is not in pdf format, it will just copy the file into output directory

S2 : Extraction of 12-Lead ECGs from Image

Code

Comments

```
import cv2
import os
```

Importing required Libraries

```
def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x,y)
```

Left mouse click will give us the coordinate of the point, helps in hardcoding

```
image = cv2.imread('image4.jpg')
scale_percent = 20
```

Reading and resizing the image

```
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
```

calculate the 50 percent of original dimensions

```
dsize = (width, height)
print(dsize)

image = cv2.resize(image, (1097,774))
```

resize image

```
cv2.imshow('Image', image)
cv2.setMouseCallback('Image', click_event)
cv2.waitKey(0)
graph = []

for i in range(3):
    for j in range(4):
        graph.append(image[336+118*i:437+118*i, 42+254*j:296+254*j])
print(len(graph))
outputdir = "graphs4/"
```

Slicing the coordinates of the rectangles of the 12 leads into a list

```
for i in range(12):
    cv2.imshow('graph'+str(i+1), graph[i])
    if not os.path.exists(outputdir):
        os.makedirs(outputdir)
        cv2.imwrite(outputdir+"graph"+str(i+1)+".jpg", graph[i])
    else:
        cv2.imwrite(outputdir+"graph"+str(i+1)+".jpg", graph[i])
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Saving the 12 seperate leads into the output directory

S3:Extraction of 12-Lead ECG signal from continuous ECG

Code

Comments

```
import cv2
import os
for k in range(1,2):
    cv2.namedWindow('Image',cv2.WINDOW_NORMAL
)
    cv2.resizeWindow('Image', 1000, 800)
```

Importing the libraries

```
    array_x = []
    array_y = []
    array_x.append(0)
    array_y.append(0)
    def click_event(event, x, y, flags, param):
```

arrays to store

```
        if event == cv2.EVENT_LBUTTONDOWN:


            print("x = ", x)
            array_x.append(x)
            cv2.line(image,(x,0), (x,height), (0,0,0),
thickness= 4 )
            cv2.imshow('Image', image)
```

If a left mouse click event occurs, a vertical black line will be drawn on the image

```
        if event == cv2.EVENT_RBUTTONDOWN:


            print("y = ", y)
            array_y.append(y)
```

If a right mouse click event occurs, a horizontal black line will be drawn on the image

```
            cv2.line(image,(0, y), (width, y), (0,0,0), thickness= 4 )
            cv2.imshow('Image', image)

    image = cv2.imread('D:/pdf2image/pdf2/stemi/photo226.jpg', 1)
```

reading and displaying the image

```
    height = image.shape[0]
    width = image.shape[1]

    cv2.imshow("Image", image)
    cv2.setMouseCallback('Image', click_event)

    cv2.waitKey(0)



    array_x.append(width)
    array_y.append(height)
    array_x.sort()
    array_y.sort()
    print(array_x)
    print(array_y)

    cv2.namedWindow('temp',cv2.WINDOW_NORMAL)
    cv2.resizeWindow('temp', 400, 200)

    outputdir = "lead_images/"
```

creating an output directory where the images will be stored

```
    count = 1

    for i in range(1, len(array_y)-2):
```

Saving the rectangles formed in the image, into seperate jpg files, hence converting single lead image into seperate 12 lead images

```
        for j in range(1, len(array_x) - 2):
            print('start = ' + str(array_x[j])+ ' ' + str(array_y[i]))
            print('end = ' + str(array_x[j+1])+ ' ' + str(array_y[i+1]))
            temp = image[array_y[i]:array_y[i+1], array_x[j]:array_x[j+1]]
```

```
                        if no
os.path.exists(outputdir+'image'+str(k)+'/'):

        os.makedirs(outputdir+'image'+str(k)+'/')
                        cv2.imwrite(outputdir + 'photo226.jpg',
temp)
                        count = count+1

        cv2.destroyAllWindows()
```

S4 : Finding Threshold value of Image using Deep Learning

Code	Comments

import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense , Dropout
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.pipeline import Pipeline
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler
	Importing Libraries

values = pd.read_excel('ImageData.xlsx') y = pd.read_excel('Threshold_values.xlsx')	Importing required files
y = y.round(0)	Rounding off values
X = values.copy()	Creating a copy of given data
for i in range(1,254,1):	Iterating through complete file

j=i+1
    X[i] =values[i] -values[j]
	Creating Delta data

X.drop(columns = [255] , axis = 1 , inplace = True)	Dropping the last column
X = X.apply(lambda x: 1/x)	Taking the inverse of every Element in X

for i in range(1,254,1): j=i+1	Iterating through whole file
X[i] =values[i] -values[j]	Creating Delta Data
X = X.apply(lambda x: x*1000)	Multiplying each element of Delta data by 1000
X.drop(columns = [254], axis=1, inplace = True)	Dropping the last column of delta Data
X = MinMaxScaler().fit_transform(X)	Scaling out the data
X = X.round(4)	Rounding off the data up to 4 decimals
y = y.to_numpy()	Converting the output column to array
X_train , X_test ,y_train , y_test = train_test_split(X,y , test_size = 0.05, random_state = 40)	Splitting the data into train and Test

model = Sequential()
n_cols = X.shape[1]
	Model Definition

model.add(Dense(253, activation= 'relu', input_shape = (n_cols,)))	Input Layers

model.add(Dense(253, activation = 'relu'))
model.add(Dense(253, activation = 'relu'))
model.add(Dropout(0.5))
	Hidden layers Formations

model.add(Dense(1, activation = 'relu'))	Output layer
model.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics= ['accuracy'])	Compiling the model

early_stopping_monitor = EarlyStopping(patience=15)
checkpointer = ModelCheckpoint('best_model.hdf5',monitor = 'val_loss',verbose = 2, save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1 , patience = 5, min_lr = 0.01, verbose = 2 , mod = 'min')
	Early stopping configuration

model.fit(X_train, y_train, validation_split=0.02, epochs= 3000,verbose = 1, callbacks=[early_stopping_monitor, checkpointer , reduce_lr])	Train the model
model.load_weights('best_model_all_data_with_2.89RMSE.hdf5')	loading the best model for prediction
pred = model.predict(X)	predicting the answer
pred = pred.round(0)	Conversion of Float to int
to integer	

```
score = np.sqrt(metrics.mean_squared_error(y,pred ))
```

applying RMSE to Predicted Data

```
print ("Score (RMSE) : {}".format(score))
```

printing the RMSE score

```
pred.to_excel('Predicted_values.xslx')
```

exporting the output as excel sheet

```
summary = model.summary()
```

getting the summary of model

S5: function for Scaling the image and preparing data

Code

Comments

```
import cv2
import numpy as np
from scipy import stats

from collections import Counter
from skimage.morphology import skeletonize
import matplotlib.pyplot as plt
def nothing(x):
    pass
cv2.namedWindow('s',cv2.WINDOW_NORMAL)
cv2.createTrackbar('R','s',0,255,nothing)
kernel = np.ones((3,3),np.uint8)
```

importing the libraries

```
class image:
    def img(imgg,thresh):
```

function for preprocessing the image

```
        for ii in range(1,2):
            img1=cv2.cvtColor(imgg,cv2.COLOR_BGR2GRAY)
            img2=imgg


            while True:

                r = cv2.getTrackbarPos('R','s')

_,th=cv2.threshold(img1,r,255,cv2.THRESH_BINARY_INV)
                thresh=th.copy()
                cv2.imshow('s',th)
```

thresholding

```
    cv2.imshow('ss',img2)
    if cv2.waitKey(1) & 0xFF==ord('q'):
        tt = cv2.dilate(th,kernel,iterations = 1)
        print(tt.size)
        break
```

dilation

```
for i in range(len(tt)):
```

Converting the image array to True-False array

```
    for j in range(len(tt[i])):
        if tt[i][j] == 0:
            tt[i][j] = False
        else:
            tt[i][j] = True
skeleton =skeletonize(tt)
```

skeletonizing the image

```
for i in range(len(skeleton)):
```

converting it back to binary array with values 0 and 255 only

```
    for j in range(len(skeleton[i])):
        if skeleton[i][j] == False:
            th[i][j] = 0
        else:
            th[i][j] = 255

return thresh,th
```

returning the skeletonized image

S6 : Function to Remove the Letters from Images

Code

Comments

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
```

importing the libraries

```
class letter:
    def remove(img):
```

taking the skeletonized image as the input

```
        scale=img

_,scal=cv2.threshold(scale,90,255,cv2.THRESH_BINARY)
```

```
        col=[]
        row=[]
        print(scal.shape[0], scal.shape[1])
        rows = scal.shape[0]
        cols = scal.shape[1]

        flag = 0
        for i in range(0,cols):
```

Removing impurities present above the lead graph, by vertically scanning every column of the image from bottom to top

```
            flag = 0
            for j in range(rows - 10, 0, -1):
                if scal[j][i]==255 and flag == 0:
                    flag = 1
                elif scal[j][i]==255 and flag == 1:
                    continue
                elif scal[j][i]==0 and flag == 1:
                    flag = 2

                if scal[j][i] == 255 and flag == 2:
                    if 255 not in [scal[k][i] for k in range(j+1, j+5)]:
                        scal=cv2.rectangle(scal,(i-10,j-
20),(i+10,j+10),(0,0,0),-1)
                        scal[j][i]=0

        for i in range(0,cols):
```

Removing impurities present below the lead graph, by vertically scanning every column of the image from top to bottom

```
            flag = 0
            for j in range(0,rows - 10):
                if scal[j][i]==255 and flag == 0:
                    flag = 1
                elif scal[j][i]==255 and flag == 1:
                    continue
                elif scal[j][i]==0 and flag == 1:
                    flag = 2

                if scal[j][i] == 255 and flag == 2:
                    if 255 not in [scal[k][i] for k in range(j,j-10,-1)]:

scal=cv2.rectangle(scal,(j,i),(j+10,i+10),(0,0,0),-1)
                        scal[j:][i] = 0
            break
        return scal
```

S7 : Shadow removal from Image and calls S7 and S6 for further processing

Code

Comments

Importing the libraries and packages

```
import cv2

import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
from scale import image
from Letter_remove import letter
import xlwt
from xlwt import Workbook
```

Workbook is created

```
wb = Workbook()

sheet4= wb.add_sheet('Sheet 4',cell_overwrite_ok=True)
```

Shadow removal algorithm

```
def shadow_rem(img):
    rgb_planes = cv2.split(img)

    result_planes = []
    result_norm_planes = []
    for plane in rgb_planes:
        dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
        bg_img = cv2.medianBlur(dilated_img, 21)
        diff_img = 255 - cv2.absdiff(plane, bg_img)
        norm_img = cv2.normalize(diff_img,None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX,
dtype=cv2.CV_8UC1)
        result_planes.append(diff_img)
        result_norm_planes.append(norm_img)

    result = cv2.merge(result_planes)
    return result
```

```
input the ecg
for k in range(1,26):
    if k>=20 and k<=23:
        continue

    ecg=cv2.imread('ECG ('+str(k)+').jpg')

    thresh,scal=image.img(ecg,172)                              preprocessing the image and
                                                                returning the skeletonized
                                                                image


    scal=letter.remove(scal)                                    removing letter



    col=[]
    row=[]
    for i in range(scal.shape[1]):                              Converting the image into
                                                                1D array

        for j in range(scal.shape[0]):


            if scal[j][i]==255 and scal.shape[0]-j>5 and j>1:

                col.append(i)
                row.append(scal.shape[0]-j)
                ss=(scal.shape[0]-j)
                sheet4.write(k-1,j,ss)



wb.save('data.xls')
```

S8 : MATLAB code for ECG diagnosis

```
Code                                                            Comments
layers = [
    imageInputLayer([400 1 1])                                   400X1X1 refers to number
                                                                of features per sample

    convolution2dLayer(3,16,'Padding','same')
    reluLayer
    fullyConnectedLayer(384)                                     384 refers to number of
                                                                neurons in next FC hidden
                                                                layer
```

fullyConnectedLayer(384)	384 refers to number of neurons in next FC hidden layer
fullyConnectedLayer(2)	2 refers to number of neurons in next output layer (number of output classes)
softmaxLayer classificationLayer];	
options = trainingOptions('sgdm',... 'MaxEpochs',500, ... 'Verbose',false,... 'Plots','training-progress')	