ML/DL

Code Review ML & DL

Chayanat Sangsawang 6110742035
Atdhasiri Sangchan 6110742084
Jirath Arnamnath 6110742175

01

01

Machine Learning

Overview

Feature Extraction
Machine Learning

augmentation

```python
def augment(image):
  image = tf.image.random_brightness(image, 0.1)
  image = tf.image.random_contrast(image,0.7,1.3)

  image = tf.image.random_hue(image,0.005)
  image = tf.image.random_saturation(image,0.8, 1.3)

  image = tf.image.adjust_gamma(image,1)
  image = tf.image.random_flip_left_right(image)
  image = tf.image.random_flip_up_down(image)
  return image
```

https://github.com/Atdhasiri/machineLearning/blob/main/Augmentation.py#L24-L34

Feature Extraction

```
"""**CLAHE**"""

def prebpicture(bgr):
  bgr = cv2.resize(bgr, (256, 256))
  lab = cv2.cvtColor(bgr, cv2.COLOR_BGR2LAB)
  lab_planes = cv2.split(lab)
  clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8,8))
  lab_planes[0] = clahe.apply(lab_planes[0])
  lab = cv2.merge(lab_planes)
  bgr = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  #save image to temp file
  #cv2.imwrite('/content/drive/MyDrive/Dataset/temp/clahe-output.jpg',bgr)
  return bgr
```

Picture preparation and Applying CLAHE

We need to convert pictures into size 256*256 and then apply CLAHE to every single picture and you can save it in temp file if you need to or just recieve *bgr*

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L31-L43

Feature Extraction

```
def fos(image):
  ans = []
  mean = np.mean(image)
  ans.append(mean)
  std = np.std(image)
  ans.append(std)
  var = np.var(image)
  ans.append(var)
  kurt = kurtosis(image, axis=None, fisher=True, bias=True)
  ans.append(kurt)
  sk = skew(image,axis=None)
  ans.append(sk)
  ent = entropy(image,axis=None)
  ans.append(ent)
  return ans
```

First order statistical(FoS)

We create fos by calculating the mean,std and so on.
Before using FoS you should use Greyscale for your image first

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L384-L398

Feature Extraction

```
def segment_cd(image,plotHis,plotPic):
  #PRE-PROCESSING AND SMOOTHING

  Abo,Ago,Aro = cv2.split(image)  #splitting into 3 colors


  Ar = Aro - Aro.mean()           #Preprocessing Red
  Ar = Ar - Ar.mean() - Aro.std() #Preprocessing Red
  Ar = Ar - Ar.mean() - Aro.std() #Preprocessing Red

  Mr = Ar.mean()                           #Mean of preprocessed red
  SDr = Ar.std()                           #SD of preprocessed red
  Thr = 49.5 - 12 - Ar.std()               #OD Threshold


  Ag = Ago - Ago.mean()           #Preprocessing Green
  Ag = Ag - Ag.mean() - Ago.std() #Preprocessing Green

  Mg = Ag.mean()                           #Mean of preprocessed green
  SDg = Ag.std()                           #SD of preprocessed green
  Thg = Ag.mean() + 2*Ag.std() + 49.5 + 12 #OC Threshold

  filter = signal.gaussian(99, std=6) #Gaussian Window
  filter=filter/sum(filter)

  hist,bins = np.histogram(Ag.ravel(),256,[0,256])   #Histogram of preprocessed green channel
  histr,binsr = np.histogram(Ar.ravel(),256,[0,256]) #Histogram of preprocessed red channel

  smooth_hist_g=np.convolve(filter,hist)  #Histogram Smoothing Green
  smooth_hist_r=np.convolve(filter,histr) #Histogram Smoothing Red
```

Cup and disc segmentation

We can segment the cup and disc using *cv2.split()* to split the picture into 3 colors (Abo - Blue , Ago-Green , Aro - Red) and we use Red for calculating treshold for disc and Green for calcualting treshold for cup
then we creating Filter and histogram to plot graph if you want to see the result in histogram format

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L49-L78

Feature Extraction

```
if plotHis:
  plt.subplot(2, 2, 1)
  plt.plot(hist)
  plt.title("Preprocessed Green Channel")

  plt.subplot(2, 2, 2)
  plt.plot(smooth_hist_g)
  plt.title("Smoothed Histogram Green Channel")

  plt.subplot(2, 2, 3)
  plt.plot(histr)
  plt.title("Preprocessed Red Channel")

  plt.subplot(2, 2, 4)
  plt.plot(smooth_hist_r)
  plt.title("Smoothed Histogram Red Channel")

  plt.show()
```

Plot histogram

this *plotHis* was set in *segment_cd()* 's parameter and if it's true it will plot you some graphs

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L80-L97

Feature Extraction

```python
r,c = Ag.shape
Dd = np.zeros(shape=(r,c))
Dc = np.zeros(shape=(r,c))

for i in range(1,r):
  for j in range(1,c):
    if Ar[i,j]>Thr:
      Dd[i,j]=255
    else:
      Dd[i,j]=0

for i in range(1,r):
  for j in range(1,c):
    if Ag[i,j]>Thg:
      Dc[i,j]=1
    else:
      Dc[i,j]=0

#DISPLAYING SEGMENTED OPTIC DISK AND CUP
if plotPic:
  plt.imshow(Dd, cmap = 'gray', interpolation = 'bicubic')
  plt.axis("off")
  plt.title("Optic Disk")
  plt.show()

  plt.imshow(Dc, cmap = 'gray', interpolation = 'bicubic')
  plt.axis("off")
  plt.title("Optic Cup")
  plt.show()
```

Applying Treshold and plot picture

First, it will create zero array(0 = black color) with the same size as the picture and check every pixel of the original picture and compare to it's threshold if that pixel has more value that it's threshold it will plot the zero array in the same location as white color (255 = white color) and if it lower it will plot zero. Do this with cup aswell.
and plotPic was set in *segment_cd()* 's parameter aswell so if it's true it will plot you the picture of segmented cup and disc

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L101-L129

Feature Extraction

```python
def cdr(cup,disc,plot):

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(10,10))
    #morphological closing and opening operations
    R1 = cv2.morphologyEx(cup, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(2,2)), iterations = 1)
    r1 = cv2.morphologyEx(R1, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7)), iterations = 1)
    R2 = cv2.morphologyEx(r1, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(1,21)), iterations = 1)
    r2 = cv2.morphologyEx(R2, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(21,1)), iterations = 1)
    R3 = cv2.morphologyEx(r2, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(33,33)), iterations = 1)
    r3 = cv2.morphologyEx(R3, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(43,43)), iterations = 1)

    img = clahe.apply(r3)


    ret,thresh = cv2.threshold(cup,127,255,0)
    contours,hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE) #Ge
    cup_diameter = 0
    largest_area = 0
    el_cup = contours[0]
    if len(contours) != 0:
        for i in range(len(contours)):
            if len(contours[i]) >= 5:
                area = cv2.contourArea(contours[i]) #Getting the contour with the largest area
                if (area>largest_area):
                    largest_area=area
                    index = i
                    el_cup = cv2.fitEllipse(contours[i])

    cv2.ellipse(img,el_cup,(140,60,150),3)  #fitting ellipse with the largest area
    x,y,w,h = cv2.boundingRect(contours[index]) #fitting a rectangle on the ellipse to get the length of major axis
    cup_diameter = max(w,h) #major axis is the diameter
```

Calculate cup-to-disc Ratio

Recieve to picture form *segment_cd()* and create clahe to apply with *morphologyEX()* to draw outline of the cup and disc and create contours out of cup's threshold and create elipse to cover the area of cup and then use *cv2.boundingReact()* to get the final size of ellipse and then get max width and height to get *cup_diameter*

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L138-L168

Feature Extraction

```
#morphological closing and opening operations
R1 = cv2.morphologyEx(disc, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(2,2)), iterations = 1)
r1 = cv2.morphologyEx(R1, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7)), iterations = 1)
R2 = cv2.morphologyEx(r1, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(1,21)), iterations = 1)
r2 = cv2.morphologyEx(R2, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(21,1)), iterations = 1)
R3 = cv2.morphologyEx(r2, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(33,33)), iterations = 1)
r3 = cv2.morphologyEx(R3, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(43,43)), iterations = 1)

img2 = clahe.apply(r3)

ret,thresh = cv2.threshold(disc,127,255,0)
contours,hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE) #Getting all possible contours in the segmented image
disk_diameter = 0
largest_area = 0
el_disc = el_cup
if len(contours) != 0:
      for i in range(len(contours)):
        if len(contours[i]) >= 5:
            area = cv2.contourArea(contours[i]) #Getting the contour with the largest area
            if (area>largest_area):
                largest_area=area
                index = i
                el_disc = cv2.fitEllipse(contours[i])

cv2.ellipse(img2,el_disc,(140,60,150),3) #fitting ellipse with the largest area
x,y,w,h = cv2.boundingRect(contours[index]) #fitting a rectangle on the ellipse to get the length of major axis
disk_diameter = max(w,h) #major axis is the diameter
```

Calculate disk_diameter

Do the same thing as *cup_diameter*

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L171-L196

Feature Extraction

```
def adjust_gamma(image, gamma=1.0):


    table = np.array([((i / 255.0) ** gamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")

    return cv2.LUT(image, table)
def extract_ma(image):
     r,g,b=cv2.split(image)
     comp=255-g
     clahe = cv2.createCLAHE(clipLimit=5.0, tileGridSize=(8,8))
     histe=clahe.apply(comp)
     adjustImage = adjust_gamma(histe,gamma=3)
     comp = 255-adjustImage
     J =  adjust_gamma(comp,gamma=4)
     J = 255-J
     J = adjust_gamma(J,gamma=4)

     K=np.ones((11,11),np.float32)
     L = cv2.filter2D(J,-1,K)

     ret3,thresh2 = cv2.threshold(L,125,255,cv2.THRESH_BINARY|cv2.THRESH_OTSU)
     kernel2=np.ones((9,9),np.uint8)
     tophat = cv2.morphologyEx(thresh2, cv2.MORPH_TOPHAT, kernel2)
     kernel3=np.ones((7,7),np.uint8)
     opening = cv2.morphologyEx(tophat, cv2.MORPH_OPEN, kernel3)
     return opening
```

Calculate MA

MA is one of the feature that can detect diabetic by checking swelling in the wall of a blood vessel
First, it will adjust gramma of the picture and calculate the microaneurysm and return the numpy of the result

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L219-L245

Feature Extraction

```python
def fast_glcm(img, vmin=0, vmax=255, nbit=8, kernel_size=5):
    mi, ma = vmin, vmax
    ks = kernel_size
    h,w = img.shape

    # digitize
    bins = np.linspace(mi, ma+1, nbit+1)
    gl1 = np.digitize(img, bins) - 1
    gl2 = np.append(gl1[:,1:], gl1[:,-1:], axis=1)

    # make glcm
    glcm = np.zeros((nbit, nbit, h, w), dtype=np.uint8)
    for i in range(nbit):
        for j in range(nbit):
            mask = ((gl1==i) & (gl2==j))
            glcm[i,j, mask] = 1

    kernel = np.ones((ks, ks), dtype=np.uint8)
    for i in range(nbit):
        for j in range(nbit):
            glcm[i,j] = cv2.filter2D(glcm[i,j], -1, kernel)

    glcm = glcm.astype(np.float32)
    return glcm
```

Calculate GLCM

This function will create GLCM but first you need to change your picture into greyscale before using *fast_glcm()* and it will return numpy array as a result

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L249-L272

Feature Extraction

```
def fast_glcm_mean(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm mean
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    mean = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            mean += glcm[i,j] * i / (nbit)**2

    return mean


def fast_glcm_std(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm std
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    mean = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            mean += glcm[i,j] * i / (nbit)**2

    std2 = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            std2 += (glcm[i,j] * i - mean)**2

    std = np.sqrt(std2)
    return std
```

Calculate GLCM using mean and std methods

the Mean method is using mean to calculate GLCM and the std method is using standard deviation to calculate GLCM

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L275-L306

Feature Extraction

```python
def fast_glcm_contrast(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm contrast
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    cont = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            cont += glcm[i,j] * (i-j)**2

    return cont


def fast_glcm_dissimilarity(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm dissimilarity
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    diss = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            diss += glcm[i,j] * np.abs(i-j)

    return diss
```

Calculate GLCM using contrast and dissimilarity methods

Contrast is measure of variation in intensity. Dissimilarity is the measure of relationship between pairs with similar and different intensities, Higher is the value of dissi, more is the dissimilarity between the pairs of pixels

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L309-L334

```python
def fast_glcm_homogeneity(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm homogeneity
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    homo = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            homo += glcm[i,j] / (1.+(i-j)**2)

    return homo


def fast_glcm_ASM(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm asm, energy
    '''
    h,w = img.shape
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    asm = np.zeros((h,w), dtype=np.float32)
    for i in range(nbit):
        for j in range(nbit):
            asm  += glcm[i,j]**2

    ene = np.sqrt(asm)
    return asm, ene
```

Feature Extraction

Calculate GLCM using homogeneity and ASM methods

Homogeneity is the metric used for analysis of homogeneity in an image.

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L337-L363

Feature Extraction

```python
def fast_glcm_max(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm max
    '''
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    max_  = np.max(glcm, axis=(0,1))
    return max_


def fast_glcm_entropy(img, vmin=0, vmax=255, nbit=8, ks=5):
    '''
    calc glcm entropy
    '''
    glcm = fast_glcm(img, vmin, vmax, nbit, ks)
    pnorm = glcm / np.sum(glcm, axis=(0,1)) + 1./ks**2
    ent  = np.sum(-pnorm * np.log(pnorm), axis=(0,1))
    return ent
```

Calculate GLCM using Max and Entropy methods

Max method is to find the max value of neighbor
Entro is the measure of randomness or uncertainty

https://github.com/Atdhasiri/machineLearning/blob/main/ML/final_feature_extraction.py#L366-L382

Machine learning

- **create machine learning model**

```
def create_model(model_name):
    if(model_name == "svm"):
        model = SVC(kernel='rbf',probability=True)
        #kernel = {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}
    elif(model_name == "knn"):
        model = KNeighborsClassifier(n_neighbors=5)
    elif(model_name == "rf"):
        model = RandomForestClassifier(n_estimators = 400, criterion = "gini")
    return model
```

https://github.com/Atdhasiri/machineLearning/blob/main/ML/Machine%20learning.py#L191-L199

Machine learning

- Do 5-fold cross validation.
- separate data to train set and validate set.
- scaling feature.

```
for i ,(train_index, val_index) in enumerate(kf.split(X)):

    print("---------------- fold",i+1,"----------------")
    X_train = sc.fit_transform(np.concatenate(X[train_index],axis=0))
    X_val = sc.transform(X[val_index][0])
    y_train = np.concatenate(y[train_index],axis=0)
    y_val = y[val_index][0]
```

https://github.com/Atdhasiri/machineLearning/blob/main/ML/Machine%20learning.py#L232-L238

Machine learning

```
print("training model")

model_algorithm = create_model("rf")
clf = Pipeline([
      ('feature_selection', SelectFromModel(model_algorithm)),
      ('classification', model_algorithm)
])
model = OneVsRestClassifier(model_algorithm)
model.fit(X_train, y_train)

print("train finished")
```

- **Create model**
- **Do feature selection by important weight of spacific model**
- **Apply those feature to classifier**
- **change model to be one vs rest**
- **train the model**

https://github.com/Atdhasiri/machineLearning/blob/main/ML/Machine%20learning.py#L240-L250

Machine learning

- **select best model of all fold by compare sum of micro average AUC and macro average AUC**

```
if roc_auc["micro"]+roc_auc["macro"] > best_sum_average_AUC:
    best_sum_average_AUC = roc_auc["micro"]+roc_auc["macro"]
    best_sum_3class_AUC = roc_auc[0] + roc_auc[1] + roc_auc[2]
    bestModel = model
    bestFold = i+1
elif roc_auc["micro"]+roc_auc["macro"] == best_sum_average_AUC and best_sum_3class_AUC < roc_auc[0] + roc_auc[1] + roc_auc[2]:
    best_sum_average_AUC = roc_auc["micro"]+roc_auc["macro"]
    best_sum_3class_AUC = roc_auc[0] + roc_auc[1] + roc_auc[2]
    bestModel = model
    bestFold = i+1
```

https://github.com/Atdhasiri/machineLearning/blob/main/ML/Machine%20learning.py#L270-L279

Machine learning

```
mean_tpr = np.mean(interp_tpr_all_fold, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)

std_tpr = np.std(interp_tpr_all_fold, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
plt.show()

plt.figure()
plt.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
        lw=2)

for i in range(5):
    plt.plot(fprs[i], tprs[i],label='micro-average ROC curve of fold{} (area = {:0.2f})'.format(i+1,aucs[i]), alpha=.5)

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Mean ROC of all fold')
plt.legend(loc="lower right")
plt.show()
```

- **plot mean ROC curve and calculate SD**

https://github.com/Atdhasiri/machineLearning/blob/main/ML/Machine%20learning.py#L287-L312

02

Deep Learning

Overview

Convert Image
Deep Learning

Convert Image

```python
def save_to_numpy(imgs,label,path,filename):
    np.save(os.path.join(path, filename + '.npy'), imgs)
    np.save(os.path.join(path, filename + '_label' + '.npy'), label)
    print('Numpy files have been saved')
```

Save picture in numpy

This function will create .npy file in the path that you define

https://github.com/Atdhasiri/machineLearning/blob/main/DL/convert%20image%20to%20numpy.py#L23-L26

Convert Image

```python
classes = ['Glaucoma', 'Normal', 'Other']
num_folders = ['1','2','3','4','5']
temp_x = []
temp_y = []
x=[]
y=[]
```

Declaration

Declare the Classes and Fold that you have and create temp array for x (data) and temp array for y (label) and the real x and y. I created temp because I will create nested array which mean len(x) will answer 5 (Fold) and in each it will contain pictures

https://github.com/Atdhasiri/machineLearning/blob/main/DL/convert%20image%20to%20numpy.py#L31-L36

Convert Image

```python
for k in range(len(num_folders)):
  for i in range(len(classes)):
    images_lst = os.listdir('/content/drive/MyDrive/Dataset/Train/' + classes[i] + '/Fold-' + num_folders[k])
    #print(images_lst)
    count = 0

    for j in images_lst:
      count += 1
      if count == 2:
        break
      print('\n' + j + ' ----> ', count, '/', len(images_lst))
      print('class : ' + classes[i])
      print('fold : ' + num_folders[k])
      img = cv2.imread('/content/drive/MyDrive/Dataset/Train/' + classes[i] + '/Fold-' + num_folders[k]+'/'+j)
      img = cv2.resize(img, (256, 256))
      temp_x.append(np.array(img))
      temp_y.append(i)

  x.append(temp_x)
  temp_x = []
  y.append(temp_y)
  temp_y = []

#save to .npy

#output_path = '/content/drive/MyDrive/Dataset/np_data/'
#save_to_numpy(x,y,output_path,'Train')
print('----------------------------------------------------------------')
print('done')
```

Loop for every pictures in Train

It will loop every Fold that you have and then every Classes that you have and then it will resize the image to 256*256 and append to temp_x first and temp_y will appen the number of class
After the finished a Fold the temp_x and temp_y will append to real x and y to add a nested array and do this to another Fold

https://github.com/Atdhasiri/machineLearning/blob/main/DL/convert%20image%20to%20numpy.py#L37-L265

Convert Image

```python
classes = ['Glaucoma', 'Normal', 'Other']
#num_folders = ['1','2','3','4','5']
#temp_x = []
#temp_y = []
x=[]
y=[]

for i in range(len(classes)):
  images_lst = os.listdir('/content/drive/MyDrive/Dataset/Test/' + classes[i])
  #print(images_lst)
  count = 0

  for j in images_lst:
    count += 1
    print('\n' + j + ' ----> ', count, '/', len(images_lst))
    print('class : ' + classes[i])
    #print('fold : ' + num_folders[k])
    img = cv2.imread('/content/drive/MyDrive/Dataset/Test/' + classes[i] +'/'+ j)
    img = cv2.resize(img, (256, 256))
    x.append(np.array(img))
    if classes[i] == 'Glaucoma':
      y.append('0')
      print('Y appended : 0')
    elif classes[i] == 'Normal':
      y.append('1')
      print('Y appended : 1')
    else:
      y.append('2')
      print('Y appended : 2')


#save to .npy

#output_path = '/content/drive/MyDrive/Dataset/np_data/'
#save_to_numpy(x,y,output_path,'test folder not the training')
print('----------------------------------------------------------------')
print('done')
```

Loop for every pictures in Test Folder

For dataset Test we don't have Fold so we just loop every pictures and add it directly to the x(data) array and y(label) array

https://github.com/Atdhasiri/machineLearning/blob/main/DL/convert%20image%20to%20numpy.py#L72-L108

Deep learning

- load data

```
X = np.load("/content/drive/MyDrive/Colab Notebooks/Dataset/np_data/Train.npy",allow_pickle=True)
y = np.load("/content/drive/MyDrive/Colab Notebooks/Dataset/np_data/TRAIN_label.npy",allow_pickle=True)

X_test = np.load("/content/drive/MyDrive/Colab Notebooks/Dataset/np_data/test.npy",allow_pickle=True)
y_test = np.load("/content/drive/MyDrive/Colab Notebooks/Dataset/np_data/test_label.npy",allow_pickle=True)

X = np.array([np.array(fold) for fold in X])
y = np.array([np.array(fold) for fold in y])
X_test = np.array(X_test)
y_test = np.array(y_test)
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L42-L51

Deep learning

- **create model**

```python
def build_model(base_model_name):

    print('Creating Model : {}.......'.format(base_model_name.upper()))
    if base_model_name == 'DenseNet121' :
        init_model = DenseNet121(input_shape= (height, width ,3), include_top=False, weights='imagenet', pooling='avg')
        x = init_model.output
        out_layer = Dense(num_output, activation="softmax")(x)
        model = Model(init_model.input, out_layer)
    elif base_model_name == 'ResNet101V2' :
        init_model = ResNet101V2(input_shape= (height, width ,3), include_top=False, weights='imagenet', pooling='avg')
        x = init_model.output
        out_layer = Dense(num_output, activation="softmax")(x)
        model = Model(init_model.input,out_layer)
    else :
        sys.exit("Model Not Support")
    return model
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L225-L240

Deep learning

- create early stop function to early stop training if validation loss not decrease for 10 epochs
- create reduce learning rate function to reduce learning rate if validation loss not decrease for 7 epochs

```
earlyStopping = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='min')

reduce_lr_loss = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=7, verbose=1, epsilon=1e-4, mode='min')
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L242-L244

Deep learning

- Create and compile model
- apply optimizer(adam) and loss algorithm

```
model_name = "DenseNet121"
model = build_model(model_name)
#model.summary()

#class_weights = get_class_weights(y_train[train_idx])
model.compile(optimizer=optimizers['adam'], loss='categorical_crossentropy', metrics=['accuracy'])
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L258-L263

Deep learning

- **save weight at the best epoch(focus on val_loss)**

```
checkpoint_path =  "/content/drive/MyDrive/Colab Notebooks/Dataset/weight/{}-weights_fold-{}.hdf5".format(model_name,i+1)
mcp_save = ModelCheckpoint(checkpoint_path, save_best_only=True, monitor='val_loss', mode='min',save_weights_only=True)
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L265-L266

Deep learning

```
history = model.fit(X_train,y_train,
  steps_per_epoch = (len(X_train) / batch_size ),
  epochs = 50,
  validation_data=(X_val, y_val),
  callbacks=[earlyStopping, mcp_save, reduce_lr_loss]
  )
```

- **train model**

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L298-L303

Deep learning

- **select best model of all fold by compare sum of micro average AUC and macro average AUC**

```
if roc_auc["micro"]+roc_auc["macro"] > best_sum_average_AUC:
    best_sum_average_AUC = roc_auc["micro"]+roc_auc["macro"]
    best_sum_3class_AUC = roc_auc[0] + roc_auc[1] + roc_auc[2]
    bestModel = model
    bestFold = i+1
elif roc_auc["micro"]+roc_auc["macro"] == best_sum_average_AUC and best_sum_3class_AUC < roc_auc[0] + roc_auc[1] + roc_auc[2]:
    best_sum_average_AUC = roc_auc["micro"]+roc_auc["macro"]
    best_sum_3class_AUC = roc_auc[0] + roc_auc[1] + roc_auc[2]
    bestModel = model
    bestFold = i+1
```

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L312-L321

Evaluation

```
def calculateFprTprAuc(binary_y_test, y_score):
    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(binary_y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(binary_y_test.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
    return fpr , tpr , roc_auc
```

Calculate FPR TPR and AUC score of each classes and micro average

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L53-L67

```python
def plotROC(fpr,tpr,roc_auc,title):
    #aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    #interpolate all ROC curves
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += interp(all_fpr, fpr[i], tpr[i])

    #average tpr and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    # Plot all ROC curves
    plt.figure()
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro-average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["macro"]),
             color='navy', linestyle=':', linewidth=4)

    colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                 label='ROC curve of {0} (area = {1:0.2f})'
                 ''.format(classes_list[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=lw)
    plt.xlim([0.0, 1.0])
```

Evaluation

calculate macro average and Plot ROC curve of each classes(One vs Rest) and plot micro average and macro average of them

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L69-L113

Evaluation

1

```python
def plot_confusion_matrix(y_true, y_pred, classes,
                          title=None,
                          cmap=plt.cm.Blues):

    np.set_printoptions(precision=2)

    if not title:
        title = 'Normalized confusion matrix'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]

    cm_normalize = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")

    #print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm_normalize, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm_normalize.shape[1]),
           yticks=np.arange(cm_normalize.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')
```

Plot 3 classes confusion metrix with normalization(percentage)

2

```python
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
fmt = '.2f' #if normalize else 'd'
thresh = cm_normalize.max() / 2.
for i in range(cm_normalize.shape[0]):
    for j in range(cm_normalize.shape[1]):
        ax.text(j, i, "{:.2%}\n({:d})".format(cm_normalize[i, j], cm[i, j]),
                ha="center", va="center",
                color="white" if cm_normalize[i, j] > thresh else "black")
fig.tight_layout()
#plt.xlim(-0.5, len(np.unique(y))-0.5)
#plt.ylim(len(np.unique(y))-0.5, -0.5)
plt.show()
```

1. https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L118-L150
2. https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L153-L167

Evaluation

```python
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
# Precision or positive predictive value
PPV = TP/(TP+FP)
# Negative predictive value
NPV = TN/(TN+FN)
# Fall out or false positive rate
FPR = FP/(FP+TN)
# False negative rate
FNR = FN/(TP+FN)
# False discovery rate
FDR = FP/(TP+FP)
# F1-score
F1 = 2*((PPV*TPR)/(PPV+TPR))
# Overall accuracy
ACC = (TP+TN)/(TP+FP+FN+TN)

print("="*60)
print("{:<26} | {:>8} {:>8} {:>8}".format("value", "Glaucoma", "Normal", "Other"))
print("="*60)
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("Sensitivity(TPR)", TPR[0], TPR[1], TPR[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("Specificity(TNR)", TNR[0], TNR[1], TNR[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("Precision(PPV)", PPV[0], PPV[1], PPV[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("Negative predictive value", NPV[0], NPV[1], NPV[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("False positive rate", FPR[0], FPR[1], FPR[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("False negative rate", FNR[0], FNR[1], FNR[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("False discovery rate", FDR[0], FDR[1], FDR[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("F1-score", F1[0], F1[1], F1[2]))
print("{:<26} | {:>8.2f} {:>8.2f} {:>8.2f}".format("Overall accuracy", ACC[0], ACC[1], ACC[2]))
```

Calculate and print sensitivity , specificity and etc. by the value from confusion matrix

https://github.com/Atdhasiri/machineLearning/blob/main/DL/Deep%20learning.py#L181-L210

ML&DL

Thanks for attending!