Retinal Blood vessel segmentation using a CNN

Done in 02/2019
Modified on 09/2022 to be compatible with the latest version of tensorflow
Sadegh

Setting up the google drive

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, cal
```

```
Train = True  #switch for training
```

Loading and preparing data

```
import os
import numpy as np
from PIL import Image
from IPython.display import display
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.activations import *
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLR
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

```
print(tf.__version__)
```

```
2.8.2
```

```
np.random.seed(70)
tf.random.set_seed(70)
```

```
data_path = '/content/gdrive/My Drive/Retinal/training/'
```

```

image_width = 565
image_height = 584
```

```
img_data_path = os.path.join(data_path, 'images')
seg_data_path = os.path.join(data_path, '1st_manual')
print(img_data_path)
print(seg_data_path)
```

```
/content/gdrive/My Drive/Retinal/training/images
/content/gdrive/My Drive/Retinal/training/1st_manual
```

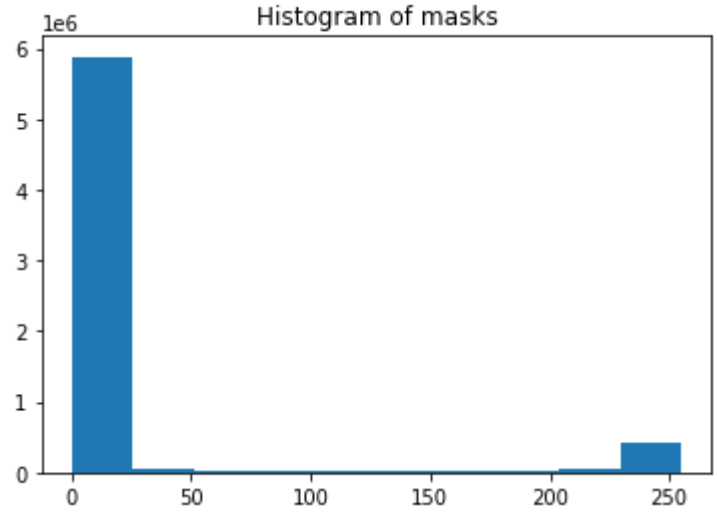
```
def create_data_arr(img_data_path):
  images = sorted(os.listdir(img_data_path))
  num_images = len(images)
  imgs_array = np.ndarray((num_images, image_width, image_height), dtype=np.uin

  n = 0
  good_imgs=[]
  for image_name in images:
      img = Image.open(os.path.join(img_data_path,image_name)).convert('L')
      good_imgs.append(image_name)
      img = np.array(img.resize((image_height,image_width)))
      imgs_array[n] = img
      n +=1
  return imgs_array

print('Creating training images array...')
imgs_array = create_data_arr(img_data_path)
np.save('/content/gdrive/My Drive/Retinal/training/imgs_train.npy', imgs_array)
print(imgs_array.shape)

print('Creating training masks array...')
segs_array = create_data_arr(seg_data_path)
np.save('/content/gdrive/My Drive/Retinal/training/seg_train.npy', segs_array)
print(segs_array.shape)

```

```
Creating training images array...
(20, 565, 584)
Creating training masks array...
(20, 565, 584)
```

```
del imgs_array
del segs_array

imgs = np.load(data_path+'imgs_train.npy')
segs = np.load(data_path+'seg_train.npy')

hh = plt.hist(segs.flatten())
plt.title("Histogram of masks")

img_no=11
```

```
print(imgs.shape)
w=Image.fromarray(imgs[img_no].reshape(image_width,image_height))
#display(w)

print(segs.shape)
w=Image.fromarray(segs[img_no].reshape(image_width,image_height))
#display(w)
```

```
(20, 565, 584)
(20, 565, 584)
```




```
def create_patches(imgs,segs, patch_size=128, step_size=16):
  #making patches of size patch_size*patch_size and stride=step_size
  from skimage.util.shape import view_as_windows

  imgs_patches = np.array([])
  segs_patches = np.array([])
  for i in range(imgs.shape[0]):
    patches=view_as_windows(imgs[i], (patch_size, patch_size), step=step_size)
    patches=patches.reshape(-1,patch_size,patch_size)
    seg_pats=view_as_windows(segs[i], (patch_size, patch_size), step=step_size)
    seg_pats=seg_pats.reshape(-1,patch_size,patch_size)

    if i==0:
      #first run
      imgs_patches=patches
      segs_patches=seg_pats
    else:
      imgs_patches=np.append(imgs_patches,patches,axis=0)
      segs_patches=np.append(segs_patches,seg_pats,axis=0)

  imgs_patches = imgs_patches[:,:,:,np.newaxis]
  segs_patches = segs_patches[:,:,:,np.newaxis]

  imgs_patches = imgs_patches.astype('float32')
  imgs_patches /=255.0
  segs_patches = segs_patches.astype('float32')
  segs_patches /=255.0

  return imgs_patches,segs_patches
```

```
#Splitting the data
train_imgs, valid_test_imgs, train_segs, valid_test_segs = \
                    train_test_split(imgs, segs, test_size=0.5, random_state=54

valid_imgs, test_imgs, valid_segs, test_segs = \
                    train_test_split(valid_test_imgs, valid_test_segs, test_siz

print(train_imgs.shape,train_segs.shape, valid_imgs.shape,valid_segs.shape,test_

#Creating patches
TrainPatchImg, TrainPatchSeg = create_patches(train_imgs,train_segs ,step_size=8
ValidPatchImg, ValidPatchSeg = create_patches(valid_imgs,valid_segs, step_size=1
TestPatchImg, TestPatchSeg = create_patches(test_imgs,test_segs, step_size=128)
TrainPatchImgEval, TrainPatchSegEval = create_patches(train_imgs,train_segs ,ste
print(TrainPatchImg.shape,TrainPatchSeg.shape, ValidPatchImg.shape,ValidPatchSeg
```

```
(10, 565, 584) (10, 565, 584) (5, 565, 584) (5, 565, 584) (5, 565, 584) (5, 5
(31900, 128, 128, 1) (31900, 128, 128, 1) (80, 128, 128, 1) (80, 128, 128, 1)
```

```
'''
del imgs
del segs
del train_imgs
del valid_imgs
del test_imgs
import gc
gc.collect()
'''
```

```
'\ndel imgs\ndel segs\ndel train_imgs\ndel valid_imgs\ndel test_imgs\nimport
gc\ngc.collect()\n'
```

Training

Model architecture

```
def BN_ReLU(input_features):
    output_features = BatchNormalization()(input_features)
    output_features = relu(output_features)
    return output_features

def Model_Architecture(input_layer):
    initial_num_filters=16
    input_layer = BatchNormalization()(input_layer)  #sd
    #in:128--out:64
    conv1 = Conv2D(initial_num_filters * 1, (3, 3), padding="same")(input_layer
    out = BN_ReLU(conv1)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
```

```
    #in:64--out:32
    conv2 = Conv2D(initial_num_filters * 2, (3, 3), padding="same")(out)
    out = BN_ReLU(conv2)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
    #in:32--out:16
    conv3 = Conv2D(initial_num_filters * 3, (3, 3), padding="same")(out)
    out = BN_ReLU(conv3)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
    #in:16--out:8
    conv4 = Conv2D(initial_num_filters * 4, (3, 3), padding="same")(out)
    out = BN_ReLU(conv4)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
    #in:8--out:4
    conv5 = Conv2D(initial_num_filters * 5, (3, 3), padding="same")(out)
    out = BN_ReLU(conv5)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
    #in:4--out:2
    conv6 = Conv2D(initial_num_filters * 6, (3, 3), padding="same")(out)
    out = BN_ReLU(conv6)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)
    #in:2--out:1
    conv7 = Conv2D(initial_num_filters * 7, (3, 3), padding="same")(out)
    out = BN_ReLU(conv7)
    out = MaxPooling2D((2, 2))(out)
    out = Dropout(0.2)(out)

    #Decoding
    convT7 = Conv2DTranspose(initial_num_filters * 7, (3, 3), strides=(2, 2), pa
    convT7 = Conv2D(initial_num_filters * 7, (3, 3), padding="same")(convT7)
    out = concatenate([convT7, conv7])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    convT6 = Conv2DTranspose(initial_num_filters * 6, (3, 3), strides=(2, 2), pa
    convT6 = Conv2D(initial_num_filters * 6, (3, 3), padding="same")(convT6)
    out = concatenate([convT6, conv6])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    convT5 = Conv2DTranspose(initial_num_filters * 5, (3, 3), strides=(2, 2), pa
    convT5 = Conv2D(initial_num_filters * 5, (3, 3), padding="same")(convT5)
    out = concatenate([convT5, conv5])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    convT4 = Conv2DTranspose(initial_num_filters * 4, (3, 3), strides=(2, 2), pa
    convT4 = Conv2D(initial_num_filters * 4, (3, 3), padding="same")(convT4)
    out = concatenate([convT4, conv4])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)
```

```
    convT3 = Conv2DTranspose(initial_num_filters * 3, (3, 3), strides=(2, 2), pa
    convT3 = Conv2D(initial_num_filters * 3, (3, 3), padding="same")(convT3)
    out = concatenate([convT3, conv3])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    convT2 = Conv2DTranspose(initial_num_filters * 2, (3, 3), strides=(2, 2), pa
    convT2 = Conv2D(initial_num_filters * 2, (3, 3), padding="same")(convT2)
    out = concatenate([convT2, conv2])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    convT1 = Conv2DTranspose(initial_num_filters * 1, (3, 3), strides=(2, 2), pa
    convT1 = Conv2D(initial_num_filters * 1, (3, 3), padding="same")(convT1)
    out = concatenate([convT1, conv1])
    out = BN_ReLU(out)
    out = Dropout(0.2)(out)

    output_layer = Conv2D(1, (1,1), padding="same", activation="sigmoid")(out)
    return output_layer


input_layer=Input(shape=(128,128,1))
output_layer=Model_Architecture(input_layer)

model=Model(inputs=input_layer,outputs=output_layer)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected t
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 1       0           []
                             )]

 batch_normalization (BatchNorm  (None, 128, 128, 1)  4           ['input_1[0
 alization)

 conv2d (Conv2D)             (None, 128, 128, 16       160         ['batch_no
                             )

 batch_normalization_1 (BatchNo  (None, 128, 128, 16  64         ['conv2d[0
 rmalization)                )

 tf.nn.relu (TFOpLambda)     (None, 128, 128, 16       0           ['batch_no
                             )

 max_pooling2d (MaxPooling2D)   (None, 64, 64, 16)    0           ['tf.nn.re

 dropout (Dropout)           (None, 64, 64, 16)    0           ['max_pool

 conv2d_1 (Conv2D)           (None, 64, 64, 32)    4640        ['dropout[0

 batch_normalization_2 (BatchNo  (None, 64, 64, 32)  128         ['conv2d_1
```

```
rmalization)

tf.nn.relu_1 (TFOpLambda)      (None, 64, 64, 32)   0           ['batch_no

max_pooling2d_1 (MaxPooling2D)  (None, 32, 32, 32)  0           ['tf.nn.re

dropout_1 (Dropout)            (None, 32, 32, 32)   0           ['max_pool

conv2d_2 (Conv2D)              (None, 32, 32, 48)   13872       ['dropout_

batch_normalization_3 (BatchNo  (None, 32, 32, 48)  192         ['conv2d_2
rmalization)

tf.nn.relu_2 (TFOpLambda)      (None, 32, 32, 48)   0           ['batch_no

max_pooling2d_2 (MaxPooling2D)  (None, 16, 16, 48)  0           ['tf.nn.re

dropout_2 (Dropout)            (None, 16, 16, 48)   0           ['max_pool

conv2d_3 (Conv2D)              (None, 16, 16, 64)   27712       ['dropout_

batch_normalization_4 (BatchNo  (None, 16, 16, 64)  256         ['conv2d_3
rmalization)

tf.nn.relu_3 (TFOpLambda)      (None, 16, 16, 64)   0           ['batch_no

max_pooling2d_3 (MaxPooling2D)  (None, 8, 8, 64)    0           ['tf.nn.re

dropout_3 (Dropout)            (None, 8, 8, 64)     0           ['max_pool

conv2d_4 (Conv2D)              (None, 8, 8, 80)     46160       ['dropout_
```

```
import tensorflow as tf
from tensorflow.keras import backend as K
def focal_loss(gamma=2., alpha=.60):
    def focal_loss_fixed(y_true, y_pred):
        pt1 = tf.where(tf.equal(y_true, 1), y_pred, tf.ones_like(y_pred))
        pt0 = tf.where(tf.equal(y_true, 0), y_pred, tf.zeros_like(y_pred))
        # clip to prevent NaN's and Inf's
        pt1 = K.clip(pt1, K.epsilon(), 1-K.epsilon())
        pt0 = K.clip(pt0, K.epsilon(), 1-K.epsilon())
        return -K.mean(alpha * K.pow(1. - pt1, gamma) * K.log(pt1), axis=-1) \
               -K.mean((1-alpha) * K.pow( pt0, gamma) * K.log(1. - pt0) ,axis=-
    return focal_loss_fixed

#loss =[focal_loss(gamma=0.0,alpha=0.60)], 'categorical_crossentropy'
```

```
model.compile(optimizer=Adam(lr=0.001), loss = [focal_loss(gamma=0.1,alpha=0.60

early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=8, verb
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, verbos
model_checkpoint = ModelCheckpoint(data_path+"checkpoint/keras.model", monitor=
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWa
  super(Adam, self).__init__(name, **kwargs)
```

```
batchsize=128

gen_args = dict(horizontal_flip=True,
                     vertical_flip=True)
image_gen = ImageDataGenerator(**gen_args)
mask_gen = ImageDataGenerator(**gen_args)

seed = 854
image_generator = image_gen.flow(TrainPatchImg, batch_size=batchsize, shuffle=T
mask_generator = mask_gen.flow(TrainPatchSeg, batch_size=batchsize, shuffle=Tru

train_generator = zip(image_generator, mask_generator)
if Train:
  model_history=model.fit(
                train_generator,
                steps_per_epoch=np.ceil(len(TrainPatchImg)/batchsize),
                validation_data = (ValidPatchImg, ValidPatchSeg),
                verbose=1,
                callbacks=[early_stopping, model_checkpoint, reduce_lr],
                initial_epoch=0,
                epochs=30)
```

```
Epoch 1/30
250/250 [==============================] - ETA: 0s - loss: 0.0876 - accuracy:
Epoch 1: val_loss improved from inf to 0.13829, saving model to /content/gdri
250/250 [==============================] - 99s 340ms/step - loss: 0.0876 - ac
Epoch 2/30
250/250 [==============================] - ETA: 0s - loss: 0.0335 - accuracy:
Epoch 2: val_loss improved from 0.13829 to 0.09217, saving model to /content/
250/250 [==============================] - 83s 330ms/step - loss: 0.0335 - ac
Epoch 3/30
250/250 [==============================] - ETA: 0s - loss: 0.0262 - accuracy:
Epoch 3: val_loss improved from 0.09217 to 0.04062, saving model to /content/
250/250 [==============================] - 83s 331ms/step - loss: 0.0262 - ac
Epoch 4/30
250/250 [==============================] - ETA: 0s - loss: 0.0233 - accuracy:
Epoch 4: val_loss improved from 0.04062 to 0.03082, saving model to /content/
250/250 [==============================] - 83s 331ms/step - loss: 0.0233 - ac
Epoch 5/30
250/250 [==============================] - ETA: 0s - loss: 0.0215 - accuracy:
Epoch 5: val_loss improved from 0.03082 to 0.02945, saving model to /content/
250/250 [==============================] - 83s 330ms/step - loss: 0.0215 - ac
Epoch 6/30
250/250 [==============================] - ETA: 0s - loss: 0.0202 - accuracy:
Epoch 6: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 300ms/step - loss: 0.0202 - ac
Epoch 7/30
250/250 [==============================] - ETA: 0s - loss: 0.0193 - accuracy:
Epoch 7: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 301ms/step - loss: 0.0193 - ac
Epoch 8/30
250/250 [==============================] - ETA: 0s - loss: 0.0186 - accuracy:
Epoch 8: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 300ms/step - loss: 0.0186 - ac
Epoch 9/30
250/250 [==============================] - ETA: 0s - loss: 0.0179 - accuracy:
```

```
Epoch 9: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 299ms/step - loss: 0.0179 - ac
Epoch 10/30
250/250 [==============================] - ETA: 0s - loss: 0.0173 - accuracy:
Epoch 10: val_loss did not improve from 0.02945

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
250/250 [==============================] - 75s 299ms/step - loss: 0.0173 - ac
Epoch 11/30
250/250 [==============================] - ETA: 0s - loss: 0.0166 - accuracy:
Epoch 11: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 299ms/step - loss: 0.0166 - ac
Epoch 12/30
250/250 [==============================] - ETA: 0s - loss: 0.0165 - accuracy:
Epoch 12: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 299ms/step - loss: 0.0165 - ac
Epoch 13/30
250/250 [==============================] - ETA: 0s - loss: 0.0164 - accuracy:
Epoch 13: val_loss did not improve from 0.02945
250/250 [==============================] - 75s 299ms/step - loss: 0.0164 - ac
Epoch 13: early stopping
```

```
1 if Train==True:
2   plt.figure(figsize=(8, 8))
3   plt.plot(model_history.history["loss"], label="Training loss")
4   plt.plot(model_history.history["val_loss"], label="Validation loss")
5   plt.plot( np.argmin(model_history.history["val_loss"]), np.min(model_history.h
6   plt.xlabel("Epochs")
7   plt.ylabel("Loss")
8   plt.legend()
9
```

Loading the trained model

```
1 from tensorflow.keras.models import load_model
2 model = load_model(data_path+'checkpoint/keras.model', custom_objects={'focal_l
```

```
1 '''
2 c=0  #0 or 1
3
4 imageNo=23 #40 80
5 #image
6 w=Image.fromarray(255*PatchImg[imageNo,:,:,c]).convert('L')
7 display(w)
8
9 #real seg
10 w=Image.fromarray(255*PatchSeg[imageNo,:,:,c]).convert('L')
11 display(w)
12
13 #predicted seg
14 w=Image.fromarray(255*predicted_[imageNo,:,:,c]).convert('L')
15 display(w)
16
17 #predicted seg -- rounded with threshold
18 thresh=0.5
19 preds_valid_rounded = np.round(predicted_>thresh)
20 preds_valid_rounded=preds_valid_rounded.astype(np.float32)
21
22 w=Image.fromarray(255*preds_valid_rounded[imageNo,:,:,c].astype('uint8')).conve
23 display(w)
24 '''
```

```
'\nc=0  #0 or 1\n\nimageNo=23 #40 80\n#image\nw=Image.fromarray(255*PatchImg
[imageNo,:,:,c]).convert('L')\ndisplay(w)\n\n#real seg\nw=Image.fromarray(255
*PatchSeg[imageNo,:,:,c]).convert('L')\ndisplay(w)\n\n#predicted seg\nw=Imag
e.fromarray(255*predicted_[imageNo,:,:,c]).convert('L')\ndisplay(w)\n\n#predi
cted seg -- rounded with threshold\nthresh=0.5\npreds_valid_rounded = np.roun
d(predicted_>thresh)\npreds_valid_rounded=preds_valid_rounded.astype(np.float
```

Evaluation of the trained model

```
1 #defining dice metric
2 smooth = 1
3 def dice_coef(y_true, y_pred):
4     y_true_f = K.flatten(y_true)
5     y_pred_f = K.flatten(y_pred)
6     intersection = K.sum(y_true_f * y_pred_f)
7     return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) +
```

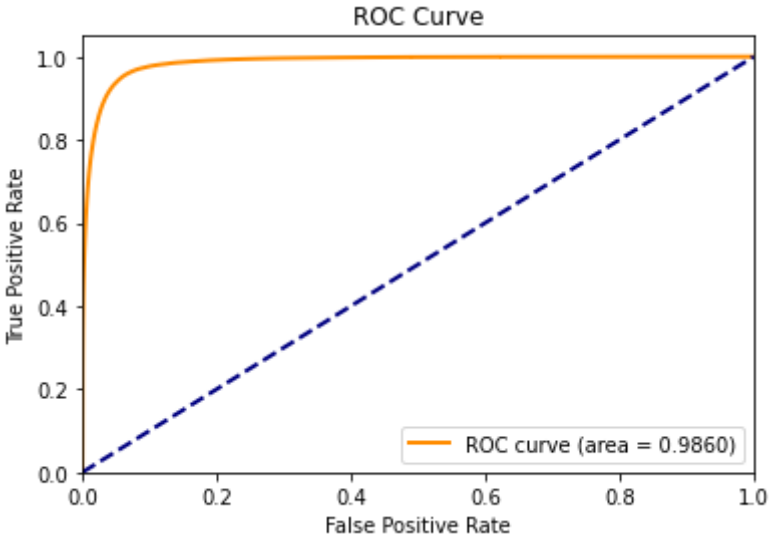
```

def dice_coef_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)


#Confusion Matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        #print("Normalized confusion matrix")
    else:
        #print('Confusion matrix, without normalization')
        pass

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


```

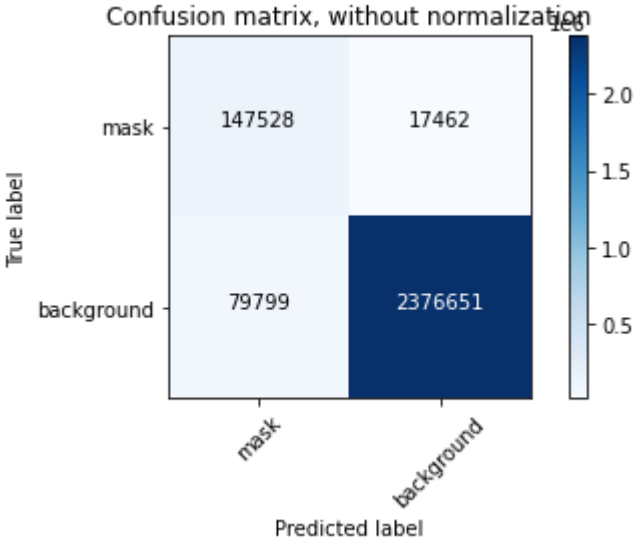
```
def generate_report(true_seg,predicted_):
  thresh=0.5
  preds_rounded = np.round(predicted_>thresh)
  preds_rounded=preds_rounded.astype(np.float32)

  c=0  #0 or 1
  y_true = true_seg[:,:,:,c].flatten().astype(int)
  y_pred = preds_rounded[:,:,:,c].flatten()  #with a specific thresh
  y_pred_prob = predicted_[:,:,:,c].flatten()

  from sklearn import metrics
  acc=metrics.accuracy_score(y_true, y_pred, normalize=True)
  print('Accuracy = '+str(acc))
```

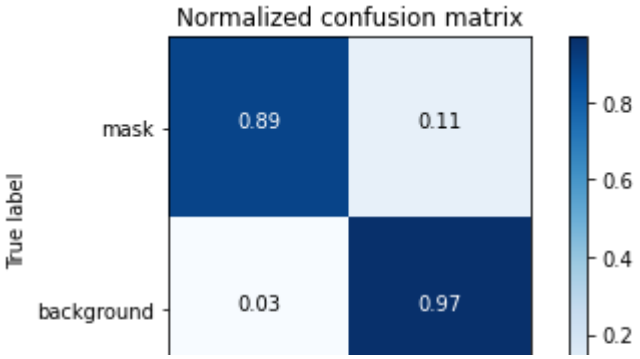
```

  dice_result=dice_coef(true_seg[:,:,:,0],preds_rounded[:,:,:,0])
  print('Dice='+str(K.eval(dice_result)))

  report=metrics.classification_report(y_true, y_pred, digits=4)
  print(report)


  #Plot ROC curve
  fpr, tpr, thresholds = metrics.roc_curve(y_true, y_pred_prob, pos_label=1)
  roc_auc = metrics.auc(fpr, tpr)
  plt.figure()
  lw = 2
  plt.plot(fpr, tpr, color='darkorange',
          lw=lw, label='ROC curve (area = %0.4f)' % roc_auc)
  plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.05])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('ROC Curve')
  plt.legend(loc="lower right")
  plt.show()


  from sklearn.metrics import confusion_matrix
  cnf_matrix = confusion_matrix(y_true, y_pred, labels=[1,0])
  #positive:vessel:1:mask:white
  #nagative:backgrnd:0:black

  np.set_printoptions(precision=2)
  class_names=['mask','background']

  # Plot non-normalized confusion matrix
  plt.figure()
  plot_confusion_matrix(cnf_matrix, classes=class_names,
                        title='Confusion matrix, without normalization');

  # Plot normalized confusion matrix
  plt.figure()
  plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                        title='Normalized confusion matrix');

  plt.show()
```

```
print('Evaluation on the TRAINING set')
predicted_ = model.predict(TrainPatchImgEval)
generate_report(TrainPatchSegEval,predicted_)
```

```
Evaluation on the TRAINING set
Accuracy = 0.9628978729248047
Dice=0.78898644
              precision    recall  f1-score   support

           0     0.9927    0.9675    0.9799   2456450
           1     0.6490    0.8942    0.7521    164990

    accuracy                         0.9629   2621440
   macro avg     0.8208    0.9308    0.8660   2621440
weighted avg     0.9711    0.9629    0.9656   2621440
```

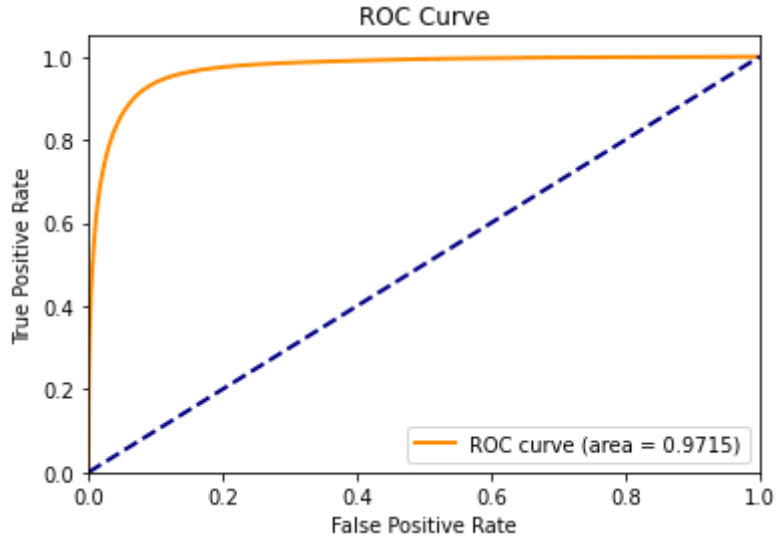


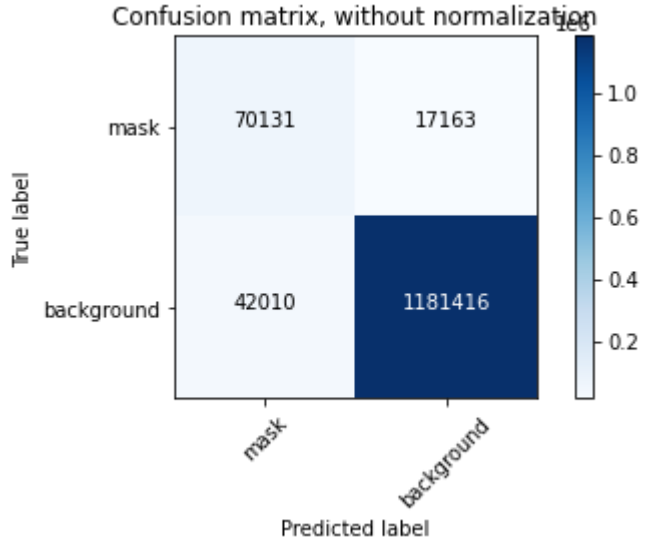


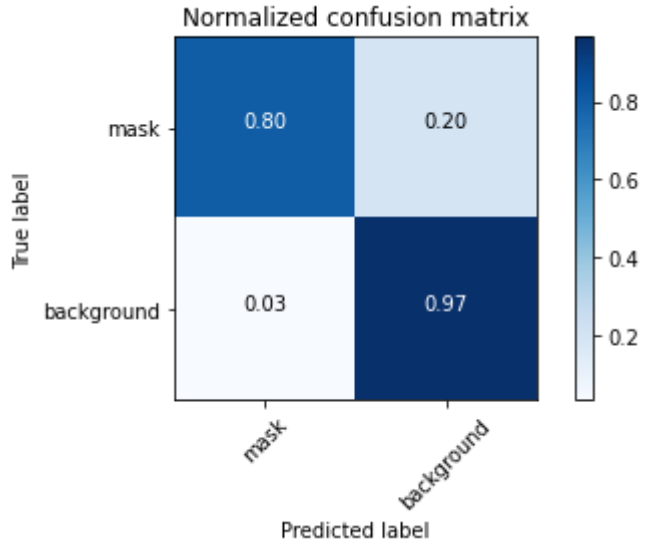



```
print('Evaluation on the VALIDATION set')
predicted_ = model.predict(ValidPatchImg)
generate_report(ValidPatchSeg,predicted_)
```

```
Evaluation on the VALIDATION set
Accuracy = 0.9548545837402344
Dice=0.73072815
              precision    recall  f1-score   support

           0     0.9857    0.9657    0.9756   1223426
           1     0.6254    0.8034    0.7033     87294

    accuracy                         0.9549   1310720
   macro avg     0.8055    0.8845    0.8394   1310720
weighted avg     0.9617    0.9549    0.9574   1310720
```







```
print('Evaluation on the TEST set')
predicted_ = model.predict(TestPatchImg)
generate_report(TestPatchSeg,predicted_)
```

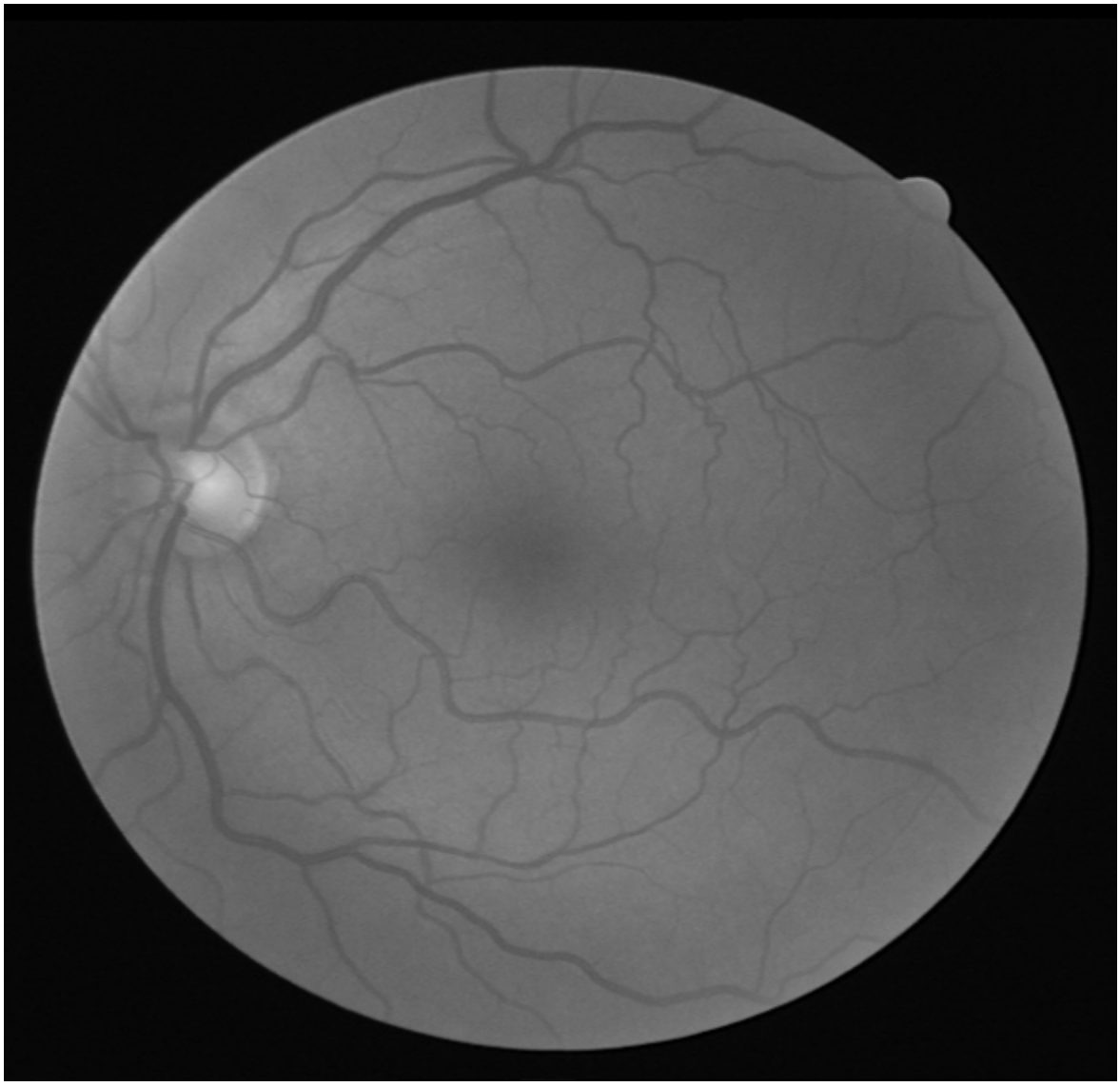
```
Evaluation on the TEST set
Accuracy = 0.9619827270507812
Dice=0.7477367
              precision    recall  f1-score   support

           0     0.9899    0.9694    0.9795   1230579
           1     0.6435    0.8482    0.7318     80141
```

Display on the test set

```
imgs = test_imgs
#resize to 640*640
imgs_resized = np.ndarray((imgs.shape[0], 640, 640), dtype=np.uint8)
n=0
import cv2
for img in imgs:
  res = cv2.resize(img, dsize=(640, 640), interpolation=cv2.INTER_CUBIC)
  imgs_resized[n]=res
  n=n+1

#making patches of size 128*128
patch_size=128
step_size=128

from skimage.util.shape import view_as_windows
'''
imgs_patches = np.array([])
for i in range(imgs_resized.shape[0]):
  patches=view_as_windows(imgs_resized[i], (patch_size, patch_size), step=step_s
  patches=patches.reshape(-1,patch_size,patch_size, order='A')
  if i==0:
    imgs_patches=patches
  else:
    imgs_patches=np.append(imgs_patches,patches,axis=0)
 '''

imgs_patches = np.ndarray((500, patch_size, patch_size), dtype=np.uint8)
p=0
for i in range(imgs_resized.shape[0]):
  #patches=view_as_windows(imgs_resized[i], (patch_size, patch_size), step=step_
  for r in range(5):
      for c in range(5):
        imgs_patches[p]=imgs_resized[i,r*128:(r+1)*128,c*128:(c+1)*128]
        p=p+1

imgs_patches = imgs_patches[:,:,:,np.newaxis]
imgs_patches = imgs_patches.astype('float32')
imgs_patches /=255.0



predicted_test = model.predict(imgs_patches)

```

```
#join patches together
full_segs = np.ndarray((20, 640, 640), dtype=np.float32)
p=0
num_tests=5
for i in range(num_tests):
  for r in range(5):
    for c in range(5):
      full_segs[i, r*128:(r+1)*128, c*128:(c+1)*128]=predicted_test[p,:,:,0]
      p=p+1

#resize back to 565*584
full_segs_resized = np.ndarray((full_segs.shape[0], 584, 565), dtype=np.float32
n=0
for img in full_segs:
  res = cv2.resize(img, dsize=(565, 584), interpolation=cv2.INTER_CUBIC)
  full_segs_resized[n]=res
  n=n+1

#predicted seg -- rounded
thresh=0.5
full_segs_rounded = np.round(full_segs_resized>thresh)
w=Image.fromarray(255*full_segs_rounded[0].astype('uint8')).convert('L')
display(w)
```

```
w=Image.fromarray(test_imgs[0])
display(w)


```



```
w=Image.fromarray(test_segs[0]).convert('L')
display(w)

```

1