Generic-Semantic-Endoscopy-Segment-Unet++ (G)

March 23, 2023

1 Segment using Unet++

[113]:

```
#Libraries------------------
import os
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import gc
from pathlib import Path
from tqdm.notebook import trange, tqdm
from itertools import chain
from skimage.io import imread, imshow, concatenate_images
from skimage.transform import resize
from skimage.morphology import label
from sklearn.model_selection import train_test_split
import glob
import cv2
from PIL import Image
import glob2
from tensorflow.keras.models import load_model
import tensorflow
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator,␣
↪array_to_img, img_to_array, load_img
from tensorflow.keras.layers import Conv2D, Input, MaxPooling2D, Dropout,␣
↪concatenate, UpSampling2D
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
↪ReduceLROnPlateau, TensorBoard
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (BatchNormalization, Conv2DTranspose,
                                     SeparableConv2D, MaxPooling2D, Activation,␣
↪Flatten, Dropout, Dense)
```

```
from tensorflow.keras.preprocessing.image import load_img, array_to_img,␣
 ↪img_to_array
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, LeakyReLU, BatchNormalization,␣
 ↪MaxPool2D,Conv2DTranspose, concatenate,Input
from tensorflow.keras.callbacks import CSVLogger
from tensorflow.keras.utils import plot_model
import pickle
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import sklearn
from sklearn.cluster import KMeans
from tensorflow.keras.layers import *
from tensorflow.keras import models
from tensorflow.keras.callbacks import *
from tensorflow.keras.applications import ResNet50
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from tensorflow.keras.metrics import MeanIoU

K.clear_session()
warnings.filterwarnings('ignore')
plt.style.use("ggplot")
get_ipython().run_line_magic('matplotlib', 'inline')
```

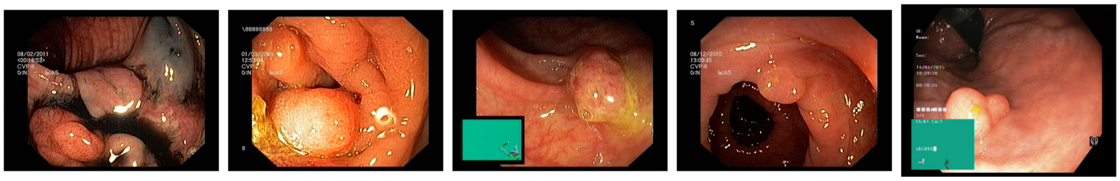
2 Reding and Preprocessing images

[114]:
```
#Load image data-------------------
H,W,CH=[128,128,3]
def cv_load_img(path):
    img= cv2.imread(path)
    img= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img=cv2.resize(img,(W,H))
    return img
```

[115]:
```
#Load data---------------------
BASE_DIR="Endoscopy/train/"
img_path= os.listdir(BASE_DIR+'images')
mask_path= os.listdir(BASE_DIR+'masks')
```

[116]:
```
#plot sample images--------------
fig, ax= plt.subplots(1,5, figsize=(20, 10))
for i in range(5):
    path= BASE_DIR + 'images/'
    ax[i].imshow(load_img(path + img_path[i]))
    ax[i].set_xticks([]); ax[i].set_yticks([])
```

```
fig.tight_layout()
plt.show()
```



[117]:

```
#plot sample masks---------------
fig, ax= plt.subplots(1,5, figsize=(20, 10))
for i in range(5):
    path= BASE_DIR + 'masks/'
    ax[i].imshow(cv_load_img(path + mask_path[i])[:, :, 0], 'gray')
    ax[i].set_xticks([]); ax[i].set_yticks([])

fig.tight_layout()
plt.show()
```

[118]:

```
#plot sample images--with blended mask ------------
fig, ax= plt.subplots(1,5, figsize=(20, 10))
for i in range(5):
    path1= BASE_DIR + 'images/'
    ax[i].imshow((cv_load_img(path1 + img_path[i])/255) * (cv_load_img(path + mask_path[i])/255))
    ax[i].set_xticks([]); ax[i].set_yticks([])

fig.tight_layout()
plt.show()
```

3 Augmented Images and Masks

[119]:
```
#data  preparation
X_train, X_test, y_train, y_test = train_test_split(img_path, mask_path, test_size=0.2, random_state=22)
len(X_train), len(X_test)
```

[119]:
```
(160, 40)
```

[120]:
```
#batch generation-----------------------
def load_data(path_list, gray=False):
    data=[]
    for path in tqdm(path_list):
        img= cv_load_img(path)
        if gray:
            img= img[:, :, 0:1]
        img= cv2.resize(img, (W, H))
        data.append(img)
    return np.array(data)
```

[121]:
```
#train data generation---------------------
X_train= load_data([BASE_DIR + 'images/' + x for x in X_train])/255.0
X_test= load_data([BASE_DIR + 'images/' + x for x in X_test])/255.0
```

```
0%|          | 0/160 [00:00<?, ?it/s]

0%|          | 0/40 [00:00<?, ?it/s]
```

[122]:
```
##test data generation---------------------
Y_train= load_data([BASE_DIR + 'masks/' + x for x in y_train], gray=True)/255.0
Y_test= load_data([BASE_DIR + 'masks/' + x for x in y_test], gray=True)/255.0
Y_train= Y_train.reshape(-1, W, H, 1)
Y_test= Y_test.reshape(-1, W, H, 1)

Y_train.shape, Y_test.shape
```

```
0%|          | 0/160 [00:00<?, ?it/s]
```

```
  0%|          | 0/40 [00:00<?, ?it/s]
```

[122]:
```
((160, 128, 128, 1), (40, 128, 128, 1))
```

Unet Model

[123]:
```python
def Conv2dBlock(inputTensor, numFilters, kernelSize = 3, doBatchNorm = True):
    #first Conv
    x = tf.keras.layers.Conv2D(filters = numFilters, kernel_size = (kernelSize,␣
 ↪kernelSize),
                              kernel_initializer = 'he_normal', padding =␣
 ↪'same') (inputTensor)

    if doBatchNorm:
        x = tf.keras.layers.BatchNormalization()(x)

    x =tf.keras.layers.Activation('relu')(x)

    #Second Conv
    x = tf.keras.layers.Conv2D(filters = numFilters, kernel_size = (kernelSize,␣
 ↪kernelSize),
                              kernel_initializer = 'he_normal', padding =␣
 ↪'same') (x)
    if doBatchNorm:
        x = tf.keras.layers.BatchNormalization()(x)

    x = tf.keras.layers.Activation('relu')(x)

    return x


# Now defining Unet
def GiveMeUnet(inputImage, numFilters = 16, droupouts = 0.1, doBatchNorm =␣
 ↪True):
    # defining encoder Path
    c1 = Conv2dBlock(inputImage, numFilters * 1, kernelSize = 3, doBatchNorm =␣
 ↪doBatchNorm)
    p1 = tf.keras.layers.MaxPooling2D((2,2))(c1)
    p1 = tf.keras.layers.Dropout(droupouts)(p1)

    c2 = Conv2dBlock(p1, numFilters * 2, kernelSize = 3, doBatchNorm =␣
 ↪doBatchNorm)
    p2 = tf.keras.layers.MaxPooling2D((2,2))(c2)
    p2 = tf.keras.layers.Dropout(droupouts)(p2)

    c3 = Conv2dBlock(p2, numFilters * 4, kernelSize = 3, doBatchNorm =␣
 ↪doBatchNorm)
```

```
    p3 = tf.keras.layers.MaxPooling2D((2,2))(c3)
    p3 = tf.keras.layers.Dropout(droupouts)(p3)

    c4 = Conv2dBlock(p3, numFilters * 8, kernelSize = 3, doBatchNorm = doBatchNorm)
    p4 = tf.keras.layers.MaxPooling2D((2,2))(c4)
    p4 = tf.keras.layers.Dropout(droupouts)(p4)

    c5 = Conv2dBlock(p4, numFilters * 16, kernelSize = 3, doBatchNorm = doBatchNorm)

    # defining decoder path
    u6 = tf.keras.layers.Conv2DTranspose(numFilters*8, (3, 3), strides = (2, 2), padding = 'same')(c5)
    u6 = tf.keras.layers.concatenate([u6, c4])
    u6 = tf.keras.layers.Dropout(droupouts)(u6)
    c6 = Conv2dBlock(u6, numFilters * 8, kernelSize = 3, doBatchNorm = doBatchNorm)

    u7 = tf.keras.layers.Conv2DTranspose(numFilters*4, (3, 3), strides = (2, 2), padding = 'same')(c6)

    u7 = tf.keras.layers.concatenate([u7, c3])
    u7 = tf.keras.layers.Dropout(droupouts)(u7)
    c7 = Conv2dBlock(u7, numFilters * 4, kernelSize = 3, doBatchNorm = doBatchNorm)

    u8 = tf.keras.layers.Conv2DTranspose(numFilters*2, (3, 3), strides = (2, 2), padding = 'same')(c7)
    u8 = tf.keras.layers.concatenate([u8, c2])
    u8 = tf.keras.layers.Dropout(droupouts)(u8)
    c8 = Conv2dBlock(u8, numFilters * 2, kernelSize = 3, doBatchNorm = doBatchNorm)

    u9 = tf.keras.layers.Conv2DTranspose(numFilters*1, (3, 3), strides = (2, 2), padding = 'same')(c8)
    u9 = tf.keras.layers.concatenate([u9, c1])
    u9 = tf.keras.layers.Dropout(droupouts)(u9)
    c9 = Conv2dBlock(u9, numFilters * 1, kernelSize = 3, doBatchNorm = doBatchNorm)

    output = tf.keras.layers.Conv2D(1, (1, 1), activation = 'sigmoid')(c9)
    model = tf.keras.Model(inputs = [inputImage], outputs = [output])

    return model
```

[124]:
```
def dice_loss(y_true, y_pred):
    numerator = tf.reduce_sum(y_true * y_pred)
    denominator = tf.reduce_sum(y_true * y_true) + tf.reduce_sum(y_pred *␣
 ↪y_pred) - tf.reduce_sum(y_true * y_pred)

    return 1 - numerator / denominator
```

[125]:
```
smooth =100
def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac
```

[126]:
```
def jacard_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) -␣
 ↪intersection + 1.0)
```

[127]:
```
metrics=['accuracy', jacard_coef, iou]

inputs = tf.keras.layers.Input((H, W, CH))
model = GiveMeUnet(inputs, droupouts= 0.07)
model.compile(optimizer = 'Adam', loss =  dice_loss, metrics =metrics)#loss =␣
 ↪binary_crossentropy "binary_accuracy",
```

4 Model Summary

[128]:
```
#Summary of model------------------
model.summary()



#Plot of model------------------
dot_img_file = 'model.png'
plot_model(model, to_file=dot_img_file, show_shapes=True)
```

```
Model: "model"
__________________________________________________________________________________
________________
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
 input_1 (InputLayer)            [(None, 128, 128, 3  0           []
                                 )]
```

```
 conv2d (Conv2D)                (None, 128, 128, 16  448
['input_1[0][0]']
                                )

 batch_normalization (BatchNorm  (None, 128, 128, 16  64
['conv2d[0][0]']
 alization)                     )

 activation (Activation)        (None, 128, 128, 16  0
['batch_normalization[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 128, 128, 16  2320
['activation[0][0]']
                                )

 batch_normalization_1 (BatchNo  (None, 128, 128, 16  64
['conv2d_1[0][0]']
 rmalization)                   )

 activation_1 (Activation)      (None, 128, 128, 16  0
['batch_normalization_1[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 64, 64, 16)   0
['activation_1[0][0]']

 dropout (Dropout)              (None, 64, 64, 16)   0
['max_pooling2d[0][0]']

 conv2d_2 (Conv2D)              (None, 64, 64, 32)   4640
['dropout[0][0]']

 batch_normalization_2 (BatchNo  (None, 64, 64, 32)  128
['conv2d_2[0][0]']
 rmalization)

 activation_2 (Activation)      (None, 64, 64, 32)   0
['batch_normalization_2[0][0]']

 conv2d_3 (Conv2D)              (None, 64, 64, 32)   9248
['activation_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 64, 64, 32)  128
['conv2d_3[0][0]']
 rmalization)
```

```
 activation_3 (Activation)      (None, 64, 64, 32)   0
['batch_normalization_3[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 32, 32, 32)  0
['activation_3[0][0]']

 dropout_1 (Dropout)            (None, 32, 32, 32)   0
['max_pooling2d_1[0][0]']

 conv2d_4 (Conv2D)              (None, 32, 32, 64)   18496
['dropout_1[0][0]']

 batch_normalization_4 (BatchNo  (None, 32, 32, 64)  256
['conv2d_4[0][0]']
 rmalization)

 activation_4 (Activation)      (None, 32, 32, 64)   0
['batch_normalization_4[0][0]']

 conv2d_5 (Conv2D)              (None, 32, 32, 64)   36928
['activation_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 32, 32, 64)  256
['conv2d_5[0][0]']
 rmalization)

 activation_5 (Activation)      (None, 32, 32, 64)   0
['batch_normalization_5[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 16, 16, 64)  0
['activation_5[0][0]']

 dropout_2 (Dropout)            (None, 16, 16, 64)   0
['max_pooling2d_2[0][0]']

 conv2d_6 (Conv2D)              (None, 16, 16, 128)  73856
['dropout_2[0][0]']

 batch_normalization_6 (BatchNo  (None, 16, 16, 128)  512
['conv2d_6[0][0]']
 rmalization)

 activation_6 (Activation)      (None, 16, 16, 128)  0
['batch_normalization_6[0][0]']

 conv2d_7 (Conv2D)              (None, 16, 16, 128)  147584
['activation_6[0][0]']
```

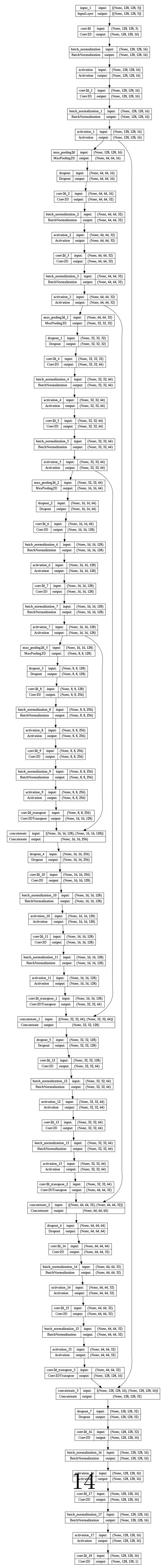
```
 batch_normalization_7 (BatchNo  (None, 16, 16, 128)  512
['conv2d_7[0][0]']
 rmalization)

 activation_7 (Activation)     (None, 16, 16, 128)  0
['batch_normalization_7[0][0]']

 max_pooling2d_3 (MaxPooling2D)  (None, 8, 8, 128)   0
['activation_7[0][0]']

 dropout_3 (Dropout)           (None, 8, 8, 128)    0
['max_pooling2d_3[0][0]']

 conv2d_8 (Conv2D)             (None, 8, 8, 256)    295168
['dropout_3[0][0]']

 batch_normalization_8 (BatchNo  (None, 8, 8, 256)   1024
['conv2d_8[0][0]']
 rmalization)

 activation_8 (Activation)     (None, 8, 8, 256)    0
['batch_normalization_8[0][0]']

 conv2d_9 (Conv2D)             (None, 8, 8, 256)    590080
['activation_8[0][0]']

 batch_normalization_9 (BatchNo  (None, 8, 8, 256)   1024
['conv2d_9[0][0]']
 rmalization)

 activation_9 (Activation)     (None, 8, 8, 256)    0
['batch_normalization_9[0][0]']

 conv2d_transpose (Conv2DTransp  (None, 16, 16, 128)  295040
['activation_9[0][0]']
 ose)

 concatenate (Concatenate)     (None, 16, 16, 256)  0
['conv2d_transpose[0][0]',
'activation_7[0][0]']

 dropout_4 (Dropout)           (None, 16, 16, 256)  0
['concatenate[0][0]']

 conv2d_10 (Conv2D)            (None, 16, 16, 128)  295040
['dropout_4[0][0]']

 batch_normalization_10 (BatchN  (None, 16, 16, 128)  512
```

```
['conv2d_10[0][0]']
 ormalization)

 activation_10 (Activation)     (None, 16, 16, 128)  0
['batch_normalization_10[0][0]']

 conv2d_11 (Conv2D)             (None, 16, 16, 128)  147584
['activation_10[0][0]']

 batch_normalization_11 (BatchN  (None, 16, 16, 128)  512
['conv2d_11[0][0]']
 ormalization)

 activation_11 (Activation)     (None, 16, 16, 128)  0
['batch_normalization_11[0][0]']

 conv2d_transpose_1 (Conv2DTran  (None, 32, 32, 64)  73792
['activation_11[0][0]']
 spose)

 concatenate_1 (Concatenate)    (None, 32, 32, 128)  0
['conv2d_transpose_1[0][0]',
'activation_5[0][0]']

 dropout_5 (Dropout)            (None, 32, 32, 128)  0
['concatenate_1[0][0]']

 conv2d_12 (Conv2D)             (None, 32, 32, 64)   73792
['dropout_5[0][0]']

 batch_normalization_12 (BatchN  (None, 32, 32, 64)  256
['conv2d_12[0][0]']
 ormalization)

 activation_12 (Activation)     (None, 32, 32, 64)   0
['batch_normalization_12[0][0]']

 conv2d_13 (Conv2D)             (None, 32, 32, 64)   36928
['activation_12[0][0]']

 batch_normalization_13 (BatchN  (None, 32, 32, 64)  256
['conv2d_13[0][0]']
 ormalization)

 activation_13 (Activation)     (None, 32, 32, 64)   0
['batch_normalization_13[0][0]']

 conv2d_transpose_2 (Conv2DTran  (None, 64, 64, 32)  18464
```

```
['activation_13[0][0]']
 spose)

 concatenate_2 (Concatenate)    (None, 64, 64, 64)   0
['conv2d_transpose_2[0][0]',
'activation_3[0][0]']

 dropout_6 (Dropout)            (None, 64, 64, 64)   0
['concatenate_2[0][0]']

 conv2d_14 (Conv2D)             (None, 64, 64, 32)   18464
['dropout_6[0][0]']

 batch_normalization_14 (BatchN  (None, 64, 64, 32)  128
['conv2d_14[0][0]']
 ormalization)

 activation_14 (Activation)     (None, 64, 64, 32)   0
['batch_normalization_14[0][0]']

 conv2d_15 (Conv2D)             (None, 64, 64, 32)   9248
['activation_14[0][0]']

 batch_normalization_15 (BatchN  (None, 64, 64, 32)  128
['conv2d_15[0][0]']
 ormalization)

 activation_15 (Activation)     (None, 64, 64, 32)   0
['batch_normalization_15[0][0]']

 conv2d_transpose_3 (Conv2DTran  (None, 128, 128, 16  4624
['activation_15[0][0]']
 spose)                         )

 concatenate_3 (Concatenate)    (None, 128, 128, 32  0
['conv2d_transpose_3[0][0]',
                                )
'activation_1[0][0]']

 dropout_7 (Dropout)            (None, 128, 128, 32  0
['concatenate_3[0][0]']
                                )

 conv2d_16 (Conv2D)             (None, 128, 128, 16  4624
['dropout_7[0][0]']
                                )

 batch_normalization_16 (BatchN  (None, 128, 128, 16  64
```

```
['conv2d_16[0][0]']
 ormalization)                  )

 activation_16 (Activation)     (None, 128, 128, 16  0
['batch_normalization_16[0][0]']
                                )

 conv2d_17 (Conv2D)             (None, 128, 128, 16  2320
['activation_16[0][0]']
                                )

 batch_normalization_17 (BatchN  (None, 128, 128, 16  64
['conv2d_17[0][0]']
 ormalization)                  )

 activation_17 (Activation)     (None, 128, 128, 16  0
['batch_normalization_17[0][0]']
                                )

 conv2d_18 (Conv2D)             (None, 128, 128, 1)  17
['activation_17[0][0]']

================================================================================
==================
Total params: 2,164,593
Trainable params: 2,161,649
Non-trainable params: 2,944
--------------------------------------------------------------------------------
------------------
```

[128]:



[129]:

```
#Train model---------------------------
nbatch_size=128
nepochs=50
history = model.fit(X_train,Y_train,batch_size=nbatch_size,
                    epochs=nepochs,validation_split=0.2,shuffle=True,
                    max_queue_size=32,workers=4,use_multiprocessing=True,
                   )
```

```
Epoch 1/50
1/1 [==============================] - 26s 26s/step - loss: 0.7980 - accuracy:
0.3636 - jacard_coef: 0.1262 - iou: 0.1263 - val_loss: 0.7594 - val_accuracy:
0.6439 - val_jacard_coef: 0.1352 - val_iou: 0.1355
Epoch 2/50
1/1 [==============================] - 15s 15s/step - loss: 0.7848 - accuracy:
0.4222 - jacard_coef: 0.1342 - iou: 0.1343 - val_loss: 0.7591 - val_accuracy:
0.4960 - val_jacard_coef: 0.1364 - val_iou: 0.1366
Epoch 3/50
1/1 [==============================] - 18s 18s/step - loss: 0.7769 - accuracy:
0.4522 - jacard_coef: 0.1404 - iou: 0.1404 - val_loss: 0.7585 - val_accuracy:
0.3660 - val_jacard_coef: 0.1389 - val_iou: 0.1391
Epoch 4/50
1/1 [==============================] - 17s 17s/step - loss: 0.7720 - accuracy:
0.4694 - jacard_coef: 0.1449 - iou: 0.1449 - val_loss: 0.7581 - val_accuracy:
0.2809 - val_jacard_coef: 0.1436 - val_iou: 0.1438
Epoch 5/50
1/1 [==============================] - 18s 18s/step - loss: 0.7680 - accuracy:
0.4811 - jacard_coef: 0.1485 - iou: 0.1485 - val_loss: 0.7613 - val_accuracy:
0.2294 - val_jacard_coef: 0.1534 - val_iou: 0.1536
Epoch 6/50
1/1 [==============================] - 19s 19s/step - loss: 0.7636 - accuracy:
0.4910 - jacard_coef: 0.1517 - iou: 0.1518 - val_loss: 0.7728 - val_accuracy:
0.2006 - val_jacard_coef: 0.1665 - val_iou: 0.1667
Epoch 7/50
1/1 [==============================] - 19s 19s/step - loss: 0.7579 - accuracy:
0.4998 - jacard_coef: 0.1554 - iou: 0.1555 - val_loss: 0.7711 - val_accuracy:
0.2043 - val_jacard_coef: 0.1722 - val_iou: 0.1724
Epoch 8/50
1/1 [==============================] - 18s 18s/step - loss: 0.7500 - accuracy:
0.5069 - jacard_coef: 0.1606 - iou: 0.1607 - val_loss: 0.7724 - val_accuracy:
0.2250 - val_jacard_coef: 0.1770 - val_iou: 0.1772
Epoch 9/50
1/1 [==============================] - 19s 19s/step - loss: 0.7416 - accuracy:
0.5175 - jacard_coef: 0.1658 - iou: 0.1659 - val_loss: 0.8180 - val_accuracy:
0.1734 - val_jacard_coef: 0.1709 - val_iou: 0.1710
Epoch 10/50
```

```
1/1 [==============================] - 20s 20s/step - loss: 0.7304 - accuracy:
0.5405 - jacard_coef: 0.1709 - iou: 0.1709 - val_loss: 0.8182 - val_accuracy:
0.1842 - val_jacard_coef: 0.1711 - val_iou: 0.1713
Epoch 11/50
1/1 [==============================] - 21s 21s/step - loss: 0.7164 - accuracy:
0.5600 - jacard_coef: 0.1781 - iou: 0.1781 - val_loss: 0.7700 - val_accuracy:
0.3844 - val_jacard_coef: 0.1865 - val_iou: 0.1867
Epoch 12/50
1/1 [==============================] - 20s 20s/step - loss: 0.7094 - accuracy:
0.5704 - jacard_coef: 0.1817 - iou: 0.1818 - val_loss: 0.8158 - val_accuracy:
0.2312 - val_jacard_coef: 0.1727 - val_iou: 0.1729
Epoch 13/50
1/1 [==============================] - 22s 22s/step - loss: 0.6966 - accuracy:
0.6136 - jacard_coef: 0.1862 - iou: 0.1863 - val_loss: 0.8188 - val_accuracy:
0.2387 - val_jacard_coef: 0.1712 - val_iou: 0.1714
Epoch 14/50
1/1 [==============================] - 20s 20s/step - loss: 0.6874 - accuracy:
0.6604 - jacard_coef: 0.1895 - iou: 0.1896 - val_loss: 0.7859 - val_accuracy:
0.4146 - val_jacard_coef: 0.1862 - val_iou: 0.1863
Epoch 15/50
1/1 [==============================] - 30s 30s/step - loss: 0.6763 - accuracy:
0.7084 - jacard_coef: 0.1950 - iou: 0.1951 - val_loss: 0.7849 - val_accuracy:
0.4417 - val_jacard_coef: 0.1862 - val_iou: 0.1864
Epoch 16/50
1/1 [==============================] - 25s 25s/step - loss: 0.6652 - accuracy:
0.7590 - jacard_coef: 0.1990 - iou: 0.1991 - val_loss: 0.8178 - val_accuracy:
0.2972 - val_jacard_coef: 0.1704 - val_iou: 0.1706
Epoch 17/50
1/1 [==============================] - 30s 30s/step - loss: 0.6585 - accuracy:
0.7967 - jacard_coef: 0.1991 - iou: 0.1992 - val_loss: 0.8002 - val_accuracy:
0.3925 - val_jacard_coef: 0.1807 - val_iou: 0.1809
Epoch 18/50
1/1 [==============================] - 24s 24s/step - loss: 0.6449 - accuracy:
0.8233 - jacard_coef: 0.2048 - iou: 0.2049 - val_loss: 0.7715 - val_accuracy:
0.5998 - val_jacard_coef: 0.1818 - val_iou: 0.1821
Epoch 19/50
1/1 [==============================] - 24s 24s/step - loss: 0.6421 - accuracy:
0.8308 - jacard_coef: 0.2051 - iou: 0.2052 - val_loss: 0.8130 - val_accuracy:
0.3362 - val_jacard_coef: 0.1742 - val_iou: 0.1744
Epoch 20/50
1/1 [==============================] - 24s 24s/step - loss: 0.6511 - accuracy:
0.8432 - jacard_coef: 0.1970 - iou: 0.1970 - val_loss: 0.8163 - val_accuracy:
0.3195 - val_jacard_coef: 0.1725 - val_iou: 0.1727
Epoch 21/50
1/1 [==============================] - 25s 25s/step - loss: 0.6191 - accuracy:
0.8569 - jacard_coef: 0.2157 - iou: 0.2157 - val_loss: 0.7928 - val_accuracy:
0.4476 - val_jacard_coef: 0.1849 - val_iou: 0.1851
Epoch 22/50
```

```
1/1 [==============================] - 29s 29s/step - loss: 0.6146 - accuracy:
0.8634 - jacard_coef: 0.2167 - iou: 0.2168 - val_loss: 0.7781 - val_accuracy:
0.5032 - val_jacard_coef: 0.1943 - val_iou: 0.1945
Epoch 23/50
1/1 [==============================] - 25s 25s/step - loss: 0.6038 - accuracy:
0.8703 - jacard_coef: 0.2221 - iou: 0.2222 - val_loss: 0.7959 - val_accuracy:
0.4313 - val_jacard_coef: 0.1851 - val_iou: 0.1853
Epoch 24/50
1/1 [==============================] - 27s 27s/step - loss: 0.5861 - accuracy:
0.8798 - jacard_coef: 0.2306 - iou: 0.2307 - val_loss: 0.8017 - val_accuracy:
0.4210 - val_jacard_coef: 0.1810 - val_iou: 0.1812
Epoch 25/50
1/1 [==============================] - 22s 22s/step - loss: 0.5827 - accuracy:
0.8845 - jacard_coef: 0.2305 - iou: 0.2306 - val_loss: 0.7869 - val_accuracy:
0.4875 - val_jacard_coef: 0.1893 - val_iou: 0.1895
Epoch 26/50
1/1 [==============================] - 21s 21s/step - loss: 0.5643 - accuracy:
0.8955 - jacard_coef: 0.2387 - iou: 0.2388 - val_loss: 0.7569 - val_accuracy:
0.5911 - val_jacard_coef: 0.2020 - val_iou: 0.2022
Epoch 27/50
1/1 [==============================] - 25s 25s/step - loss: 0.5567 - accuracy:
0.9014 - jacard_coef: 0.2415 - iou: 0.2416 - val_loss: 0.7786 - val_accuracy:
0.4989 - val_jacard_coef: 0.1917 - val_iou: 0.1919
Epoch 28/50
1/1 [==============================] - 21s 21s/step - loss: 0.5368 - accuracy:
0.9137 - jacard_coef: 0.2494 - iou: 0.2495 - val_loss: 0.7860 - val_accuracy:
0.4736 - val_jacard_coef: 0.1862 - val_iou: 0.1864
Epoch 29/50
1/1 [==============================] - 32s 32s/step - loss: 0.5339 - accuracy:
0.9151 - jacard_coef: 0.2486 - iou: 0.2487 - val_loss: 0.7771 - val_accuracy:
0.5088 - val_jacard_coef: 0.1894 - val_iou: 0.1896
Epoch 30/50
1/1 [==============================] - 24s 24s/step - loss: 0.5127 - accuracy:
0.9215 - jacard_coef: 0.2604 - iou: 0.2605 - val_loss: 0.7694 - val_accuracy:
0.5437 - val_jacard_coef: 0.1915 - val_iou: 0.1917
Epoch 31/50
1/1 [==============================] - 24s 24s/step - loss: 0.5098 - accuracy:
0.9212 - jacard_coef: 0.2630 - iou: 0.2631 - val_loss: 0.7948 - val_accuracy:
0.4212 - val_jacard_coef: 0.1845 - val_iou: 0.1847
Epoch 32/50
1/1 [==============================] - 24s 24s/step - loss: 0.5003 - accuracy:
0.9228 - jacard_coef: 0.2675 - iou: 0.2676 - val_loss: 0.7795 - val_accuracy:
0.5098 - val_jacard_coef: 0.1882 - val_iou: 0.1884
Epoch 33/50
1/1 [==============================] - 23s 23s/step - loss: 0.4959 - accuracy:
0.9237 - jacard_coef: 0.2707 - iou: 0.2708 - val_loss: 0.7712 - val_accuracy:
0.5424 - val_jacard_coef: 0.1915 - val_iou: 0.1917
Epoch 34/50
```

```
1/1 [==============================] - 25s 25s/step - loss: 0.4799 - accuracy:
0.9283 - jacard_coef: 0.2782 - iou: 0.2783 - val_loss: 0.7606 - val_accuracy:
0.6007 - val_jacard_coef: 0.1912 - val_iou: 0.1915
Epoch 35/50
1/1 [==============================] - 22s 22s/step - loss: 0.4726 - accuracy:
0.9301 - jacard_coef: 0.2824 - iou: 0.2824 - val_loss: 0.7703 - val_accuracy:
0.6114 - val_jacard_coef: 0.1819 - val_iou: 0.1822
Epoch 36/50
1/1 [==============================] - 21s 21s/step - loss: 0.4546 - accuracy:
0.9350 - jacard_coef: 0.2922 - iou: 0.2922 - val_loss: 0.7630 - val_accuracy:
0.6559 - val_jacard_coef: 0.1795 - val_iou: 0.1798
Epoch 37/50
1/1 [==============================] - 20s 20s/step - loss: 0.4511 - accuracy:
0.9355 - jacard_coef: 0.2939 - iou: 0.2940 - val_loss: 0.7613 - val_accuracy:
0.6256 - val_jacard_coef: 0.1878 - val_iou: 0.1881
Epoch 38/50
1/1 [==============================] - 21s 21s/step - loss: 0.4289 - accuracy:
0.9460 - jacard_coef: 0.3027 - iou: 0.3028 - val_loss: 0.7535 - val_accuracy:
0.6529 - val_jacard_coef: 0.1887 - val_iou: 0.1890
Epoch 39/50
1/1 [==============================] - 19s 19s/step - loss: 0.4295 - accuracy:
0.9443 - jacard_coef: 0.3030 - iou: 0.3031 - val_loss: 0.7450 - val_accuracy:
0.6979 - val_jacard_coef: 0.1833 - val_iou: 0.1835
Epoch 40/50
1/1 [==============================] - 21s 21s/step - loss: 0.4225 - accuracy:
0.9457 - jacard_coef: 0.3069 - iou: 0.3070 - val_loss: 0.7492 - val_accuracy:
0.7042 - val_jacard_coef: 0.1762 - val_iou: 0.1765
Epoch 41/50
1/1 [==============================] - 19s 19s/step - loss: 0.4118 - accuracy:
0.9477 - jacard_coef: 0.3137 - iou: 0.3137 - val_loss: 0.7505 - val_accuracy:
0.7507 - val_jacard_coef: 0.1609 - val_iou: 0.1612
Epoch 42/50
1/1 [==============================] - 27s 27s/step - loss: 0.3999 - accuracy:
0.9504 - jacard_coef: 0.3204 - iou: 0.3205 - val_loss: 0.7453 - val_accuracy:
0.7427 - val_jacard_coef: 0.1670 - val_iou: 0.1673
Epoch 43/50
1/1 [==============================] - 27s 27s/step - loss: 0.3982 - accuracy:
0.9506 - jacard_coef: 0.3202 - iou: 0.3203 - val_loss: 0.7472 - val_accuracy:
0.7594 - val_jacard_coef: 0.1601 - val_iou: 0.1604
Epoch 44/50
1/1 [==============================] - 19s 19s/step - loss: 0.3940 - accuracy:
0.9514 - jacard_coef: 0.3231 - iou: 0.3232 - val_loss: 0.7672 - val_accuracy:
0.7944 - val_jacard_coef: 0.1331 - val_iou: 0.1335
Epoch 45/50
1/1 [==============================] - 21s 21s/step - loss: 0.3798 - accuracy:
0.9547 - jacard_coef: 0.3313 - iou: 0.3314 - val_loss: 0.7669 - val_accuracy:
0.7755 - val_jacard_coef: 0.1389 - val_iou: 0.1392
Epoch 46/50
```

```
1/1 [==============================] - 27s 27s/step - loss: 0.3760 - accuracy:
0.9551 - jacard_coef: 0.3336 - iou: 0.3337 - val_loss: 0.7606 - val_accuracy:
0.7402 - val_jacard_coef: 0.1534 - val_iou: 0.1537
Epoch 47/50
1/1 [==============================] - 23s 23s/step - loss: 0.3674 - accuracy:
0.9567 - jacard_coef: 0.3386 - iou: 0.3387 - val_loss: 0.7665 - val_accuracy:
0.6911 - val_jacard_coef: 0.1617 - val_iou: 0.1619
Epoch 48/50
1/1 [==============================] - 27s 27s/step - loss: 0.3683 - accuracy:
0.9561 - jacard_coef: 0.3374 - iou: 0.3375 - val_loss: 0.7598 - val_accuracy:
0.7328 - val_jacard_coef: 0.1547 - val_iou: 0.1550
Epoch 49/50
1/1 [==============================] - 23s 23s/step - loss: 0.3571 - accuracy:
0.9604 - jacard_coef: 0.3422 - iou: 0.3423 - val_loss: 0.7578 - val_accuracy:
0.6861 - val_jacard_coef: 0.1702 - val_iou: 0.1705
Epoch 50/50
1/1 [==============================] - 25s 25s/step - loss: 0.3545 - accuracy:
0.9610 - jacard_coef: 0.3433 - iou: 0.3434 - val_loss: 0.7600 - val_accuracy:
0.7028 - val_jacard_coef: 0.1629 - val_iou: 0.1632
```

[130]:

```
df_result = pd.DataFrame(history.history)
df_result
```

[130]:

	loss	accuracy	jacard_coef	iou	val_loss	val_accuracy	\
0	0.798049	0.363575	0.126215	0.126282	0.759393	0.643923	
1	0.784817	0.422194	0.134232	0.134299	0.759146	0.496050	
2	0.776891	0.452172	0.140356	0.140424	0.758508	0.365967	
3	0.772003	0.469357	0.144859	0.144927	0.758108	0.280945	
4	0.767987	0.481122	0.148453	0.148521	0.761281	0.229422	
5	0.763621	0.490982	0.151729	0.151797	0.772766	0.200575	
6	0.757877	0.499775	0.155413	0.155481	0.771101	0.204332	
7	0.749980	0.506860	0.160585	0.160654	0.772372	0.224953	
8	0.741581	0.517471	0.165845	0.165914	0.817990	0.173391	
9	0.730388	0.540536	0.170860	0.170930	0.818221	0.184151	
10	0.716402	0.559953	0.178053	0.178124	0.769989	0.384352	
11	0.709371	0.570383	0.181724	0.181796	0.815787	0.231226	
12	0.696607	0.613554	0.186219	0.186292	0.818781	0.238659	
13	0.687356	0.660358	0.189526	0.189601	0.785875	0.414583	
14	0.676288	0.708398	0.195002	0.195077	0.784929	0.441656	
15	0.665217	0.758972	0.199009	0.199085	0.817783	0.297159	
16	0.658541	0.796706	0.199133	0.199211	0.800168	0.392502	
17	0.644876	0.823269	0.204786	0.204865	0.771490	0.599764	
18	0.642066	0.830811	0.205101	0.205180	0.813003	0.336170	
19	0.651136	0.843201	0.196967	0.197048	0.816264	0.319475	
20	0.619052	0.856896	0.215666	0.215747	0.792800	0.447567	
21	0.614576	0.863359	0.216700	0.216782	0.778071	0.503187	
22	0.603832	0.870285	0.222094	0.222176	0.795903	0.431301	

23	0.586051	0.879779	0.230614	0.230697	0.801701	0.420975
24	0.582680	0.884513	0.230483	0.230566	0.786913	0.487499
25	0.564259	0.895481	0.238743	0.238827	0.756888	0.591133
26	0.556749	0.901408	0.241537	0.241622	0.778594	0.498911
27	0.536810	0.913702	0.249433	0.249519	0.786009	0.473572
28	0.533903	0.915131	0.248639	0.248725	0.777105	0.508793
29	0.512680	0.921525	0.260382	0.260469	0.769355	0.543713
30	0.509837	0.921229	0.263029	0.263115	0.794828	0.421244
31	0.500333	0.922839	0.267465	0.267552	0.779546	0.509798
32	0.495853	0.923718	0.270713	0.270801	0.771246	0.542383
33	0.479891	0.928258	0.278216	0.278303	0.760597	0.600679
34	0.472617	0.930125	0.282353	0.282441	0.770314	0.611385
35	0.454596	0.934973	0.292152	0.292240	0.762977	0.655884
36	0.451090	0.935520	0.293910	0.293999	0.761319	0.625641
37	0.428864	0.946005	0.302738	0.302827	0.753461	0.652918
38	0.429527	0.944317	0.302998	0.303087	0.744956	0.697876
39	0.422532	0.945704	0.306925	0.307014	0.749174	0.704166
40	0.411812	0.947737	0.313660	0.313749	0.750487	0.750658
41	0.399940	0.950417	0.320402	0.320491	0.745337	0.742662
42	0.398235	0.950592	0.320217	0.320307	0.747163	0.759428
43	0.394043	0.951363	0.323113	0.323203	0.767179	0.794399
44	0.379812	0.954715	0.331263	0.331352	0.766915	0.775478
45	0.376024	0.955114	0.333629	0.333719	0.760554	0.740202
46	0.367351	0.956743	0.338632	0.338722	0.766535	0.691097
47	0.368307	0.956122	0.337411	0.337501	0.759752	0.732786
48	0.357121	0.960428	0.342216	0.342307	0.757841	0.686104
49	0.354494	0.960978	0.343273	0.343364	0.760017	0.702797

	val_jacard_coef	val_iou
0	0.135164	0.135450
1	0.136357	0.136640
2	0.138855	0.139132
3	0.143561	0.143826
4	0.153368	0.153612
5	0.166526	0.166741
6	0.172206	0.172415
7	0.176953	0.177157
8	0.170855	0.171025
9	0.171080	0.171251
10	0.186546	0.186748
11	0.172737	0.172909
12	0.171220	0.171391
13	0.186156	0.186347
14	0.186184	0.186379
15	0.170415	0.170592
16	0.180678	0.180866
17	0.181815	0.182056

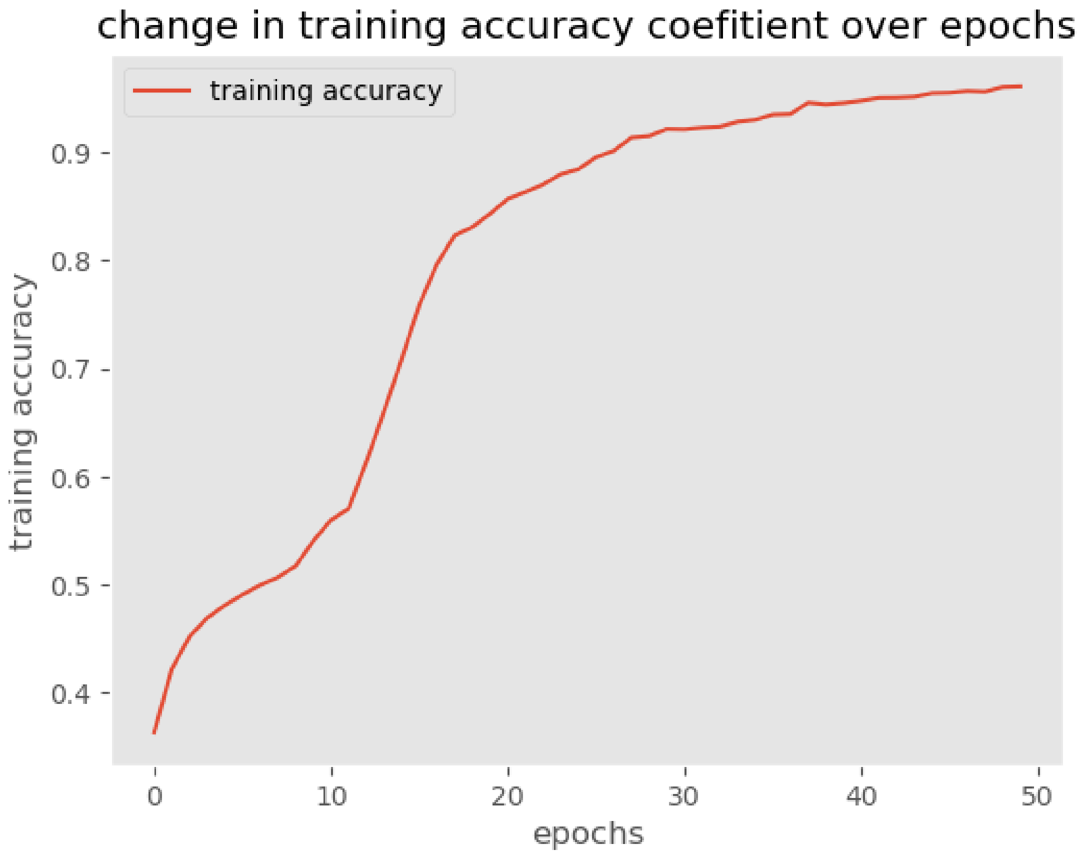
18	0.174179	0.174361
19	0.172511	0.172691
20	0.184946	0.185148
21	0.194263	0.194475
22	0.185112	0.185310
23	0.180996	0.181194
24	0.189296	0.189508
25	0.202010	0.202246
26	0.191659	0.191867
27	0.186152	0.186355
28	0.189400	0.189609
29	0.191491	0.191708
30	0.184480	0.184672
31	0.188201	0.188410
32	0.191509	0.191726
33	0.191219	0.191455
34	0.181937	0.182186
35	0.179537	0.179803
36	0.187849	0.188095
37	0.188725	0.188978
38	0.183262	0.183534
39	0.176230	0.176510
40	0.160926	0.161237
41	0.166955	0.167256
42	0.160102	0.160415
43	0.133100	0.133453
44	0.138855	0.139192
45	0.153357	0.153666
46	0.161652	0.161935
47	0.154700	0.155006
48	0.170206	0.170480
49	0.162942	0.163228

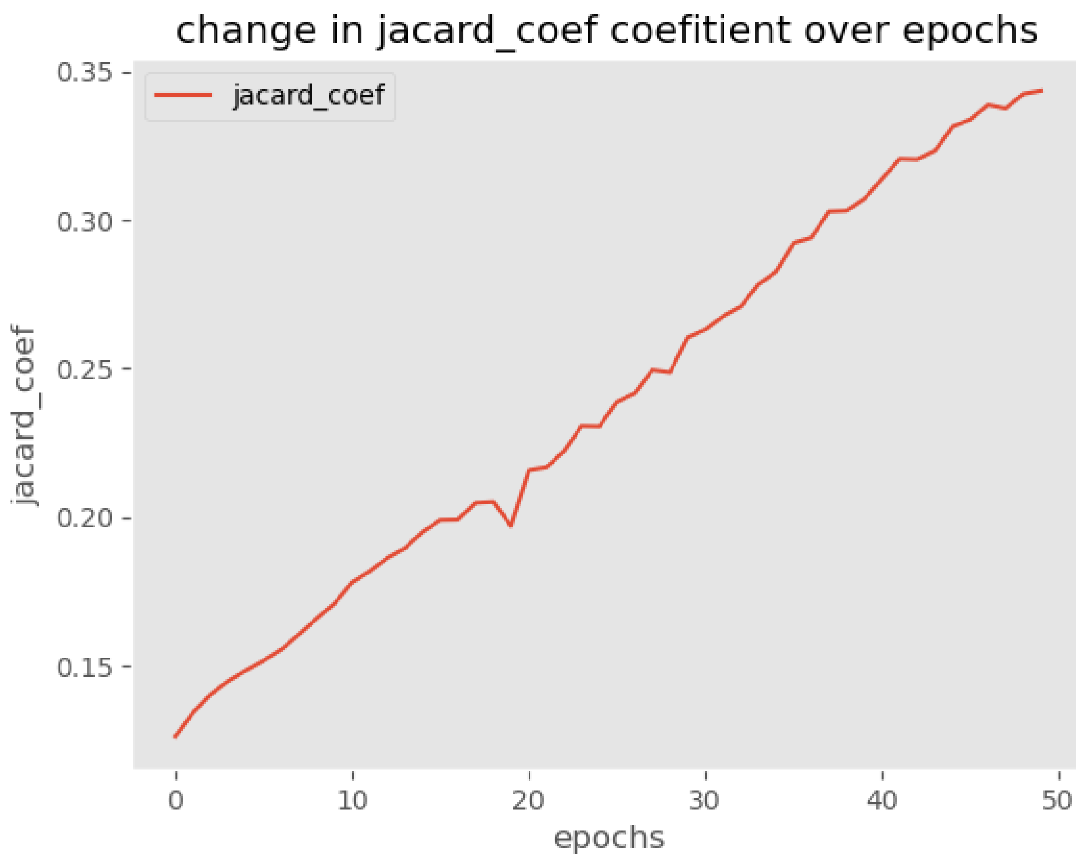
5 Visualize the model predictions

[131]:
```
# Plotting loss change over epochs---------------
nrange=nepochs
x = [i for i in range(nrange)]
plt.plot(x,history.history['loss'])
plt.title('change in loss over epochs')
plt.legend(['training_loss'])
plt.xlabel('epochs')
plt.ylabel('loss')
#plt.axis('off')
plt.grid(None)
plt.show()
```

```
plt.tight_layout()

# Plotting accuracy change over epochs---------------------
x = [i for i in range(nrange)]
plt.plot(x,history.history['accuracy'])
plt.title('change in training accuracy coefitient over epochs')
plt.legend(['training accuracy'])
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.grid(None)
plt.show()
plt.tight_layout()

# Plotting accuracy change over epochs---------------------
x = [i for i in range(nrange)]
plt.plot(x,history.history['jacard_coef'])
plt.title('change in jacard_coef coefitient over epochs')
plt.legend(['jacard_coef'])
plt.xlabel('epochs')
plt.ylabel('jacard_coef')
plt.grid(None)
plt.show()
plt.tight_layout()
```


change in loss over epochs
training_loss
0.8
0.7
0.6
0.5
0.4
loss
0
10
20
30
40
50
epochs


change in training accuracy coefitient over epochs
training accuracy
0.9
0.8
0.7
0.6
0.5
0.4
training accuracy
0
10
20
30
40
50
epochs



<Figure size 640x480 with 0 Axes>

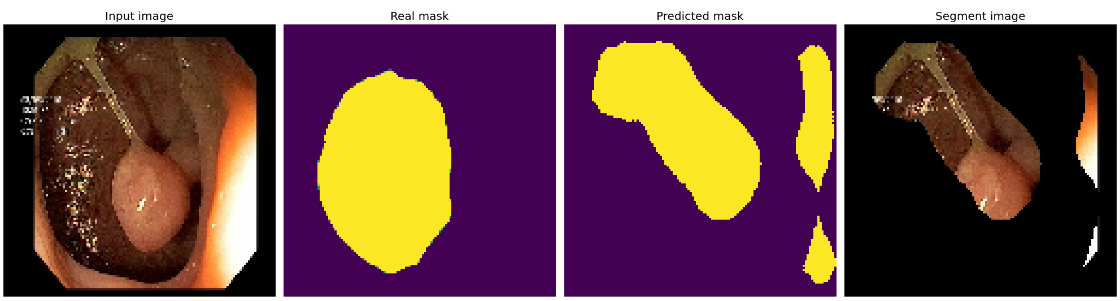
[132]:
```
# Creating predictions on our test set-----------------
predictions = model.predict(X_test)

# create predictes mask--------------

def create_mask(predictions,input_shape=(W,H,1)):
    mask = np.zeros(input_shape)
    mask[predictions>0.5] = 1
    return mask
```

2/2 [==============================] - 1s 148ms/step

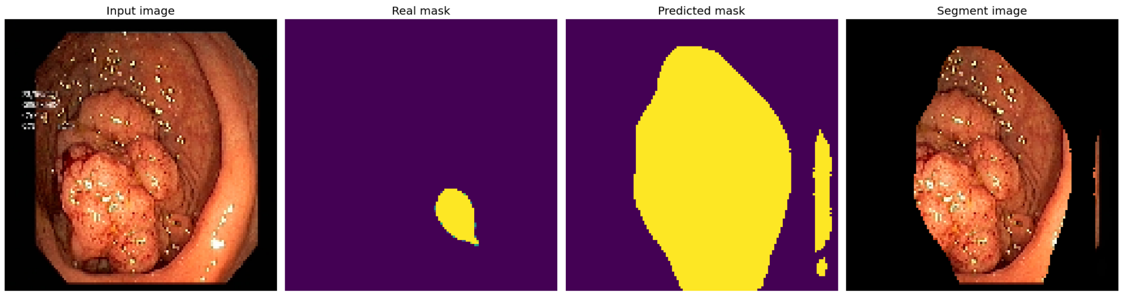
[133]:
```
# Ploting results for one image

def plot_results_for_one_sample(sample_index):

    mask = create_mask(predictions[sample_index])
    fig = plt.figure(figsize=(20,20))
```

```
    #image
    fig.add_subplot(1,4,1)
    plt.title('Input image')
    plt.imshow(X_test[sample_index])
    plt.axis('off')
    plt.grid(None)
    #mask
    fig.add_subplot(1,4,2)
    plt.title('Real mask')
    plt.imshow(Y_test[sample_index])
    plt.axis('off')
    plt.grid(None)
    #Predicted mask
    fig.add_subplot(1,4,3)
    plt.title('Predicted mask')
    plt.imshow(mask)
    plt.axis('off')
    plt.grid(None)
    #Segment
    fig.add_subplot(1,4,4)
    plt.title("Segment image")
    plt.imshow(X_test[sample_index]*mask)
    plt.grid(None)
    plt.axis('off')
    fig.tight_layout()
```

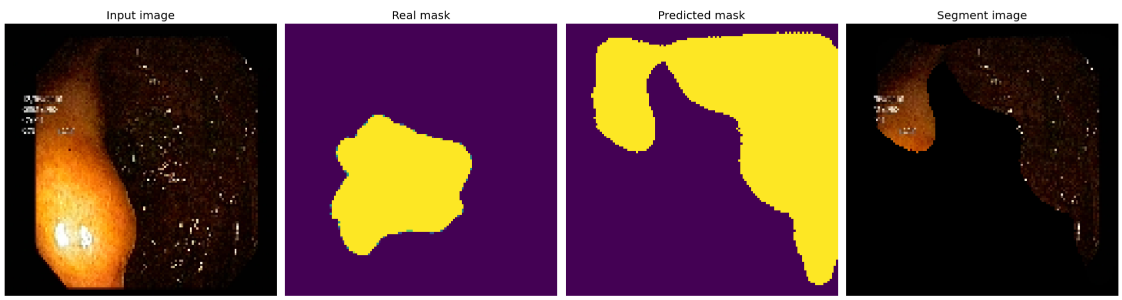
[134]:
```
#Show predicted result---------------
plot_results_for_one_sample(0)
```



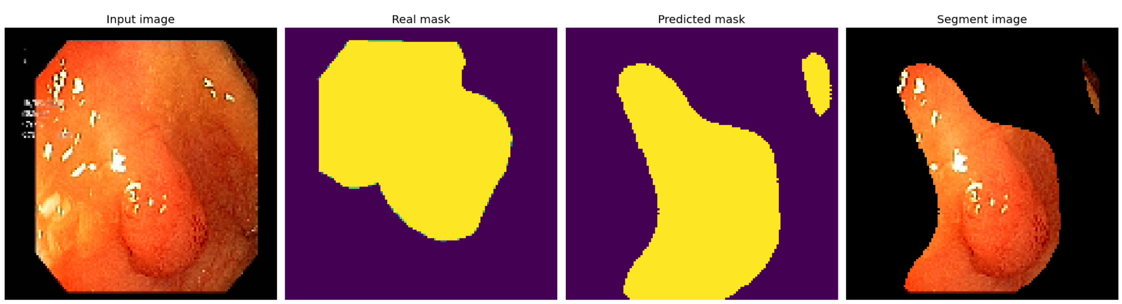
[135]:
```
#Show predicted result---------------
plot_results_for_one_sample(1)
```



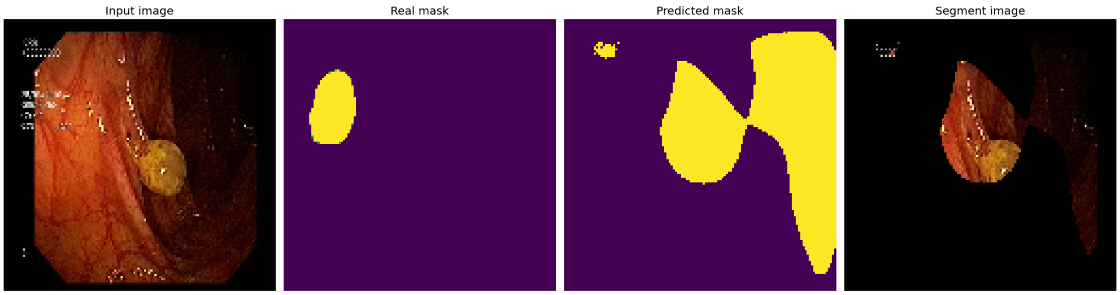
[136]:
```
#Show predicted result---------------
plot_results_for_one_sample(2)
```



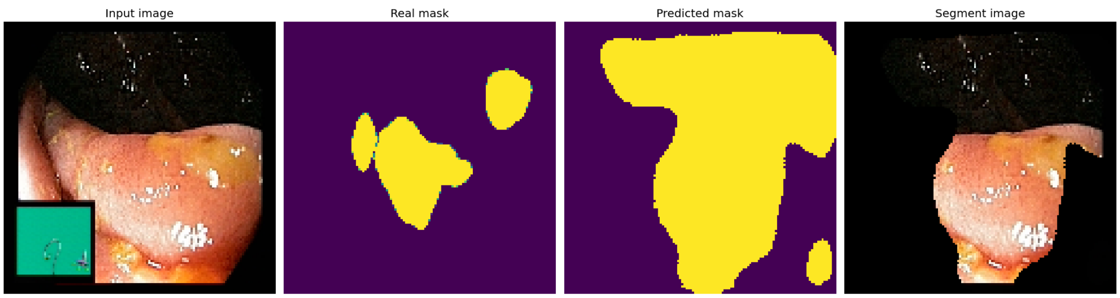
[137]:
```
#Show predicted result---------------
plot_results_for_one_sample(3)
```



[138]:
```
#Show predicted result---------------
plot_results_for_one_sample(4)
```




```
[139]: #Show predicted result---------------
       plot_results_for_one_sample(5)
```




```
[ ]:
```

```
[ ]:
```