```
import segmentation_models as sm #1.1
from tensorflow.keras.callbacks import EarlyStopping
import albumentations as A
import tensorflow as tf #2.2
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import model_from_json

sm.set_framework('tf.keras')



class segmentation:
  def __init__(self,PATH,CLASSES,target_size=(224,224), params=None):
    """
    parameters
    ----------
    PATH : string
      path to directory containing folders: 'images' and 'masks'
      optionally it will include 'val-images' and 'val-masks'
    CLASSES : list
      Names of classes
    target_size : tuple of ints
      (WxH) Images will be resized to this. Each dimension should be divisible by
      32 for best results.
    """
    self.PATH=PATH
    self.classes=CLASSES
    self.n_classes = len(CLASSES)
    self.activation = 'sigmoid' if self.n_classes == 1 else 'softmax'
    self.metrics=['accuracy',sm.metrics.IOUScore(threshold=0.5),sm.metrics.FScore(threshold=0.5)]
    self.target_size = target_size






    # check if validation directory exists
    self.has_validation = False
    if os.path.isdir(os.path.join(PATH, 'validation_image')) and os.path.isdir(os.path.join(PATH, 'validation_mask')):
      self.has_validation = True



    if params is None:
      #Questions :)
      self.backbone=self.get_good_answer("\n Which backbone model do you want to use? \n -'mobilenet' or 'mobilen
etv2': efficient and light for real-word application \n -'inceptionv3': Deep Convolutional Neural Network with sparse
ly connected architecture developped by Google (using different types of convolutional blocks at each layer) \n -'res
net18','resnet34','resnet50','resnet101' or'resnet152': core idea of this model is  'identity shortcut connection' that skip
s one or more layers \n We encourage you to try mobilenet first to see if it is sufficient for your segmentation task \n
",acceptable_answers=['mobilenet','mobilenetv2','inceptionv3','resnet18','resnet34','resnet50','resnet101','resnet152'])
```

```
      self.loss,self.weights= self.find_loss()
      self.augmentation= self.get_good_answer("Do you want Data Augmentation ? Yes or No \n",["yes","Yes","No",
"no"]).lower()
      answer_weights_pretrained = self.get_good_answer("Do you want to use pre-trained weights trained on Imagen
et for the encoder ? \n Yes or No \n",["yes","Yes","No","no"]).lower()
      self.weights_pretrained=self.convert_true_false(answer_weights_pretrained,'imagenet',None)

      #More Questions :)
      self.batch_size= int(input("\n What is your batch_size ? \n "))
      self.steps_per_epoch= int(input("\n What is your steps_per_epoch ? \n For guidance, you have %s training imag
es and a chosen batch_size of %s \n Normally (with many images), the steps_per_epoch is equal to Nbr_training_im
ages//batch_size==%s \n However, if you have a few images, you could increase that number because you'll have da
ta augmentation \n"%(len(os.listdir(PATH+'train_imgs/train')),self.batch_size,len(os.listdir(PATH+'train_imgs/train'
))//self.batch_size)))
      self.n_epochs= int(self.get_good_answer("\n How many epochs do you want to run ? \n ",acceptable_answers=
None))
      answer_encoder_freeze= input("\n Do you want to freeze the encoder layer ? Yes or No \n ")
      self.encoder_freeze=self.convert_true_false(answer_encoder_freeze,True,False)

    else :
      self.backbone=params['backbone']
      self.loss =params['loss']
      self.weights =params['weights']
      self.augmentation =params['augmentation']
      self.weights_pretrained =params['weights_pretrained']
      self.batch_size =params['batch_size']
      self.steps_per_epoch =params['steps_per_epoch']
      self.n_epochs =params['n_epochs']
      self.encoder_freeze =params['encoder_freeze']
      self.model_load_json=params['model_load_json']
      self.model_load_h5=params['model_load_h5']

    #Get everything Set Up
    #Data
    self.create_datagenerator(PATH)
    #Model

    self.model = model_from_json(open(self.model_load_json, "r").read())
    self.model.load_weights(self.model_load_h5)
    #self.model = sm.FPN(self.backbone, encoder_weights=self.weights_pretrained,classes=self.n_classes, activation
=self.activation, encoder_freeze=self.encoder_freeze)

    try:
      #Now Train
      self.train()
      #Next Step
      a=self.next_step()
      while a =='continue':
        a=self.next_step()
    except KeyboardInterrupt:
      # allow user to press crtl-C to end execution without losing model
      pass

  def convert_true_false(self,answer,true_answer=True,false_answer=False):
```

```
    if answer.lower()=='yes':
      return true_answer
    return false_answer

  def train(self):

    print("\n Starting Training \n")
    early_stopping=EarlyStopping(monitor='val_loss',patience=50,verbose=1,min_delta=0.001)

    self.model.compile('adam', self.loss, self.metrics, loss_weights=self.weights)
    if self.has_validation:
      self.model_history=self.model.fit(self.train_generator, epochs=self.n_epochs,
                    steps_per_epoch = self.steps_per_epoch,
                    validation_data=self.val_generator,
                    validation_steps=1)
    else:
      self.model_history=self.model.fit(self.train_generator, epochs=self.n_epochs,
                    steps_per_epoch = self.steps_per_epoch,callbacks=[early_stopping]
                    )
    print("\n Finished Training \n")

  def next_step(self):
    answ=self.get_good_answer("\n What do you want to do now ? \n -'save_model' \n -'plot_history' \n -'show_predi
ctions' \n -'continue_training' \n -'predict_on_new_data' \n -'end' \n",['save_model','plot_history','show_predictions','c
ontinue_training','predict_on_new_data','end'])

    if answ=='save_model':
      # location=self.get_good_answer("\n Type location to save to \n Example :'./models/unet_mobilenetv2' \n ",Non
e)
      model_json =  self.model.to_json()
      with open(self.backbone+"lossless.json", "w") as json_file:
        json_file.write(model_json)
        self.model.save_weights(self.backbone+".h5")
        print("Saved model to disk")
      # self.model.save_weights(location)

    if answ=='plot_history':
      self.plot_history()
    if answ =='show_predictions':
      dir=os.listdir(PATH+'predicted_mask/')
      if(len(dir)!=0):
        for file in dir:
          os.remove(PATH+'predicted_mask/'+file)
        self.show_predictions()
      else:
        self.show_predictions()
    if answ =='continue_training':
      self.change_parameters()
      self.train()
    if answ =='predict_on_new_data':
      self.predict_unlabeled()

    if answ=='end':
      return 'end'
```

```
  return 'continue'

def get_good_answer(self,prompt,acceptable_answers):
  while True:
      try:
          value = input(prompt)
      except ValueError:
          print("Sorry, I didn't understand that.")
          continue
      if acceptable_answers==None:
        break
      if value not in acceptable_answers:
          print("Sorry, your response must not be in %s"%acceptable_answers)
          continue
      else:
          break
  return value


def data_augment(self):
  options = {#'shear_range': 5.,
        #'zoom_range': 0.5,
        'horizontal_flip': True,
        'vertical_flip': True,
        'rotation_range' : 90,
         #'width_shift_range':0.0,
         #'height_shift_range':0.0,
       }
  return options

def create_augmentation_pipeline(self):
  augmentation_pipeline = A.Compose(
  [
      A.HorizontalFlip(p = 0.5), # apply horizontal flip to 50% of images
      A.OneOf(
          [
              # apply one of transforms to 50% of images
              A.RandomContrast(), # apply random contrast
              A.RandomGamma(), # apply random gamma
              A.RandomBrightness(), # apply random brightness
          ],
          p = 0.5
      ),
      A.OneOf(
          [
              # apply one of transforms to 50% images
              A.ElasticTransform(
                  alpha = 120,
                  sigma = 120 * 0.05,
                  alpha_affine = 120 * 0.03
              ),
              A.GridDistortion(),
              #A.OpticalDistortion(
              #    distort_limit = 2,
```

```
            #    shift_limit = 0.5
            #),
        ],
        p = 0.5
    )
  ],
  p = 0.9 #in 10% of cases keep same image because that's interesting also haha
  )
  return augmentation_pipeline

 def create_datagenerator(self,PATH,):
  options=self.data_augment()
  image_datagen = ImageDataGenerator(rescale=1./255,**options)
  mask_datagen = ImageDataGenerator(**options)
  val_datagen = ImageDataGenerator(rescale=1./255)
  val_datagen_mask = ImageDataGenerator(rescale=1)
  test_datagen = ImageDataGenerator(rescale=1./255)
  test_datagen_mask = ImageDataGenerator(rescale=1)
  #Create custom zip anc dustom batch_size

  def combine_generator(gen1, gen2,batch_size=6,training=True):
    while True:
      image_batch, label_batch=next(gen1)[0], np.expand_dims(next(gen2)[0][:,:,0],axis=-1)
      image_batch, label_batch=np.expand_dims(image_batch,axis=0),np.expand_dims(label_batch,axis=0)

      for i in range(batch_size-1):
        image_i,label_i = next(gen1)[0], np.expand_dims(next(gen2)[0][:,:,0],axis=-1)

        if self.augmentation == 'yes' and training==True :
          aug_pipeline=self.create_augmentation_pipeline()
          augmented = aug_pipeline(image = image_i, mask = label_i)
          image_i,label_i=augmented['image'],augmented['mask']

        image_i, label_i=np.expand_dims(image_i,axis=0),np.expand_dims(label_i,axis=0)
        image_batch=np.concatenate([image_batch,image_i],axis=0)
        label_batch=np.concatenate([label_batch,label_i],axis=0)

      yield((image_batch,label_batch))


  seed = np.random.randint(0,1e5)

  train_image_generator = image_datagen.flow_from_directory(PATH+'train_image',seed=seed, target_size=self.ta
rget_size,class_mode=None,batch_size = self.batch_size)
  train_mask_generator = mask_datagen.flow_from_directory(PATH+'train_mask',seed=seed, target_size=self.targ
et_size,class_mode=None,batch_size = self.batch_size)
  self.train_generator = combine_generator(train_image_generator, train_mask_generator,training=True)

  if self.has_validation:
    val_image_generator = val_datagen.flow_from_directory(PATH+'validation_image',seed=seed, target_size=self.
target_size,class_mode=None,batch_size = self.batch_size)
    val_mask_generator = val_datagen_mask.flow_from_directory(PATH+'validation_mask',seed=seed, target_size
=self.target_size,class_mode=None,batch_size = self.batch_size)
    self.val_generator = combine_generator(val_image_generator, val_mask_generator,training=False)
```

```
    test_image_generator = test_datagen.flow_from_directory(PATH+'test_x',seed=seed, target_size=self.target_siz
e,class_mode=None,batch_size=self.batch_size)
    test_mask_generator = test_datagen_mask.flow_from_directory(PATH+'test_y',seed=seed, target_size=self.targe
t_size,class_mode=None,batch_size=self.batch_size)
    self.test_generator = combine_generator(test_image_generator, test_mask_generator, training = False)


  def change_parameters(self):
   answer=self.get_good_answer("\n Do you want to change any parameters for the new training ? - 'No' \n - 'epochs
' \n - 'loss' \n -'batch_size' \n -'encode_freeze",['No','no','epochs','loss','batch_size','encode_freeze'])
   while answer !='No' and answer !='no':
    if answer == 'epochs':
     self.n_epochs= int(self.get_good_answer("\n How many epochs do you now want to run ? \n ",acceptable_ans
wers=None))
    if answer == 'batch_size':
     self.batch_size= int(input("\n What is your new batch_size ? \n "))
    if answer == 'encode_freeze':
     answer_encoder_freeze= input("\n Do you want to freeze the encoder layer ? Yes or No \n ")
     self.encoder_freeze=False
     if answer_encoder_freeze.lower()=='yes':
      self.encoder_freeze=True
    if answer == 'loss':
     self.loss,self.weights= self.find_loss()

    answer=self.get_good_answer("\n Do you want to change any other parameters for the new training ? - 'No' \n - '
epochs' \n - 'loss' \n -'batch_size' \n -'encode_freeze",['No','no','epochs','loss','batch_size','encode_freeze'])


  def plot_history(self):

   fig,ax=plt.subplots(1,3,figsize=(20,5))
   epochs = range(self.n_epochs)

   loss = self.model_history.history['loss']
   val_loss = self.model_history.history['val_loss']



   ax[0].plot(epochs, loss, 'r', label='Training loss')
   ax[0].plot(epochs, val_loss, 'bo', label='Validation loss')
   ax[0].set_title('Training and Validation Loss')
   ax[0].set_ylabel('Loss Value')

   accuracy = self.model_history.history['accuracy']
   val_accuracy = self.model_history.history['val_accuracy']
   print('Train_accuracy:{}'.format(np.mean(accuracy)))
   print('Validation_accuracy:{}'.format(np.mean(val_accuracy)))
   ax[1].plot(epochs, accuracy, 'r', label='Training Iou Score')
   ax[1].plot(epochs, val_accuracy, 'bo', label='Validation Iou Score')
   ax[1].set_title('Training and Validation Accuracy')
   ax[1].set_ylabel('Accuracy Score')
   ax[1].set_ylim(0,1)

   iou_score = self.model_history.history['iou_score']
```

```
    val_iou_score = self.model_history.history['val_iou_score']

    print('Train_IOU:{}'.format(np.mean(iou_score)))
    print('Validation_IOU:{}'.format(np.mean(val_iou_score)))
    ax[2].plot(epochs, iou_score, 'r', label='Training Iou Score')
    ax[2].plot(epochs, val_iou_score, 'bo', label='Validation Iou Score')
    ax[2].set_title('Training and Validation IOU Score')
    ax[2].set_ylabel('IOU Score')
    ax[2].set_ylim(0,1)

    for i in range(3):
        ax[i].set_xlabel('Epoch')
        ax[i].legend();
    plt.show()



  def find_loss(self):
    loss_answer=self.get_good_answer("\n Which loss function do you want to use ? \n -'cross_entropy': fastest to co
mpute, \n -'dice_loss': Overlap measure that performs better at class imbalanced problems \n -'focal_loss' : To down-
weight the contribution of easy examples so that the CNN focuses more on hard examples \n Could also be a mix of
those loss functions \n Examples : \n - cross_entropy + dice_loss \n - dice_loss + focal_loss \n ", acceptable_answers
=['cross_entropy','dice_loss','focal_loss','cross_entropy + dice_loss','cross_entropy + focal_loss','dice_loss + focal_lo
ss','cross_entropy + dice_loss + focal_loss'])
    if '+' in loss_answer :
        weights=self.get_good_answer(" \n Because you chose multiple loss functions, set the weights given to the diff
erent loss_functions separated by space : \n Examples : if 'dice_loss + focal_loss' was given, answers could be '1 1' o
r '2 1' \n", acceptable_answers=None)
        weights = weights.split()
        weights=[float(i) for i in weights]
    else :
        weights=[1]
    loss=[]
    if 'cross_entropy' in loss_answer:
      loss.append(tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
    if 'dice_loss' in loss_answer :
        weights_class=input("\n Because you chose dice_loss, you need to specify the weights you give to each class se
parated by space:  \n for example, if you have 2 classes +background :('backgound':0',Car':1,'Human':2)and you hav
e a hard time prediction humans :\n your answer should be something like '0.5 1 4' or '1 2 3' \n")
        weights_class = weights_class.split()
        weights_class=[float(i) for i in weights_class]
        assert len(weights_class) == self.n_classes

        loss.append(sm.losses.DiceLoss(class_weights=np.array(weights_class)))
    if 'focal_loss' in loss_answer :
      loss.append(sm.losses.BinaryFocalLoss() if self.n_classes == 1 else sm.losses.CategoricalFocalLoss())
    assert len(loss)==len(weights)
    return loss,weights


  def show_predictions(self,generator=None, num=3):
    if generator ==None:
      generator = self.train_generator
    for i in range(num):
```

```
      image, mask=next(generator)
      sample_image, sample_mask= image[1], mask[1]
      image = np.expand_dims(sample_image, axis=0)

      pr_mask = self.model.predict(image)
      pr_mask=np.expand_dims(pr_mask[0].argmax(axis=-1),axis=-1)
      cv2.imwrite(PATH+"predicted_mask/"+str(i)+'.png',pr_mask)

      display_user([sample_image, sample_mask,pr_mask])
      intersection=np.logical_and(sample_mask,pr_mask)
      union=np.logical_or(sample_mask,pr_mask)
      score=np.sum(intersection)/np.sum(union)
      print(score)


  def predict_unlabeled(self,show_predictions=True):

    test_datagen= ImageDataGenerator(rescale=1./255)
    test_datagen_mask = ImageDataGenerator(rescale=1)

    unlabeled_image_generator = test_datagen.flow_from_directory(PATH+'test_x',class_mode=None,batch_size = 1
)
    unlabeled_mask_generator =test_datagen_mask.flow_from_directory(PATH+'test_y',class_mode=None,batch_siz
e=1)
    if show_predictions:
      for _ in range(3):
        sample_image=next(unlabeled_image_generator)
        test_mask=next(unlabeled_mask_generator)

        sample_mask = self.model.predict(sample_image)
        sample_mask=np.expand_dims(sample_mask[0].argmax(axis=-1),axis=-1)
        display_user([sample_image[0],sample_mask],title=['Input Image','Predicted Mask'])

        intersection=np.logical_and(test_mask,sample_mask)
        union=np.logical_or(test_mask,sample_mask)
        score=np.sum(intersection)/np.sum(union)
        print(score)



    return self.model.predict(unlabeled_image_generator,steps=len(os.listdir(PATH+'test_x'+'/t_x')))


def display_user(display_list,title=['Input Image', 'True Mask', 'Predicted Mask']):
  plt.figure(figsize=(15, 15))
  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]),cmap='magma')
    plt.axis('off')
  plt.show()
```