```
import json, cv2, glob, os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import keras
from keras import backend as K
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input, Reshape, Conv2D, MaxPooling2D, Conv2DTranspose, Concatenate
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
```

```
!unzip '/content/drive/MyDrive/Machine_Learning/Cut_Person2.zip'
```

```
Archive:  /content/drive/MyDrive/Machine_Learning/Cut_Person2.zip
  inflating: Cut_Person2/README.dataset.txt
  inflating: Cut_Person2/README.roboflow.txt
   creating: Cut_Person2/test/
  inflating: Cut_Person2/test/_annotations.coco.json
  inflating: Cut_Person2/test/cr_120_jpg.rf.e746b7e161f7d1c9b5c38906f22197d3.jpg
  inflating: Cut_Person2/test/cr_126_jpg.rf.d5dc45f4f660131cbf6a6af03b5c704c.jpg
  inflating: Cut_Person2/test/cr_162_jpg.rf.cfe502a358f6ba770b7c771c857dba4d.jpg
  inflating: Cut_Person2/test/cr_171_jpg.rf.d1bd43b08f67b18236c840f9373c4a86.jpg
  inflating: Cut_Person2/test/cr_195_jpg.rf.f92c65538a2bdbc7e4c8c3521f6c2539.jpg
  inflating: Cut_Person2/test/cr_2_jpg.rf.4f078740576a0201caa0614ae3172a4f.jpg
  inflating: Cut_Person2/test/cr_205_jpg.rf.918afe1163885da9fddee902aac74a69.jpg
  inflating: Cut_Person2/test/cr_216_jpg.rf.832891328326117512634f4be05c97dc.jpg
  inflating: Cut_Person2/test/cth_104_jpg.rf.d9dc039ccff5802708bdd9109592b425.jpg
  inflating: Cut_Person2/test/cth_11_jpg.rf.aeef126e06eb6a3fae11c9299547fbb1.jpg
  inflating: Cut_Person2/test/cth_167_jpg.rf.b67b891814056d69938213d9604a7d53.jpg
  inflating: Cut_Person2/test/cth_173_jpg.rf.af05762e6daf634cf07725bb93c9e664.jpg
  inflating: Cut_Person2/test/cth_192_jpg.rf.46a3ead72dd7d36a23cf3444cdc6e468.jpg
  inflating: Cut_Person2/test/cth_21_jpg.rf.4d1366831fb96f4f90b4f4db0a3e8381.jpg
  inflating: Cut_Person2/test/cth_34_jpg.rf.3c8c98ae5f439726135a7b396d39376e.jpg
  inflating: Cut_Person2/test/cth_56_jpg.rf.4439288784183a7f656463e789e6555c.jpg
  inflating: Cut_Person2/test/cth_71_jpg.rf.0f2700424a7546611f1755bfb4f0b564.jpg
  inflating: Cut_Person2/test/ctn_102_jpg.rf.5eede42ba27c0cca40cb54e7f8db7ab6.jpg
  inflating: Cut_Person2/test/ctn_107_jpg.rf.335888eb3d73216afaab594308e9d89c.jpg
  inflating: Cut_Person2/test/ctn_11_jpg.rf.1c8316fd84f8ab80f51f455b530eda69.jpg
  inflating: Cut_Person2/test/ctn_111_jpg.rf.b5c81d678cf8ec418db5b4485723b5d2.jpg
  inflating: Cut_Person2/test/ctn_151_jpg.rf.b50867f94106186e19ef86981e00a703.jpg
  inflating: Cut_Person2/test/ctn_169_jpg.rf.ec191ccd84eab42bf2ea5bc668d914fd.jpg
  inflating: Cut_Person2/test/ctn_2_jpg.rf.18e826d78f8dca95a1f863edaf8cf0b7.jpg
  inflating: Cut_Person2/test/ctn_27_jpg.rf.e482b354c2626869ab09de315b31c497.jpg
  inflating: Cut_Person2/test/ctn_28_jpg.rf.784d2f649da829c425ced45034dfc46d.jpg
  inflating: Cut_Person2/test/ctn_7_jpg.rf.fff801404d8cb296447e576d80781241.jpg
  inflating: Cut_Person2/test/CTT_100_jpg.rf.550eed7e74940f40e755f3107fee478a.jpg
  inflating: Cut_Person2/test/CTT_107_jpg.rf.1060e5ff55a0d2725668835e3c82efc8.jpg
  inflating: Cut_Person2/test/CTT_110_jpg.rf.17c121ef3457172cc51dd8f6bc66abdd.jpg
  inflating: Cut_Person2/test/CTT_113_jpg.rf.6db0c36f0b1be107a6f2d64a21ceb547.jpg
  inflating: Cut_Person2/test/CTT_177_jpg.rf.47e653e8d9e8bd7e621a48b318f90056.jpg
  inflating: Cut_Person2/test/CTT_178_jpg.rf.05e506eceaad9587da70c611878804f8.jpg
  inflating: Cut_Person2/test/CTT_179_jpg.rf.08106ea365b71f65ce5b852fee0c34a7.jpg
  inflating: Cut_Person2/test/CTT_207_jpg.rf.34f24c0deec78413453988ff3a9caa61.jpg
  inflating: Cut_Person2/test/CTT_233_jpg.rf.48d5116c596cdd66c347a4d3e67456a9.jpg
  inflating: Cut_Person2/test/CTT_246_jpg.rf.8d4530551c8f4a8ad0e175c822e8d5b2.jpg
  inflating: Cut_Person2/test/CTT_3_jpg.rf.401b6c9701f666620b7931ae6e05a38d.jpg
  inflating: Cut_Person2/test/CTT_33_jpg.rf.45bc47a44f029946738d416638dffb58.jpg
  inflating: Cut_Person2/test/CTT_39_jpg.rf.835f09792f9b9a9052bef5f6ef67ec69.jpg
  inflating: Cut_Person2/test/CTT_51_jpg.rf.f11782e45dc4657a869cd0f9b50cd822.jpg
  inflating: Cut_Person2/test/CTT_8_jpg.rf.fa154dc290bde412ded2814edf6e5622.jpg
  inflating: Cut_Person2/test/CTT_87_jpg.rf.c4e36fc62ab3472bf913e2ab00621c76.jpg
  inflating: Cut_Person2/test/CTT_88_jpg.rf.be9d7e12248eed0c6e42f32638318ae3.jpg
  inflating: Cut_Person2/test/CTT_90_jpg.rf.1a697ae0111c2074c0d6f30dda2b64cc.jpg
  inflating: Cut_Person2/test/CTT_91_jpg.rf.72255d69dcf5809051941a44d54f3e63.jpg
  inflating: Cut_Person2/test/ctt2_147_jpg.rf.68cf7f54f81d40db0597794423a5dd10.jpg
  inflating: Cut_Person2/test/ctt2_162_jpg.rf.9f15e351729a95929b47218b9c6b823f.jpg
  inflating: Cut_Person2/test/ctt2_168_jpg.rf.e8cdea2b1f6ff043d016f6bcbdfcb074.jpg
  inflating: Cut_Person2/test/ctt2_20_jpg.rf.92445bec4c415ecfc2870bdcdaa4f942.jpg
  inflating: Cut_Person2/test/ctt2_201_jpg.rf.f5d542e285a8aa0e0837bbe0c14aaddf.jpg
  inflating: Cut_Person2/test/ctt2_62_jpg.rf.3abb1be1fb00ca55b69e88c9f1552d4e.jpg
  inflating: Cut_Person2/test/ctt2_82_jpg.rf.3167585b9ca4637aa3a8a9a5c7c08a17.jpg
```

```
train_path = '/content/Cut_Person2/train/'
valid_path = '/content/Cut_Person2/valid/'
test_path = '/content/Cut_Person2/test/'
```

```
json_name = '_annotations.coco.json'
train_file = open(train_path+json_name)
data = json.load(train_file)

train_id_segment = []
#train_segment = []
train_segment_frame = []
train_file_frame = []
for i in data['annotations']:
    image_id = i['image_id']
    value = np.array(i['segmentation'])
    black = np.zeros((640, 640, 1))
    change = np.reshape(value[0], (int(len(value[0])/2), 2))
    change = np.array(change, np.int32)
    black = cv2.fillPoly(black, [change], (255, 255, 255))
    black[black >= 1] = 1
    black[black == 0] = 0

    # train_frame.append((id, name_img, value))
    train_segment_frame.append([image_id, black])

for j in data['images']:
    id = j['id']
    name_img = j['file_name']
    train_file_frame.append([id, name_img])
train_file.close()

train_data = []
for i in range(len(train_segment_frame)):
    for j in range(len(train_file_frame)):
        if train_segment_frame[i][0] == train_file_frame[j][0]:
            train_data.append([i, train_file_frame[j][1], train_segment_frame[i][1]])


test_file = open(test_path+json_name)
data = json.load(test_file)

test_id_segment = []
#train_segment = []
test_segment_frame = []
test_file_frame = []
for i in data['annotations']:
    image_id = i['image_id']
    value = np.array(i['segmentation'])
    black = np.zeros((640, 640, 1))
    change = np.reshape(value[0], (int(len(value[0])/2), 2))
    change = np.array(change, np.int32)
    black = cv2.fillPoly(black, [change], (255, 255, 255))
    black[black >= 1] = 1
    black[black == 0] = 0
    # train_frame.append((id, name_img, value))
    test_segment_frame.append([image_id, black])

for j in data['images']:
    id = j['id']
    name_img = j['file_name']
    test_file_frame.append([id, name_img])
test_file.close()

test_data = []
for i in range(len(test_segment_frame)):
    for j in range(len(test_file_frame)):
        if test_segment_frame[i][0] == test_file_frame[j][0]:
            test_data.append([i, test_file_frame[j][1], test_segment_frame[i][1]])
valid_file = open(valid_path+json_name)
data = json.load(valid_file)

valid_id_segment = []
#train_segment = []
valid_segment_frame = []
valid_file_frame = []
for i in data['annotations']:
    image_id = i['image_id']
    value = np.array(i['segmentation'])
    black = np.zeros((640, 640, 1))
    change = np.reshape(value[0], (int(len(value[0])/2), 2))
    change = np.array(change, np.int32)
    black = cv2.fillPoly(black, [change], (255, 255, 255))
    black[black >= 1] = 1
    black[black == 0] = 0


    # train_frame.append((id, name_img, value))
    valid_segment_frame.append([image_id, black])

for j in data['images']:
    id = j['id']
    name_img = j['file_name']
    valid_file_frame.append([id, name_img])
valid_file.close()

valid_data = []
for i in range(len(valid_segment_frame)):
    for j in range(len(valid_file_frame)):
        if valid_segment_frame[i][0] == valid_file_frame[j][0]:
            valid_data.append([i, valid_file_frame[j][1], valid_segment_frame[i][1]])


x_train = []
y_train = []
x_valid = []
y_valid = []
x_test = []
y_test = []
for i in range(int(3*len(train_data)/4)):
    img = cv2.imread(train_path+train_data[i][1])
    img = cv2.resize(img, (160, 160))/255.
    img = np.array(img, dtype=np.float32)
    x_train.append(img)

    mask = train_data[i][2]
    mask = cv2.resize(mask, (160, 160))
    y_train.append(mask)






for i in range(int(len(valid_data)/2)):
    img = cv2.imread(valid_path+valid_data[i][1])
    img = cv2.resize(img, (160, 160))/255.
    img = np.array(img, dtype=np.float32)
    x_valid.append(img)

    mask = valid_data[i][2]
    mask = cv2.resize(mask, (160, 160))
    y_valid.append(mask)


for i in range(len(test_data)):
    img = cv2.imread(test_path+test_data[i][1])
    img = cv2.resize(img, (160, 160))/255.
    img = np.array(img, dtype=np.float32)
    x_test.append(img)

    mask = test_data[i][2]
    mask = cv2.resize(mask, (160, 160))
    y_test.append(mask)
```

```
x_train = np.array(x_train)
y_train = np.array(y_train)
x_valid = np.array(x_valid)
y_valid = np.array(y_valid)
x_test = np.array(x_test)
y_test = np.array(y_test)
```

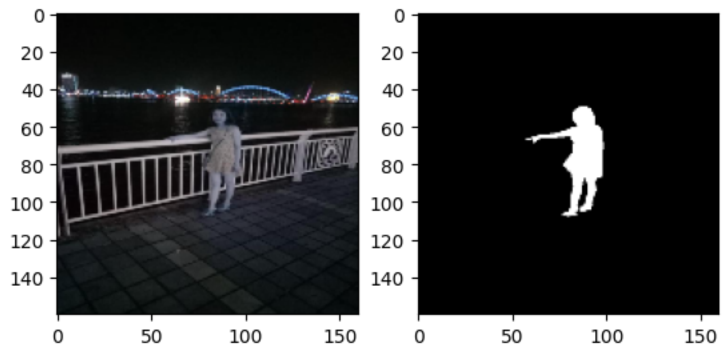
```
y_train = y_train.reshape((y_train.shape[0], y_train.shape[1], y_train.shape[2], 1))
y_valid = y_valid.reshape((y_valid.shape[0], y_valid.shape[1], y_valid.shape[2], 1))
y_test = y_test.reshape((y_test.shape[0], y_test.shape[1], y_test.shape[2], 1))
```

```
print(y_train.shape)
```

```
(1380, 160, 160, 1)
```

```
plt.subplot(1, 2, 1)
plt.imshow(x_valid[10])
plt.subplot(1, 2, 2)
plt.imshow(y_valid[10], 'gray')
```

```
<matplotlib.image.AxesImage at 0x7b14e7170c70>
```



```
img = x_train[0]
img2 = y_train[0]
img2 = cv2.resize(img2, (160, 160))
img2 = np.array(img2, np.int32)
img2 = img2.squeeze()
img2 = np.stack((img2,)*3, axis=-1)

print(img)
plt.imshow(np.concatenate([x_train[0], img2, img*img2], axis = 1))
```

```
 [[[0.78039217 0.8156863  0.73333335]
   [0.78039217 0.81960785 0.73333335]
   [0.78039217 0.8235294  0.72156864]
   ...
   [0.78431374 0.8235294  0.73333335]
   [0.7647059  0.8039216  0.7137255 ]
   [0.76862746 0.80784315 0.7176471 ]]

  [[0.77254903 0.80784315 0.7254902 ]
   [0.77254903 0.8117647  0.7254902 ]
   [0.7764706  0.81960785 0.7176471 ]
   ...
   [0.78431374 0.8235294  0.73333335]
   [0.76862746 0.80784315 0.7176471 ]
   [0.76862746 0.80784315 0.7176471 ]]

  [[0.7764706  0.8117647  0.7294118 ]
   [0.7764706  0.8156863  0.7294118 ]
   [0.78039217 0.8235294  0.72156864]
   ...
   [0.78039217 0.81960785 0.7294118 ]
   [0.7764706  0.8156863  0.7254902 ]
   [0.7764706  0.8156863  0.7254902 ]]

  ...

  [[0.4862745  0.44705883 0.3764706 ]
   [0.4862745  0.44705883 0.3764706 ]
   [0.47843137 0.4392157  0.36862746]
   ...
   [0.41960785 0.39607844 0.32156864]
   [0.38039216 0.35686275 0.28235295]
   [0.3529412  0.32941177 0.25490198]]
```

Build Model

```
   [0.4862745  0.45490196 0.38039216]
```

```
def mean_iou(y_true, y_pred):
    yt0 = y_true[:,:,:,0]
    yp0 = K.cast(y_pred[:,:,:,0] > 0.5, 'float32')
    inter = tf.compat.v1.count_nonzero(tf.logical_and(tf.equal(yt0, 1), tf.equal(yp0, 1)))
    union = tf.compat.v1.count_nonzero(tf.add(yt0, yp0))
    iou = tf.where(tf.equal(union, 0), 1., tf.cast(inter/union, 'float32'))
    return iou
```

```
   [0.45882353 0.41960785 0.34901962]]]
```

```
# def encoder_block(inputs, num_filters):

#     # Convolution with 3x3 filter followed by ReLU activation
#     x = tf.keras.layers.Conv2D(num_filters,
#                                3,
#                                padding = 'valid')(inputs)
#     x = tf.keras.layers.Activation('relu')(x)

#     # Convolution with 3x3 filter followed by ReLU activation
#     x = tf.keras.layers.Conv2D(num_filters,
#                                3,
#                                padding = 'valid')(x)
#     x = tf.keras.layers.Activation('relu')(x)

#     # Max Pooling with 2x2 filter
#     x = tf.keras.layers.MaxPool2D(pool_size = (2, 2),
#                                   strides = 2)(x)

#     return x
# def decoder_block(inputs, skip_features, num_filters):

#     # Upsampling with 2x2 filter
#     x = tf.keras.layers.Conv2DTranspose(num_filters,
#                                         (2, 2),
#                                         strides = 2,
#                                         padding = 'valid')(inputs)

#     # Copy and crop the skip features
#     # to match the shape of the upsampled input
#     skip_features = tf.image.resize(skip_features,
#                                     size = (x.shape[1],
#                                             x.shape[2]))
#     x = tf.keras.layers.Concatenate()([x, skip_features])

#     # Convolution with 3x3 filter followed by ReLU activation
#     x = tf.keras.layers.Conv2D(num_filters,
#                                3,
#                                padding = 'valid')(x)
#     x = tf.keras.layers.Activation('relu')(x)

#     # Convolution with 3x3 filter followed by ReLU activation
#     x = tf.keras.layers.Conv2D(num_filters, 3, padding = 'valid')(x)
#     x = tf.keras.layers.Activation('relu')(x)

#     return x


# def unet(input_shape = (576, 576, 3), num_classes = 1):
#     inputs = tf.keras.layers.Input(input_shape)

#     s1 = encoder_block(inputs, 64)
#     s2 = encoder_block(s1, 128)
#     s3 = encoder_block(s2, 256)
#     s4 = encoder_block(s3, 512)

#     # Bottleneck
#     b1 = tf.keras.layers.Conv2D(1024, 3, padding = 'valid')(s4)
#     b1 = tf.keras.layers.Activation('relu')(b1)
#     b1 = tf.keras.layers.Conv2D(1024, 3, padding = 'valid')(b1)
#     b1 = tf.keras.layers.Activation('relu')(b1)

#     # Expansive Path
#     s5 = decoder_block(b1, s4, 512)
#     s6 = decoder_block(s5, s3, 256)
#     s7 = decoder_block(s6, s2, 128)
#     s8 = decoder_block(s7, s1, 64)

#     outputs = tf.keras.layers.Conv2D(num_classes,
#                                      1,
#                                      padding = 'valid',
#                                      activation = 'sigmoid')(s8)


#     model = Model(inputs=[inputs], outputs=[outputs])
#     model.compile(optimizer= tf.keras.optimizers.experimental.RMSprop(learning_rate=0.002), loss = 'binary_crossentropy', metrics = [mean_

#     return model
```

```
def unet(sz = (160, 160, 3)):
  x = Input(sz)
  inputs = x

  #down sampling
  f = 12
  layers = []

  for i in range(0, 5):
    x = Conv2D(f, 3, activation='relu', padding='same') (x)
    x = Conv2D(f, 3, activation='relu', padding='same') (x)
    layers.append(x)
    x = MaxPooling2D() (x)
    f = f*2
  ff2 = 64

  #bottleneck
  j = len(layers) - 1
  x = Conv2D(f, 3, activation='relu', padding='same') (x)
  x = Conv2D(f, 3, activation='relu', padding='same') (x)
  x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same') (x)
  x = Concatenate(axis=3)([x, layers[j]])
  j = j -1

  #upsampling
  for i in range(0, 4):
    ff2 = ff2//2
    f = f // 2
    x = Conv2D(f, 3, activation='relu', padding='same') (x)
    x = Conv2D(f, 3, activation='relu', padding='same') (x)
    x = Conv2DTranspose(ff2, 2, strides=(2, 2), padding='same') (x)
    x = Concatenate(axis=3)([x, layers[j]])
    j = j -1


  #classification
  x = Conv2D(f, 3, activation='relu', padding='same') (x)
  x = Conv2D(f, 3, activation='relu', padding='same') (x)
  outputs = Conv2D(1, 1, activation='sigmoid') (x)

  #model creation
  model = Model(inputs=[inputs], outputs=[outputs])
  model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = [mean_iou])

  return model


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

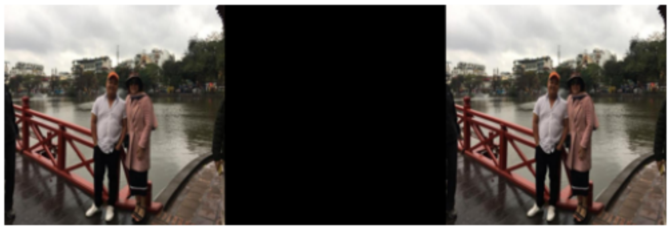
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 160, 160, 3)]        0         []

 conv2d (Conv2D)             (None, 160, 160, 12)         336       ['input_1[0][0]']

 conv2d_1 (Conv2D)           (None, 160, 160, 12)         1308      ['conv2d[0][0]']

 max_pooling2d (MaxPooling2  (None, 80, 80, 12)           0         ['conv2d_1[0][0]']
 D)

 conv2d_2 (Conv2D)           (None, 80, 80, 24)           2616      ['max_pooling2d[0][0]']

 conv2d_3 (Conv2D)           (None, 80, 80, 24)           5208      ['conv2d_2[0][0]']

 max_pooling2d_1 (MaxPoolin  (None, 40, 40, 24)           0         ['conv2d_3[0][0]']
 g2D)

 conv2d_4 (Conv2D)           (None, 40, 40, 48)           10416     ['max_pooling2d_1[0][0]']

 conv2d_5 (Conv2D)           (None, 40, 40, 48)           20784     ['conv2d_4[0][0]']

 max_pooling2d_2 (MaxPoolin  (None, 20, 20, 48)           0         ['conv2d_5[0][0]']
 g2D)

 conv2d_6 (Conv2D)           (None, 20, 20, 96)           41568     ['max_pooling2d_2[0][0]']

 conv2d_7 (Conv2D)           (None, 20, 20, 96)           83040     ['conv2d_6[0][0]']
 max_pooling2d_3 (MaxPoolin  (None, 10, 10, 96)           0         ['conv2d_7[0][0]']
 g2D)

 conv2d_8 (Conv2D)           (None, 10, 10, 192)          166080    ['max_pooling2d_3[0][0]']

 conv2d_9 (Conv2D)           (None, 10, 10, 192)          331968    ['conv2d_8[0][0]']

 max_pooling2d_4 (MaxPoolin  (None, 5, 5, 192)            0         ['conv2d_9[0][0]']
 g2D)

 conv2d_10 (Conv2D)          (None, 5, 5, 384)            663936    ['max_pooling2d_4[0][0]']

 conv2d_11 (Conv2D)          (None, 5, 5, 384)            1327488   ['conv2d_10[0][0]']

 conv2d_transpose (Conv2DTr  (None, 10, 10, 64)           98368     ['conv2d_11[0][0]']
 anspose)

 concatenate (Concatenate)   (None, 10, 10, 256)          0         ['conv2d_transpose[0][0]',
                                                                     'conv2d_9[0][0]']

 conv2d_12 (Conv2D)          (None, 10, 10, 192)          442560    ['concatenate[0][0]']

 conv2d_13 (Conv2D)          (None, 10, 10, 192)          331968    ['conv2d_12[0][0]']

 conv2d_transpose_1 (Conv2D  (None, 20, 20, 32)           24608     ['conv2d_13[0][0]']
 Transpose)
```

```
test_filename = []
for i in range(len(test_data)):
    test_filename.append(test_data[i][1])
```

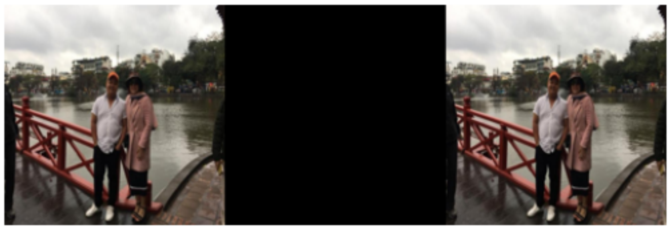
```
def build_callbacks():
        checkpointer = ModelCheckpoint(filepath='unet.h5', verbose=0, save_best_only=True, save_weights_only=True)
        callbacks = [checkpointer, PlotLearning()]
        return callbacks

# inheritance for training process plot
class PlotLearning(keras.callbacks.Callback):

    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []
        self.acc = []
        self.val_acc = []
        #self.fig = plt.figure()
        self.logs = []
    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.acc.append(logs.get('mean_iou'))
        self.val_acc.append(logs.get('val_mean_iou'))
        self.i += 1
        print('i=',self.i,'loss=',logs.get('loss'),'val_loss=',logs.get('val_loss'),'mean_iou=',logs.get('mean_iou'),'val_mean_iou=',logs.ge

        #choose a random test image and preprocess
        path = np.random.choice(test_filename)
        raw = Image.open(test_path+path)
        raw = np.array(raw.resize((160, 160)))/255.
        raw = raw[:,:,0:3]

        #predict the mask
        pred = model.predict(np.expand_dims(raw, 0))

        #mask post-processing
        msk  = pred.squeeze()
        msk = np.stack((msk,)*3, axis=-1)
        msk[msk >= 0.5] = 1
        msk[msk < 0.5] = 0

        #show the mask and the segmented image
        combined = np.concatenate([raw, msk, raw*(1-msk)], axis = 1)
        plt.axis('off')
        plt.imshow(combined)
        plt.show()


print(x_train.shape, y_train.shape)
print(x_valid.shape, y_valid.shape)
print(x_test.shape, y_test.shape)
```

```
(1380, 160, 160, 3) (1380, 160, 160, 1)
(159, 160, 160, 3) (159, 160, 160, 1)
(73, 160, 160, 3) (73, 160, 160, 1)
```

```
history = model.fit(x_train, y_train, batch_size = 8, epochs=100, validation_data = (x_valid, y_valid), callbacks = build_callbacks(), verbo
i= 1 loss= 0.34794414043426514 val_loss= 0.3269573748111725 mean_iou= 0.0010524748358
1/1 [==============================] - 1s 650ms/step
```

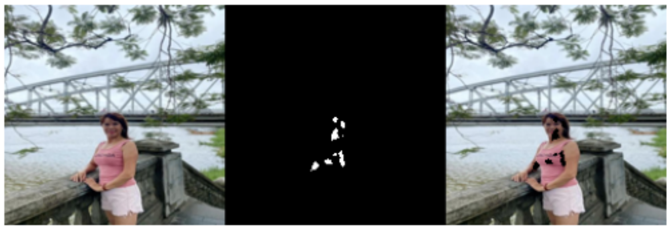


```
i= 2 loss= 0.3109108507633209 val_loss= 0.3018815815448761 mean_iou= 0.0 val_mean_iou
1/1 [==============================] - 0s 19ms/step
```



```
i= 3 loss= 0.280737042427063 val_loss= 0.28699204325675964 mean_iou= 0.00377267436124
1/1 [==============================] - 0s 62ms/step
```

```
i= 4 loss= 0.2603984475135803 val_loss= 0.250316321849823 mean_iou= 0.093492329120635
1/1 [==============================] - 0s 18ms/step
```



```
i= 5 loss= 0.23784394562244415 val_loss= 0.2781476080417633 mean_iou= 0.2691929042339
1/1 [==============================] - 0s 22ms/step
```