Введение в искусственные нейронные сети

Урок 6. Сегментация

Практическое задание

1. Попробуйте обучить нейронную сеть U-Net на любом другом датасете.
2. Опишите результат. Что помогло вам улучшить ее точность?

Используемый dataset - https://www.kaggle.com/dansbecker/cityscapes-image-pairs (https://www.kaggle.com/dansbecker/cityscapes-image-pairs)

В [1]:
```
import zipfile  # библиотека для работы с zip-архивами
import os  # работа с файловой системой
import time

from google.colab import drive  # Модуль для работы с Google Disc
from PIL import Image  # Модуль для работы с изображениями
```

Использование Google Drive

Подключение Google Drive к виртуальной машине:

В [3]:
```
from google.colab import drive

drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

Просматриваем подключенные диски

В [4]:
```
!df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
overlay          79G   43G   37G  54% /
tmpfs            64M     0   64M   0% /dev
shm             5.7G     0  5.7G   0% /dev/shm
/dev/root       2.0G  1.2G  817M  59% /sbin/docker-init
tmpfs           6.4G   36K  6.4G   1% /var/colab
/dev/sda1        86G   47G   40G  55% /opt/bin/.nvidia
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
drive            15G  2.7G   13G  18% /content/gdrive
```

В [5]:
```
# Просматриваем содержимое диска
!ls /content/gdrive/
```

```
MyDrive
```

В [12]:
```
!ls /content/gdrive/"MyDrive"/nn
```

```
lesson_6
```

В [13]:
```
!cp /content/gdrive/"MyDrive"/nn/lesson_6/cityscapes_data.zip .
```

В [14]:
```
!ls
```

```
cityscapes_data.zip  gdrive  sample_data
```

B [15]:
```
zip_file = "cityscapes_data.zip"
# Распаковываем архив

!unzip '/content/cityscapes_data.zip' -d '/content/data'
```

```
Archive:  /content/cityscapes_data.zip
  inflating: /content/data/cityscapes_data/train/1.jpg
  inflating: /content/data/cityscapes_data/train/10.jpg
  inflating: /content/data/cityscapes_data/train/100.jpg
  inflating: /content/data/cityscapes_data/train/1000.jpg
  inflating: /content/data/cityscapes_data/train/1001.jpg
  inflating: /content/data/cityscapes_data/train/1002.jpg
  inflating: /content/data/cityscapes_data/train/1003.jpg
  inflating: /content/data/cityscapes_data/train/1004.jpg
  inflating: /content/data/cityscapes_data/train/1005.jpg
  inflating: /content/data/cityscapes_data/train/1006.jpg
  inflating: /content/data/cityscapes_data/train/1007.jpg
  inflating: /content/data/cityscapes_data/train/1008.jpg
  inflating: /content/data/cityscapes_data/train/1009.jpg
  inflating: /content/data/cityscapes_data/train/101.jpg
  inflating: /content/data/cityscapes_data/train/1010.jpg
  inflating: /content/data/cityscapes_data/train/1011.jpg
  inflating: /content/data/cityscapes_data/train/1012.jpg
  inflating: /content/data/cityscapes_data/train/1013.jpg
```

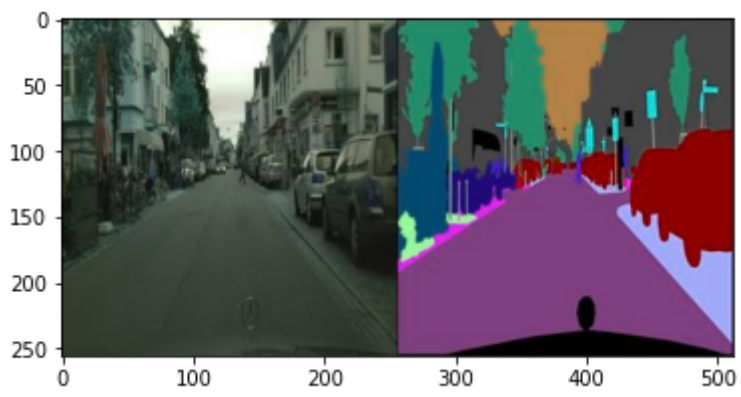
B [16]:
```
!pwd
```

```
/content
```

Работа с изображением

B [61]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

path_train = '/content/data/cityscapes_data/train/'
img1 = cv2.imread(path_train, 0)

file_train_list = os.listdir(path_train)
img0 = cv2.imread(path_train + file_train_list[0])
plt.imshow(img0)
print(np.shape(img0))
print(len(file_train_list))
```

```
(256, 512, 3)
2975
```

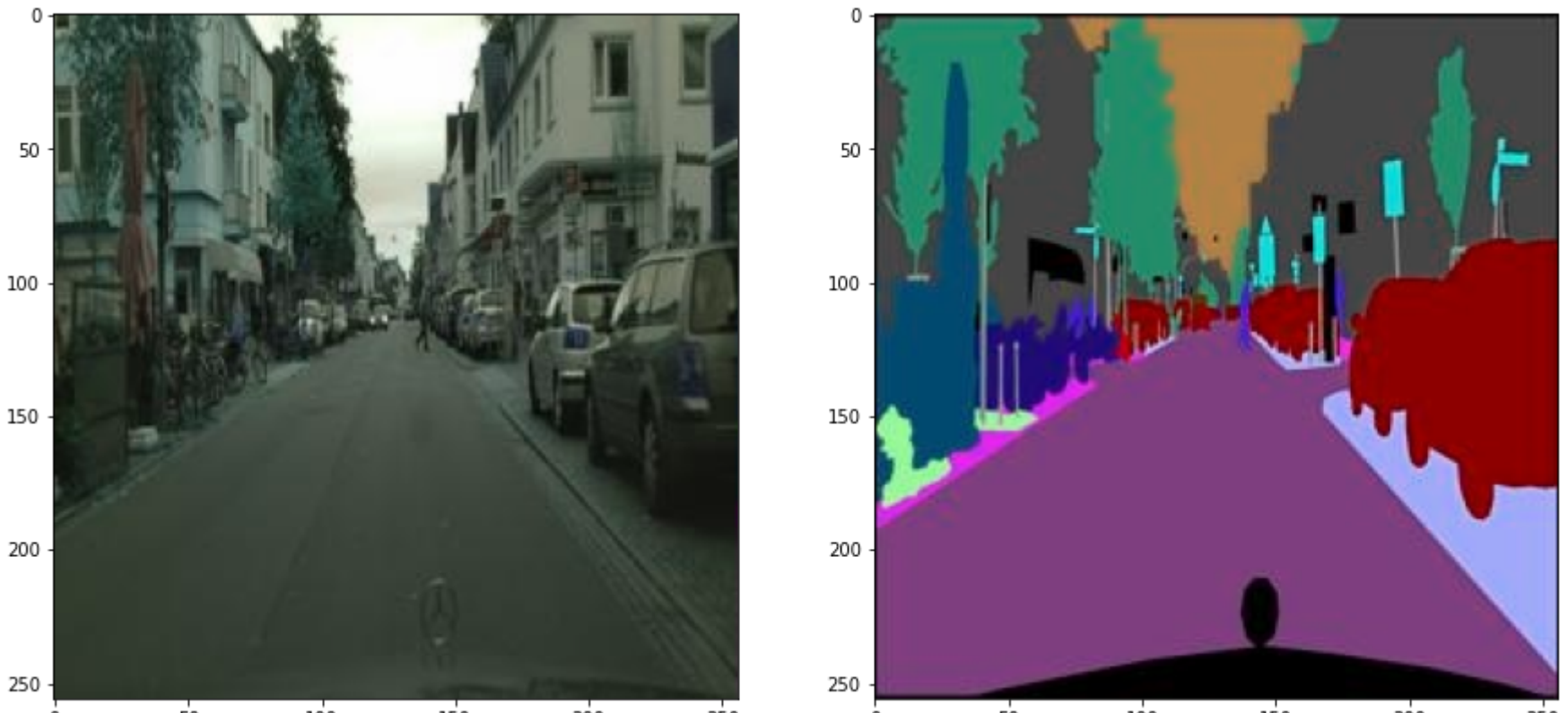


Чтение картинок и формирование тренировочного датасета

B [62]:
```
szy, szx, _ = np.shape(img0)
N_ex = 500  # при 1500 - colab отключается, исчерпан лимит памяти
N_bias = 0
x_train = np.zeros((N_ex, szy, int(szx/2),3))
y_train = np.zeros((N_ex, szy, int(szx/2),3))
k = 0;

for f in file_train_list[N_bias:N_bias + N_ex]:
    x_train[k] = cv2.imread(path_train + f)[:,:256]/256
    y_train[k] = cv2.imread(path_train + f)[:,256:]/256
    k = k + 1
```

B [63]:
```
plt.figure(figsize = (15,15))
plt.subplot(1,2,1)
plt.imshow(x_train[0])
plt.subplot(1,2,2)
plt.imshow(y_train[0])
```

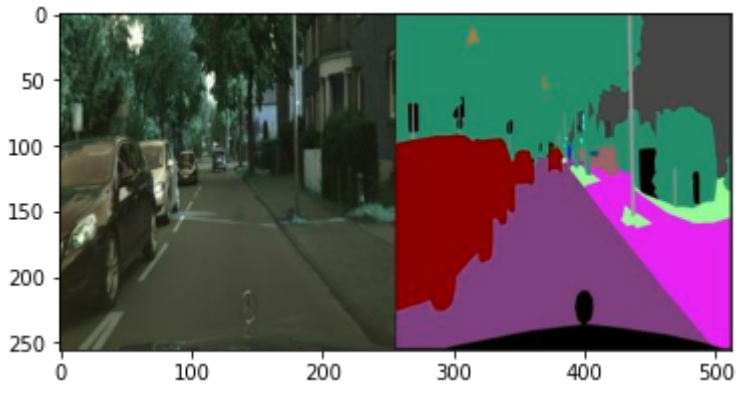
Out[63]: <matplotlib.image.AxesImage at 0x7f764a498410>



Аналогично подготавливаем тестовый дата сет

B [64]:
```
path_val = '/content/data/cityscapes_data/val/'
img1 = cv2.imread(path_val, 0)

file_val_list = os.listdir(path_val)
img0 = cv2.imread(path_val + file_val_list[0])
N_val = 50

plt.imshow(img0)
print(np.shape(img0))
print(len(file_val_list))
```

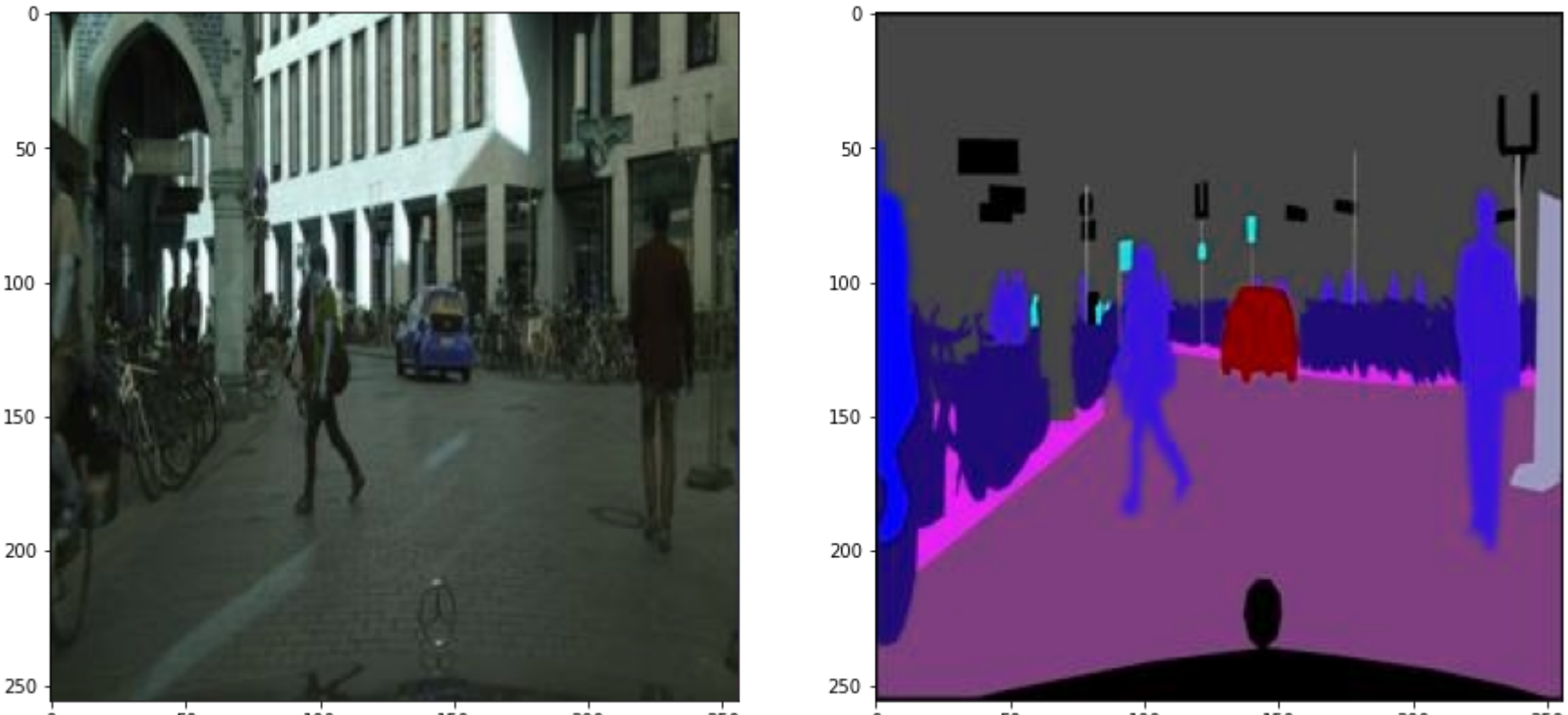
```
(256, 512, 3)
500
```



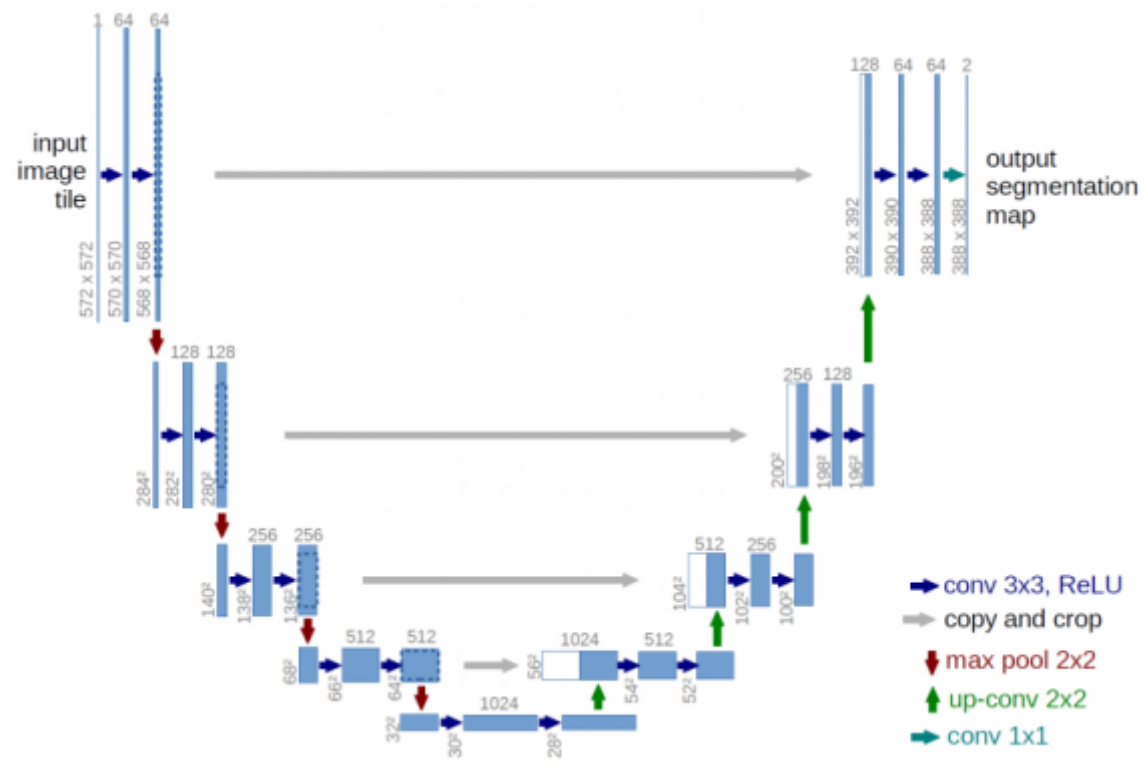
B [65]:
```
szy, szx, _ = np.shape(img0)
x_val = np.zeros((N_val, szy, int(szx/2),3))
y_val = np.zeros((N_val, szy, int(szx/2),3))
k = 0;

for f in file_val_list[0:N_val]:
    x_val[k] = cv2.imread(path_val + f)[:,:256]/256
    y_val[k] = cv2.imread(path_val + f)[:,256:]/256
    k = k + 1
```

B [66]:
```
plt.figure(figsize = (15,15))
plt.subplot(1,2,1)
plt.imshow(x_val[1])
plt.subplot(1,2,2)
plt.imshow(y_val[1])
```

Out[66]: <matplotlib.image.AxesImage at 0x7f7542ced910>



U-net architecture



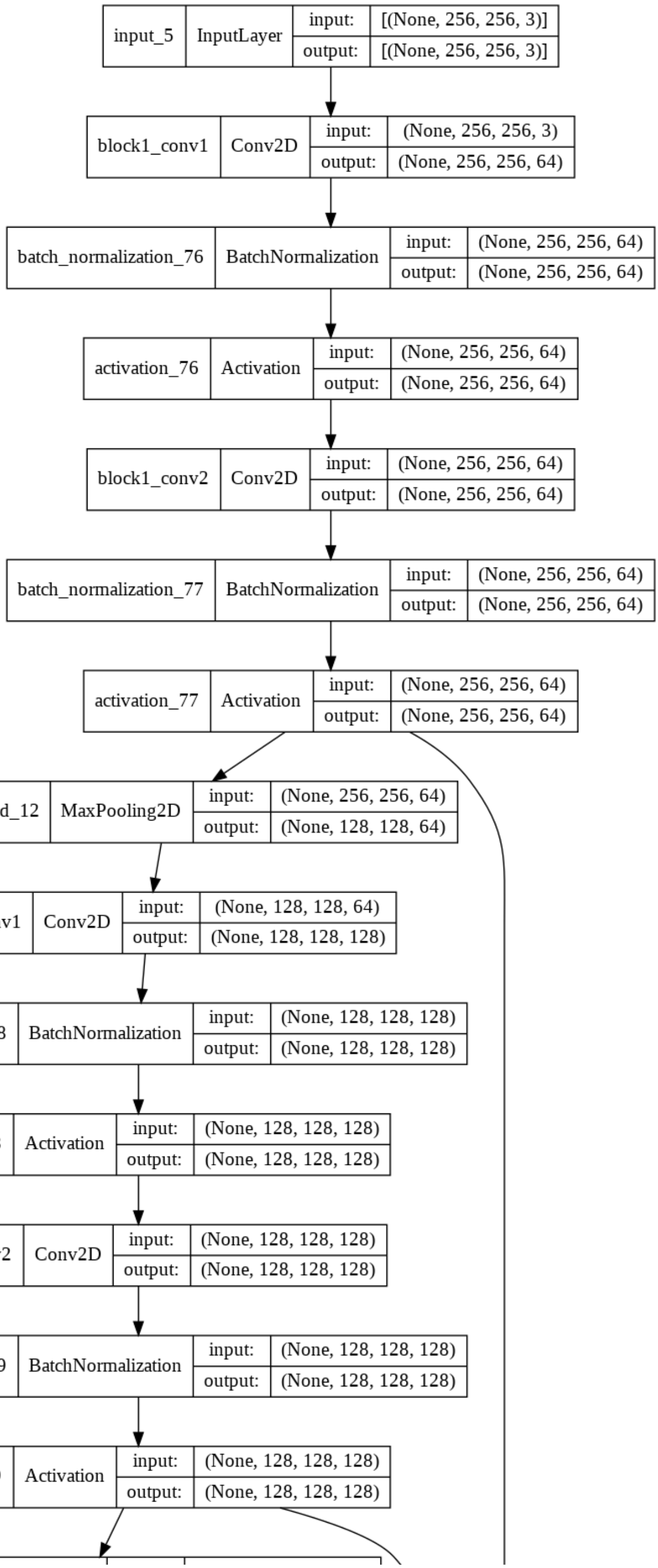
B [67]:
```
# U-net architecture
import tensorflow as tf
import keras
from tensorflow.keras.utils import plot_model

from tensorflow.keras.layers import BatchNormalization
# from keras.layers import Input, Dense, Dropout, Activation, Flatten, Convolution2D, MaxPooling2D, UpSampling2D, Concate
# from tensorflow.keras.layers import Input, Conv2DTranspose, concatenate, Activation, Conv2D

from tensorflow.keras.layers import concatenate, Input, Dense, Dropout, BatchNormalization, Flatten, Conv1D, Conv2D, LSTM
from keras.layers import (Input, Dense, Dropout, Activation, Flatten, Convolution2D,
                          MaxPooling2D, UpSampling2D, Conv2DTranspose, concatenate)
from keras.layers.convolutional import Conv2D
from keras.models import Sequential, Model, model_from_json, load_model
from keras.regularizers import l2
from keras.preprocessing import image
from tensorflow.keras.optimizers import Adam


import keras.backend as K  # !!!

# Это наша метрика на Tensorflow
def dice_coef(y_true, y_pred):
    return (2. * K.sum(y_true * y_pred) + 1.) / (K.sum(y_true) + K.sum(y_pred) + 1.)
```

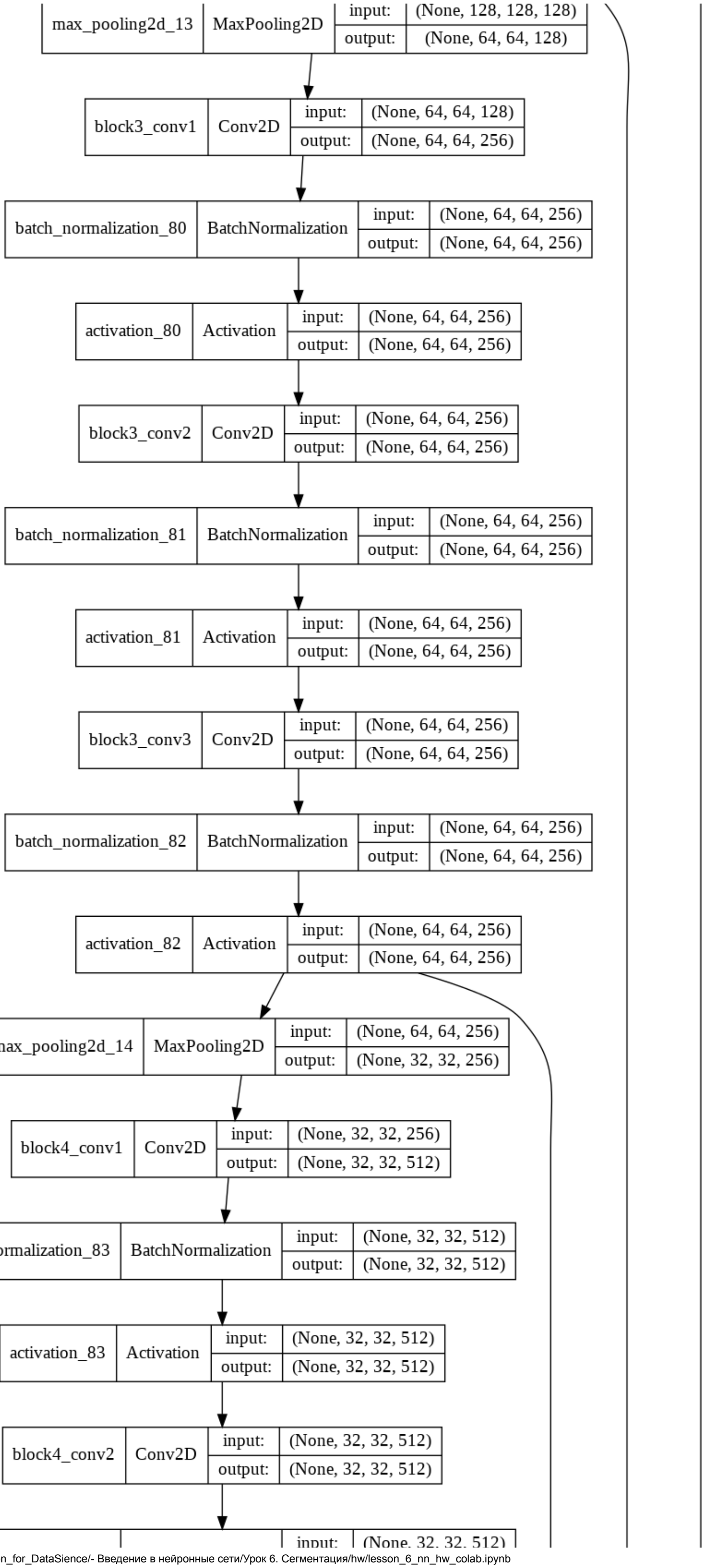
B [68]:
```
def Unet(num_classes = 14, input_shape= (256, 256, 3)):
    img_input = Input(input_shape)

    # Block 1
    x = Conv2D(64, (3, 3), padding='same', name='block1_conv1')(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(64, (3, 3), padding='same', name='block1_conv2')(x)
    x = BatchNormalization()(x)

    # запомним тензор для переноса
    block_1_out = Activation('relu')(x)

    x = MaxPooling2D()(block_1_out) #


    # Block 2
    x = Conv2D(128, (3, 3), padding='same', name='block2_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(128, (3, 3), padding='same', name='block2_conv2')(x)
    x = BatchNormalization()(x)

    # запомним тензор для переноса
    block_2_out = Activation('relu')(x)
    x = MaxPooling2D()(block_2_out) #

    # Block 3
    x = Conv2D(256, (3, 3), padding='same', name='block3_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(256, (3, 3), padding='same', name='block3_conv2')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(256, (3, 3), padding='same', name='block3_conv3')(x)
    x = BatchNormalization()(x)

    # запомним тензор для переноса
    block_3_out = Activation('relu')(x)
    x = MaxPooling2D()(block_3_out) #

    # Block 4
    x = Conv2D(512, (3, 3), padding='same', name='block4_conv1')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(512, (3, 3), padding='same', name='block4_conv2')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(512, (3, 3), padding='same', name='block4_conv3')(x)
    x = BatchNormalization()(x)

    # запомним тензор для переноса
    block_4_out = Activation('relu')(x)

    #x = MaxPooling2D()(block_4_out)

    # Block 5
    #x = Conv2D(512, (3, 3), padding='same', name='block5_conv1')(x)
    #x = BatchNormalization()(x)
    #x = Activation('relu')(x)

    #x = Conv2D(512, (3, 3), padding='same', name='block5_conv2')(x)
    #x = BatchNormalization()(x)
    #x = Activation('relu')(x)

    #x = Conv2D(512, (3, 3), padding='same', name='block5_conv3')(x)
    #x = BatchNormalization()(x)
    #x = Activation('relu')(x)


    # Load pretrained weights.
    #for_pretrained_weight = MaxPooling2D()(x)
    #vgg16 = Model(img_input, for_pretrained_weight)
    #vgg16.load_weights('vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', by_name=True)

    # Растягиваем картинку
    # UP 1
    x = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # конкатенируем то что получили после 3-го увеличения и блок 3
```

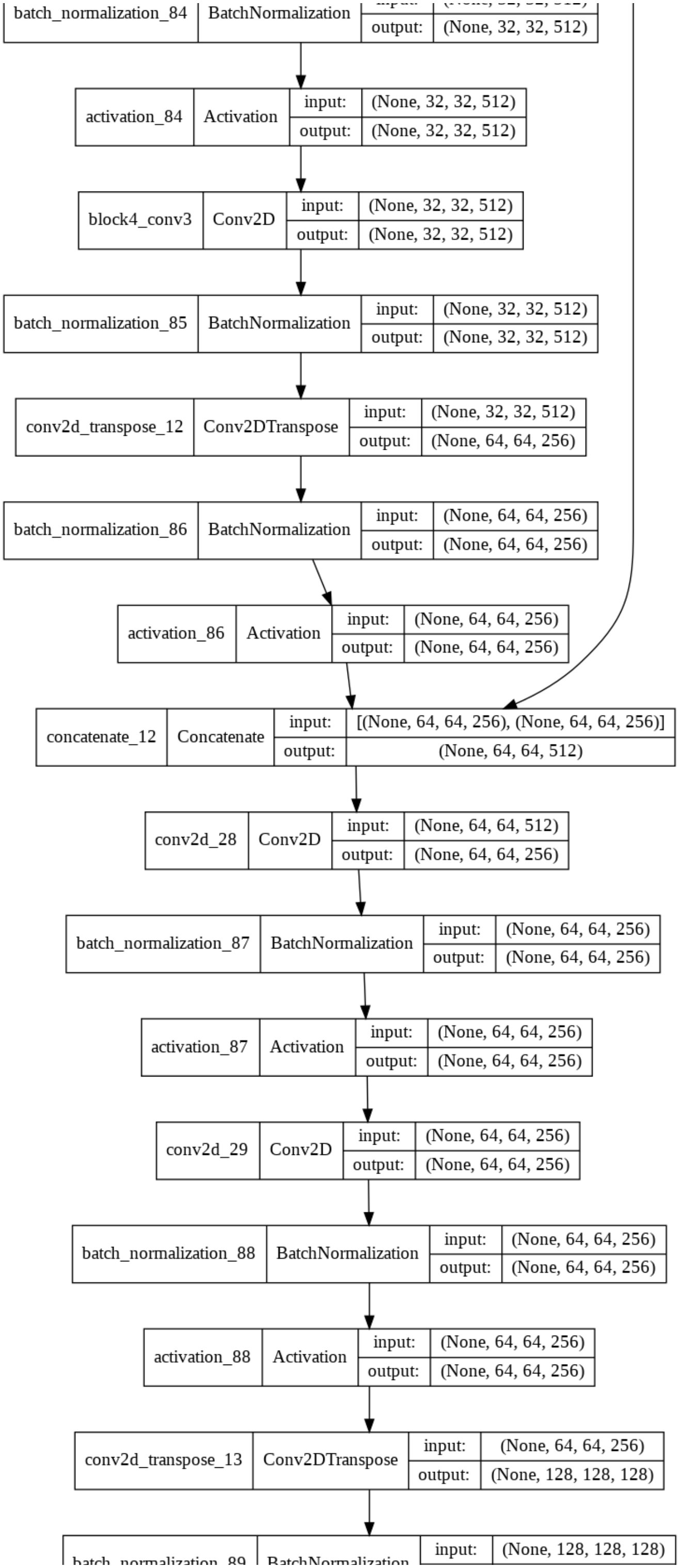
```
x = concatenate([x, block_3_out])
x = Conv2D(256, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(256, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# UP 2
x = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(x) #50x150
x = BatchNormalization()(x)
x = Activation('relu')(x)

# добавили перенос из понижающего плеча
x = concatenate([x, block_2_out])
x = Conv2D(128, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(128, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# UP 3
x = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(x) # 100x300
x = BatchNormalization()(x)
x = Activation('relu')(x)

# добавили перенос из понижающего плеча
x = concatenate([x, block_1_out])
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# # UP 4
# x = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(x) # 200x600
# x = BatchNormalization()(x)
# x = Activation('relu')(x)

# добавили перенос из понижающего плеча
# x = concatenate([x, block_1_out])
# x = Conv2D(64, (3, 3), padding='same')(x)
# x = BatchNormalization()(x)
# x = Activation('relu')(x)

# x = Conv2D(64, (3, 3), padding='same')(x)
# x = BatchNormalization()(x)
# x = Activation('relu')(x)

# слой классификатор (14 классов)
x = Conv2D(num_classes, (3, 3), activation='softmax', padding='same')(x)

model = Model(img_input, x)
# model.compile(optimizer=Adam(),
#               loss='categorical_crossentropy',
#               metrics=[dice_coef])
# model.summary()
return model
```

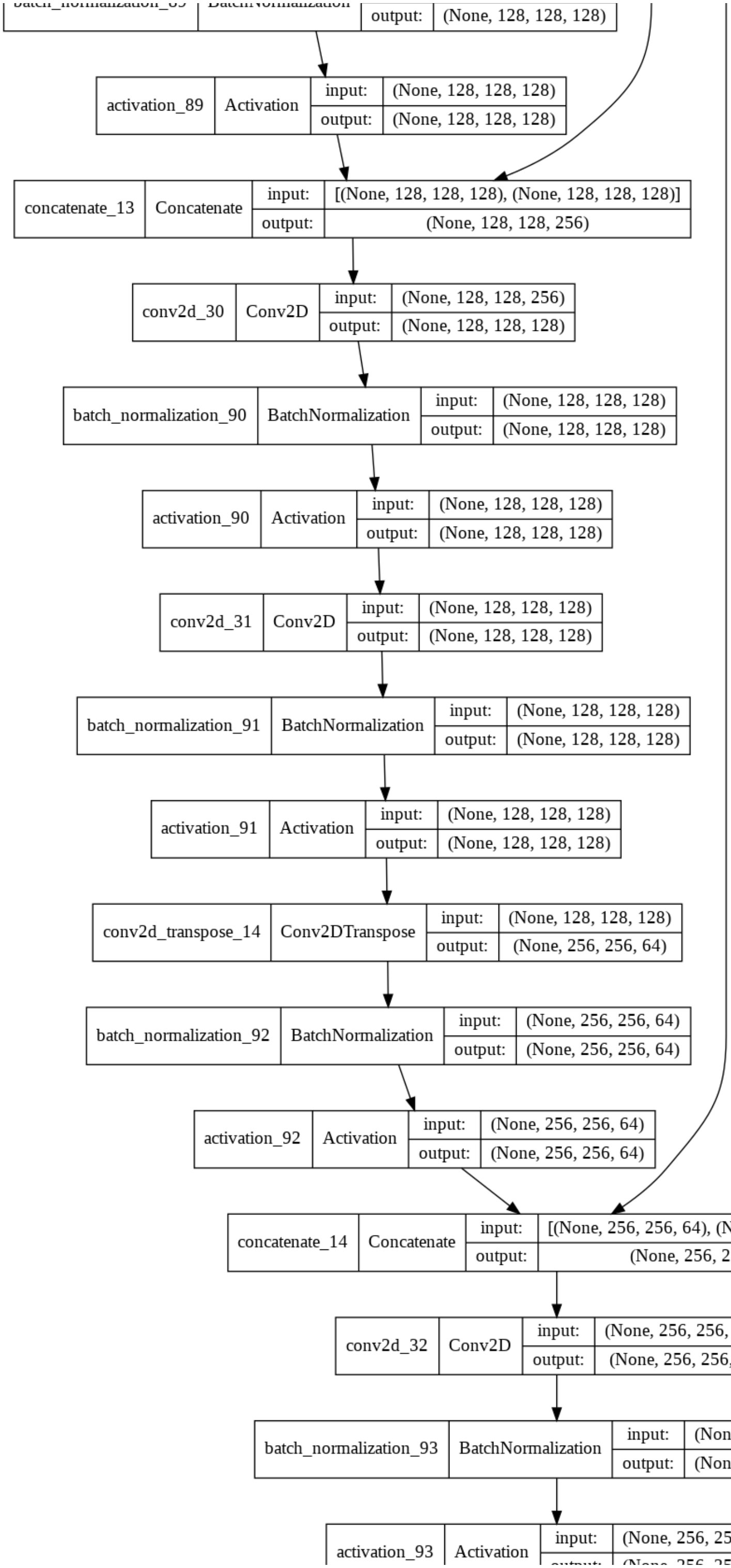
B [69]:
```
modelC = Unet(3, (256, 256, 3))

# plot_model(modelC, to_file='modelC.png')
plot_model(modelC, to_file='modelC.png', show_shapes=True)
```

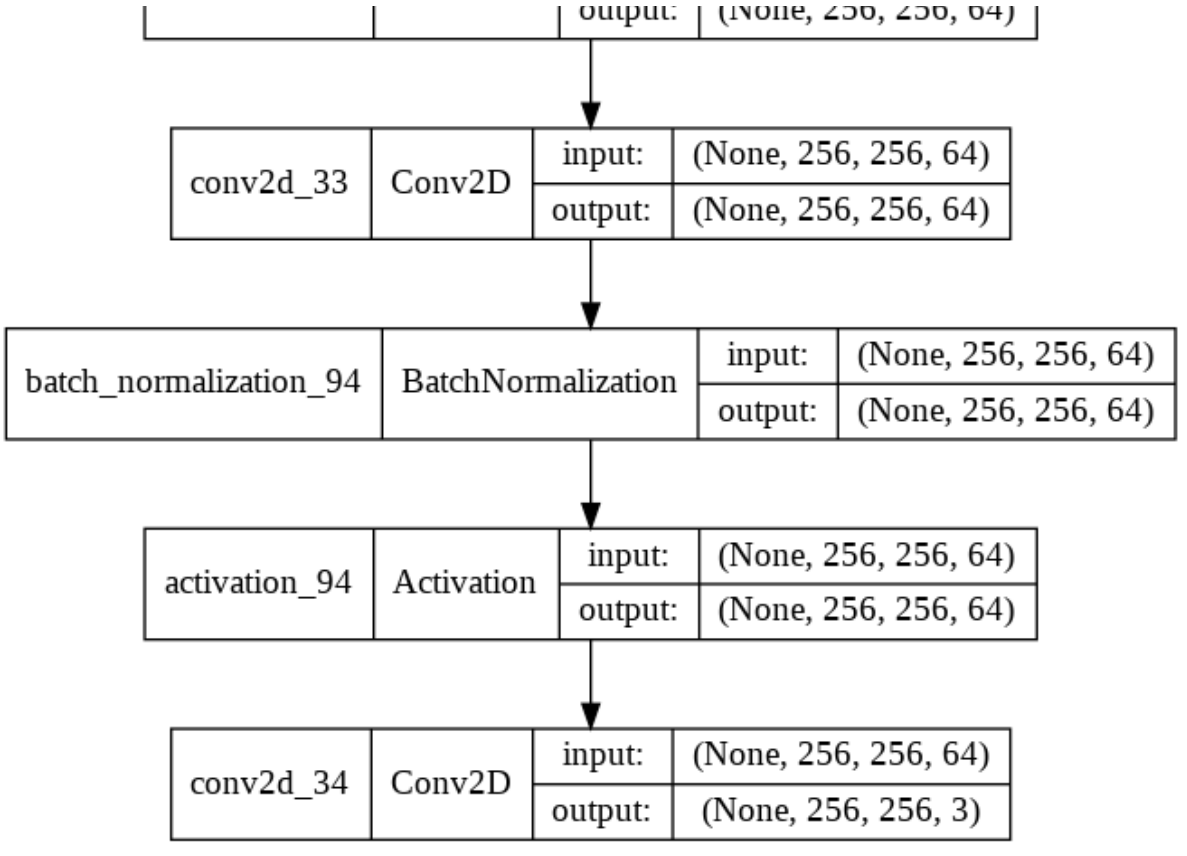
Out[69]:




max_pooling2d_13 MaxPooling2D input: (None, 128, 128, 128) output: (None, 64, 64, 128)
block3_conv1 Conv2D input: (None, 64, 64, 128) output: (None, 64, 64, 256)
batch_normalization_80 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_80 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
block3_conv2 Conv2D input: (None, 64, 64, 256) output: (None, 64, 64, 256)
batch_normalization_81 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_81 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
block3_conv3 Conv2D input: (None, 64, 64, 256) output: (None, 64, 64, 256)
batch_normalization_82 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_82 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
max_pooling2d_14 MaxPooling2D input: (None, 64, 64, 256) output: (None, 32, 32, 256)
block4_conv1 Conv2D input: (None, 32, 32, 256) output: (None, 32, 32, 512)
batch_normalization_83 BatchNormalization input: (None, 32, 32, 512) output: (None, 32, 32, 512)
activation_83 Activation input: (None, 32, 32, 512) output: (None, 32, 32, 512)
block4_conv2 Conv2D input: (None, 32, 32, 512) output: (None, 32, 32, 512)


batch_normalization_84 BatchNormalization output: (None, 32, 32, 512)
activation_84 Activation input: (None, 32, 32, 512) output: (None, 32, 32, 512)
block4_conv3 Conv2D input: (None, 32, 32, 512) output: (None, 32, 32, 512)
batch_normalization_85 BatchNormalization input: (None, 32, 32, 512) output: (None, 32, 32, 512)
conv2d_transpose_12 Conv2DTranspose input: (None, 32, 32, 512) output: (None, 64, 64, 256)
batch_normalization_86 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_86 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
concatenate_12 Concatenate input: [(None, 64, 64, 256), (None, 64, 64, 256)] output: (None, 64, 64, 512)
conv2d_28 Conv2D input: (None, 64, 64, 512) output: (None, 64, 64, 256)
batch_normalization_87 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_87 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
conv2d_29 Conv2D input: (None, 64, 64, 256) output: (None, 64, 64, 256)
batch_normalization_88 BatchNormalization input: (None, 64, 64, 256) output: (None, 64, 64, 256)
activation_88 Activation input: (None, 64, 64, 256) output: (None, 64, 64, 256)
conv2d_transpose_13 Conv2DTranspose input: (None, 64, 64, 256) output: (None, 128, 128, 128)
input: (None, 128, 128, 128)


output: (None, 128, 128, 128)
activation_89 Activation input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
concatenate_13 Concatenate input: [(None, 128, 128, 128), (None, 128, 128, 128)]
output: (None, 128, 128, 256)
conv2d_30 Conv2D input: (None, 128, 128, 256)
output: (None, 128, 128, 128)
batch_normalization_90 BatchNormalization input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
activation_90 Activation input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
conv2d_31 Conv2D input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
batch_normalization_91 BatchNormalization input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
activation_91 Activation input: (None, 128, 128, 128)
output: (None, 128, 128, 128)
conv2d_transpose_14 Conv2DTranspose input: (None, 128, 128, 128)
output: (None, 256, 256, 64)
batch_normalization_92 BatchNormalization input: (None, 256, 256, 64)
output: (None, 256, 256, 64)
activation_92 Activation input: (None, 256, 256, 64)
output: (None, 256, 256, 64)
concatenate_14 Concatenate input: [(None, 256, 256, 64), (None, 256, 256, 64)]
output: (None, 256, 256, 128)
conv2d_32 Conv2D input: (None, 256, 256, 128)
output: (None, 256, 256, 64)
batch_normalization_93 BatchNormalization input: (None, 256, 256, 64)
output: (None, 256, 256, 64)
activation_93 Activation input: (None, 256, 256, 64)


output: (None, 256, 256, 64)
conv2d_33 Conv2D input: (None, 256, 256, 64) output: (None, 256, 256, 64)
batch_normalization_94 BatchNormalization input: (None, 256, 256, 64) output: (None, 256, 256, 64)
activation_94 Activation input: (None, 256, 256, 64) output: (None, 256, 256, 64)
conv2d_34 Conv2D input: (None, 256, 256, 64) output: (None, 256, 256, 3)

B [70]:
```
modelC.compile(optimizer=Adam(),
               loss='categorical_crossentropy',
               metrics=[dice_coef])
modelC.summary()
```

```
Model: "model_4"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_5 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 256, 256, 64  1792        ['input_5[0][0]']
                                )

 batch_normalization_76 (BatchN  (None, 256, 256, 64  256        ['block1_conv1[0][0]']
 ormalization)                  )

 activation_76 (Activation)     (None, 256, 256, 64  0           ['batch_normalization_76[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 256, 256, 64  36928       ['activation_76[0][0]']
                                )

 batch_normalization_77 (BatchN  (None, 256, 256, 64  256        ['block1_conv2[0][0]']
 ormalization)                  )

 activation_77 (Activation)     (None, 256, 256, 64  0           ['batch_normalization_77[0][0]']
                                )

 max_pooling2d_12 (MaxPooling2D  (None, 128, 128, 64  0          ['activation_77[0][0]']
 )                              )

 block2_conv1 (Conv2D)          (None, 128, 128, 12  73856       ['max_pooling2d_12[0][0]']
                                8)

 batch_normalization_78 (BatchN  (None, 128, 128, 12  512        ['block2_conv1[0][0]']
 ormalization)                  8)

 activation_78 (Activation)     (None, 128, 128, 12  0           ['batch_normalization_78[0][0]']
                                8)

 block2_conv2 (Conv2D)          (None, 128, 128, 12  147584      ['activation_78[0][0]']
                                8)

 batch_normalization_79 (BatchN  (None, 128, 128, 12  512        ['block2_conv2[0][0]']
 ormalization)                  8)

 activation_79 (Activation)     (None, 128, 128, 12  0           ['batch_normalization_79[0][0]']
                                8)

 max_pooling2d_13 (MaxPooling2D  (None, 64, 64, 128)  0          ['activation_79[0][0]']
 )

 block3_conv1 (Conv2D)          (None, 64, 64, 256)  295168      ['max_pooling2d_13[0][0]']

 batch_normalization_80 (BatchN  (None, 64, 64, 256)  1024       ['block3_conv1[0][0]']
 ormalization)

 activation_80 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_80[0][0]']

 block3_conv2 (Conv2D)          (None, 64, 64, 256)  590080      ['activation_80[0][0]']

 batch_normalization_81 (BatchN  (None, 64, 64, 256)  1024       ['block3_conv2[0][0]']
 ormalization)

 activation_81 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_81[0][0]']

 block3_conv3 (Conv2D)          (None, 64, 64, 256)  590080      ['activation_81[0][0]']

 batch_normalization_82 (BatchN  (None, 64, 64, 256)  1024       ['block3_conv3[0][0]']
 ormalization)

 activation_82 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_82[0][0]']

 max_pooling2d_14 (MaxPooling2D  (None, 32, 32, 256)  0          ['activation_82[0][0]']
 )

 block4_conv1 (Conv2D)          (None, 32, 32, 512)  1180160     ['max_pooling2d_14[0][0]']

 batch_normalization_83 (BatchN  (None, 32, 32, 512)  2048       ['block4_conv1[0][0]']
 ormalization)

 activation_83 (Activation)     (None, 32, 32, 512)  0           ['batch_normalization_83[0][0]']

 block4_conv2 (Conv2D)          (None, 32, 32, 512)  2359808     ['activation_83[0][0]']

 batch_normalization_84 (BatchN  (None, 32, 32, 512)  2048       ['block4_conv2[0][0]']
```

```
ormalization)

activation_84 (Activation)     (None, 32, 32, 512)  0           ['batch_normalization_84[0][0]']

block4_conv3 (Conv2D)          (None, 32, 32, 512)  2359808     ['activation_84[0][0]']

batch_normalization_85 (BatchN  (None, 32, 32, 512)  2048       ['block4_conv3[0][0]']
ormalization)

conv2d_transpose_12 (Conv2DTra  (None, 64, 64, 256)  524544     ['batch_normalization_85[0][0]']
nspose)

batch_normalization_86 (BatchN  (None, 64, 64, 256)  1024       ['conv2d_transpose_12[0][0]']
ormalization)

activation_86 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_86[0][0]']

concatenate_12 (Concatenate)   (None, 64, 64, 512)  0           ['activation_86[0][0]',
                                                                 'activation_82[0][0]']

conv2d_28 (Conv2D)             (None, 64, 64, 256)  1179904     ['concatenate_12[0][0]']

batch_normalization_87 (BatchN  (None, 64, 64, 256)  1024       ['conv2d_28[0][0]']
ormalization)

activation_87 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_87[0][0]']

conv2d_29 (Conv2D)             (None, 64, 64, 256)  590080      ['activation_87[0][0]']

batch_normalization_88 (BatchN  (None, 64, 64, 256)  1024       ['conv2d_29[0][0]']
ormalization)

activation_88 (Activation)     (None, 64, 64, 256)  0           ['batch_normalization_88[0][0]']

conv2d_transpose_13 (Conv2DTra  (None, 128, 128, 12  131200     ['activation_88[0][0]']
nspose)                        8)

batch_normalization_89 (BatchN  (None, 128, 128, 12  512        ['conv2d_transpose_13[0][0]']
ormalization)                  8)

activation_89 (Activation)     (None, 128, 128, 12  0           ['batch_normalization_89[0][0]']
                               8)

concatenate_13 (Concatenate)   (None, 128, 128, 25  0           ['activation_89[0][0]',
                               6)                                'activation_79[0][0]']

conv2d_30 (Conv2D)             (None, 128, 128, 12  295040      ['concatenate_13[0][0]']
                               8)

batch_normalization_90 (BatchN  (None, 128, 128, 12  512        ['conv2d_30[0][0]']
ormalization)                  8)

activation_90 (Activation)     (None, 128, 128, 12  0           ['batch_normalization_90[0][0]']
                               8)

conv2d_31 (Conv2D)             (None, 128, 128, 12  147584      ['activation_90[0][0]']
                               8)

batch_normalization_91 (BatchN  (None, 128, 128, 12  512        ['conv2d_31[0][0]']
ormalization)                  8)

activation_91 (Activation)     (None, 128, 128, 12  0           ['batch_normalization_91[0][0]']
                               8)

conv2d_transpose_14 (Conv2DTra  (None, 256, 256, 64  32832      ['activation_91[0][0]']
nspose)                        )

batch_normalization_92 (BatchN  (None, 256, 256, 64  256        ['conv2d_transpose_14[0][0]']
ormalization)                  )

activation_92 (Activation)     (None, 256, 256, 64  0           ['batch_normalization_92[0][0]']
                               )

concatenate_14 (Concatenate)   (None, 256, 256, 12  0           ['activation_92[0][0]',
                               8)                                'activation_77[0][0]']

conv2d_32 (Conv2D)             (None, 256, 256, 64  73792       ['concatenate_14[0][0]']
                               )

batch_normalization_93 (BatchN  (None, 256, 256, 64  256        ['conv2d_32[0][0]']
ormalization)                  )

activation_93 (Activation)     (None, 256, 256, 64  0           ['batch_normalization_93[0][0]']
                               )

conv2d_33 (Conv2D)             (None, 256, 256, 64  36928       ['activation_93[0][0]']
                               )
```

```
 batch_normalization_94 (BatchN  (None, 256, 256, 64  256        ['conv2d_33[0][0]']
 ormalization)                  )

 activation_94 (Activation)     (None, 256, 256, 64  0          ['batch_normalization_94[0][0]']
                                )

 conv2d_34 (Conv2D)             (None, 256, 256, 3)  1731       ['activation_94[0][0]']

==================================================================================================
Total params: 10,665,027
Trainable params: 10,656,963
Non-trainable params: 8,064
__________________________________________________________________________________________________
```

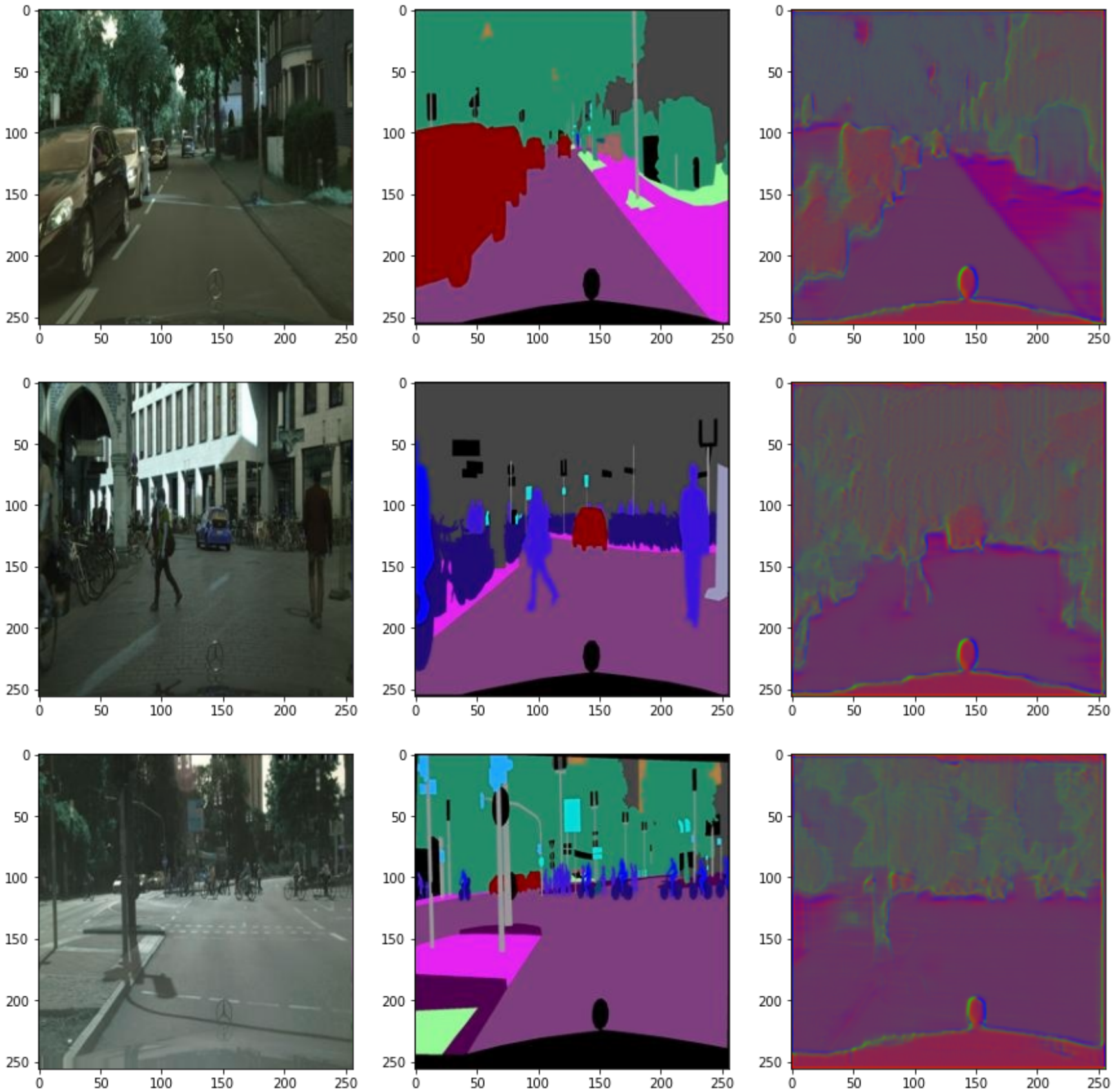
B [71]:
```
history = modelC.fit(x_train, y_train,
                     epochs=25,
                     batch_size=10,
                     validation_data=(x_val, y_val)
                     )
```

```
Epoch 1/25
50/50 [==============================] - 31s 575ms/step - loss: 1.1620 - dice_coef: 0.3618 - val_loss: 6.1119 - val_dic
e_coef: 0.3513
Epoch 2/25
50/50 [==============================] - 29s 581ms/step - loss: 1.1372 - dice_coef: 0.3579 - val_loss: 1.1457 - val_dic
e_coef: 0.3477
Epoch 3/25
50/50 [==============================] - 29s 588ms/step - loss: 1.1363 - dice_coef: 0.3592 - val_loss: 1.3210 - val_dic
e_coef: 0.3589
Epoch 4/25
50/50 [==============================] - 30s 594ms/step - loss: 1.1366 - dice_coef: 0.3600 - val_loss: 1.2466 - val_dic
e_coef: 0.3580
Epoch 5/25
50/50 [==============================] - 30s 597ms/step - loss: 1.1389 - dice_coef: 0.3602 - val_loss: 1.1803 - val_dic
e_coef: 0.3497
Epoch 6/25
50/50 [==============================] - 30s 593ms/step - loss: 1.1385 - dice_coef: 0.3604 - val_loss: 1.3006 - val_dic
e_coef: 0.3625
Epoch 7/25
50/50 [==============================] - 30s 603ms/step - loss: 1.1399 - dice_coef: 0.3617 - val_loss: 1.2298 - val_dic
e_coef: 0.3686
Epoch 8/25
50/50 [==============================] - 30s 598ms/step - loss: 1.1423 - dice_coef: 0.3615 - val_loss: 1.2431 - val_dic
e_coef: 0.3697
Epoch 9/25
50/50 [==============================] - 30s 608ms/step - loss: 1.1411 - dice_coef: 0.3625 - val_loss: 1.1713 - val_dic
e_coef: 0.3592
Epoch 10/25
50/50 [==============================] - 30s 611ms/step - loss: 1.1436 - dice_coef: 0.3630 - val_loss: 1.1450 - val_dic
e_coef: 0.3575
Epoch 11/25
50/50 [==============================] - 30s 604ms/step - loss: 1.1469 - dice_coef: 0.3636 - val_loss: 1.1990 - val_dic
e_coef: 0.3700
Epoch 12/25
50/50 [==============================] - 30s 607ms/step - loss: 1.1466 - dice_coef: 0.3628 - val_loss: 1.2173 - val_dic
e_coef: 0.3678
Epoch 13/25
50/50 [==============================] - 30s 606ms/step - loss: 1.1522 - dice_coef: 0.3637 - val_loss: 1.1565 - val_dic
e_coef: 0.3638
Epoch 14/25
50/50 [==============================] - 30s 610ms/step - loss: 1.1490 - dice_coef: 0.3642 - val_loss: 1.2008 - val_dic
e_coef: 0.3655
Epoch 15/25
50/50 [==============================] - 31s 612ms/step - loss: 1.1470 - dice_coef: 0.3640 - val_loss: 1.2415 - val_dic
e_coef: 0.3734
Epoch 16/25
50/50 [==============================] - 30s 608ms/step - loss: 1.1512 - dice_coef: 0.3650 - val_loss: 1.1930 - val_dic
e_coef: 0.3674
Epoch 17/25
50/50 [==============================] - 31s 614ms/step - loss: 1.1523 - dice_coef: 0.3634 - val_loss: 1.1790 - val_dic
e_coef: 0.3695
Epoch 18/25
50/50 [==============================] - 30s 608ms/step - loss: 1.1491 - dice_coef: 0.3637 - val_loss: 1.1745 - val_dic
e_coef: 0.3606
Epoch 19/25
50/50 [==============================] - 30s 607ms/step - loss: 1.1538 - dice_coef: 0.3642 - val_loss: 1.1673 - val_dic
e_coef: 0.3662
Epoch 20/25
50/50 [==============================] - 30s 607ms/step - loss: 1.1489 - dice_coef: 0.3638 - val_loss: 1.1597 - val_dic
e_coef: 0.3621
Epoch 21/25
50/50 [==============================] - 31s 614ms/step - loss: 1.1573 - dice_coef: 0.3649 - val_loss: 1.1269 - val_dic
e_coef: 0.3570
Epoch 22/25
50/50 [==============================] - 30s 608ms/step - loss: 1.1505 - dice_coef: 0.3643 - val_loss: 1.1807 - val_dic
e_coef: 0.3682
Epoch 23/25
50/50 [==============================] - 31s 614ms/step - loss: 1.1503 - dice_coef: 0.3644 - val_loss: 1.1388 - val_dic
e_coef: 0.3650
Epoch 24/25
50/50 [==============================] - 30s 608ms/step - loss: 1.1524 - dice_coef: 0.3643 - val_loss: 1.1687 - val_dic
e_coef: 0.3641
Epoch 25/25
50/50 [==============================] - 30s 607ms/step - loss: 1.1564 - dice_coef: 0.3645 - val_loss: 1.1425 - val_dic
e_coef: 0.3629
```

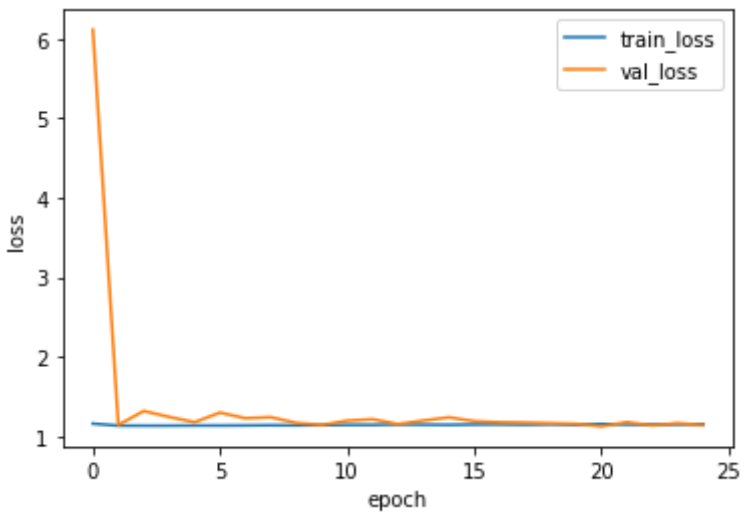
B [72]:
```
pred = modelC.predict(x_val)
print(pred.shape)
```

```
(50, 256, 256, 3)
```

B [73]:
```
n = 3
for k in range(n):
    plt.figure(figsize=(20, 60))

    plt.subplot(n, 4, 1 + k*4)
    plt.imshow(x_val[k])

    plt.subplot(n, 4, 2 + k*4)
    plt.imshow(y_val[k])

    plt.subplot(n, 4, 3 + k*4)
    plt.imshow(pred[k])
```

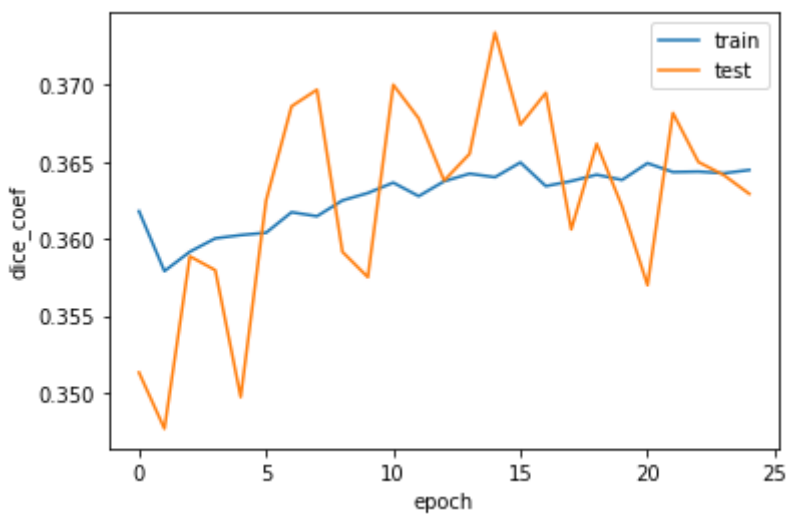


B [74]:
```
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show();
```



После 5 эпохи потери на тестовой выборке практически не меняется.

B [75]:
```
plt.plot(history.history['dice_coef'], label='train')
plt.plot(history.history['val_dice_coef'], label='test')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('dice_coef')
plt.show()
```



Как на тестовой, так и на тренировочной выборках изменение dice_coef с увеличением числа эпох меняется незначительно.

Вывод:

Факторами влияющими на точность сети являются:

- число эпох;
- количество ядер в слоях Conv2D (с увеличением числа ядер, растёт точность);
- количество классов рассматриваемых объектов;
- batch_size.

Jсновной фактор число эпох.

B [75]: