Deep Learning School

Курс "Глубокое обучение". Первый семестр

Домашнее задание. Сегментация изображений

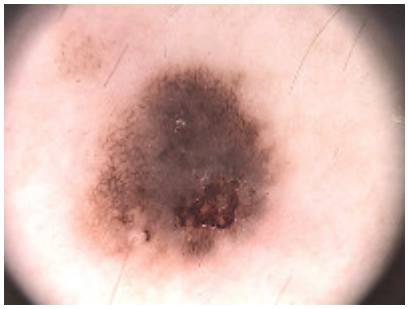
Шерешевский Дмитрий, ID 36196483

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive/')
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call drive.mo
unt("/content/gdrive/", force_remount=True).
```

In [2]:
```
# import os
# os.chdir("gdrive/MyDrive/Colab Notebooks/DLSchool_mipt_1sem/segmentation/")
data_path = "gdrive/MyDrive/Colab Notebooks/DLSchool_mipt_1sem/segmentation/"
```

1.Для начала мы скачаем датасет: ADDI project.




1. Разархивируем .rar файл.
2. Обратите внимание, что папка `PH2 Dataset images` должна лежать там же где и ipynb notebook.

Это фотографии двух типов **поражений кожи:** меланома и родинки. В данном задании мы не будем заниматься их классификацией, а будем сегментировать их.

In [3]:
```
! wget https://www.dropbox.com/s/k88qukc20ljnbuo/PH2Dataset.rar
```

```
--2020-12-21 07:17:29--  https://www.dropbox.com/s/k88qukc20ljnbuo/PH2Dataset.rar
Resolving www.dropbox.com (www.dropbox.com)... 162.125.4.18, 2620:100:6019:18::a27d:412
Connecting to www.dropbox.com (www.dropbox.com)|162.125.4.18|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/k88qukc20ljnbuo/PH2Dataset.rar [following]
--2020-12-21 07:17:29--  https://www.dropbox.com/s/raw/k88qukc20ljnbuo/PH2Dataset.rar
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com/cd/0/inline/BFe
CY-b1hvbgYPdcH2_RMJPujRQ-V3hq5u5nCqob1-Rn3YIN3gNOGYnIbLVQGHHMCSXxTd-57XtxBe1uens2XyiXPKM
0iL5Df8kDwUpcEUMxUQ/file# [following]
--2020-12-21 07:17:29--  https://uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com/
cd/0/inline/BFeCY-b1hvbgYPdcH2_RMJPujRQ-V3hq5u5nCqob1-Rn3YIN3gNOGYnIbLVQGHHMCSXxTd-57Xtx
Be1uens2XyiXPKM0iL5Df8kDwUpcEUMxUQ/file
Resolving uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com (uc060a877a8a4b0b27268b
c59e62.dl.dropboxusercontent.com)... 162.125.4.15, 2620:100:6019:15::a27d:40f
Connecting to uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com (uc060a877a8a4b0b27
268bc59e62.dl.dropboxusercontent.com)|162.125.4.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/BFdpOUZxmFVLxs4Hpon8PUTpoB3GEahX9YmE45v285uc0ZKKVvFBq4Fy9AFzYass
rVP-FY3F3_sOI9H-i-maCD6p-8_3T64X_eyxGydJCTb9lMZ3pVhUpzSGLxmGtyMK92m-oorFLutjfBWYSvDhaBKw
-9aY-l906WKnvrd9WyamPQcPR0X5zpyw_d03jlwu9DdaEY8Ox3oa11weEWX3b_JX_FcphvVHsTyYJCAWs1YqRoQU
KMlvC1lD0xpGYax1wJcMbw97J0e0Nxj1o9xv21g45lg5cDg3uU3OkiP3FXeatxO41hxtbjzNhYumQwfU3sFE-mXW
YnW1-BhRTFdvDaok/file [following]
--2020-12-21 07:17:30--  https://uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com/
cd/0/inline2/BFdpOUZxmFVLxs4Hpon8PUTpoB3GEahX9YmE45v285uc0ZKKVvFBq4Fy9AFzYassrVP-FY3F3_s
OI9H-i-maCD6p-8_3T64X_eyxGydJCTb9lMZ3pVhUpzSGLxmGtyMK92m-oorFLutjfBWYSvDhaBKw-9aY-l906WK
nvrd9WyamPQcPR0X5zpyw_d03jlwu9DdaEY8Ox3oa11weEWX3b_JX_FcphvVHsTyYJCAWs1YqRoQUKMlvC1lD0xp
GYax1wJcMbw97J0e0Nxj1o9xv21g45lg5cDg3uU3OkiP3FXeatxO41hxtbjzNhYumQwfU3sFE-mXWYnW1-BhRTFd
vDaok/file
Reusing existing connection to uc060a877a8a4b0b27268bc59e62.dl.dropboxusercontent.com:44
3.
HTTP request sent, awaiting response... 200 OK
Length: 116457882 (111M) [application/rar]
Saving to: ‘PH2Dataset.rar.4’

PH2Dataset.rar.4    100%[===================>] 111.06M   121MB/s    in 0.9s

2020-12-21 07:17:32 (121 MB/s) - ‘PH2Dataset.rar.4’ saved [116457882/116457882]
```

In [4]:
```
get_ipython().system_raw("unrar x PH2Dataset.rar")
```

Стуктура датасета у нас следующая:

```
IMD_002/
    IMD002_Dermoscopic_Image/
        IMD002.bmp
    IMD002_lesion/
        IMD002_lesion.bmp
    IMD002_roi/
        ...
IMD_003/
    ...
    ...
```

Для загрузки датасета я предлагаю использовать skimage: `skimage.io.imread()`

In [5]:
```
images = []
lesions = []
from skimage.io import imread
import os
root = 'PH2Dataset'

for root, dirs, files in os.walk(os.path.join(root, 'PH2 Dataset images')):
    if root.endswith('_Dermoscopic_Image'):
        images.append(imread(os.path.join(root, files[0])))
    if root.endswith('_lesion'):
        lesions.append(imread(os.path.join(root, files[0])))
```

Изображения имеют разные размеры. Давайте изменим их размер на $256\times256 $ пикселей. `skimage.transform.resize()` можно использовать для изменения размера изображений. Эта функция также автоматически нормализует изображения в диапазоне $[0,1]$.

In [6]:
```
from skimage.transform import resize
size = (256, 256)
X = [resize(x, size, mode='constant', anti_aliasing=True,) for x in images]
Y = [resize(y, size, mode='constant', anti_aliasing=False) > 0.5 for y in lesions]
```

In [7]:
```
import numpy as np
X = np.array(X, np.float32)
Y = np.array(Y, np.float32)
print(f'Loaded {len(X)} images')
```

```
Loaded 200 images
```

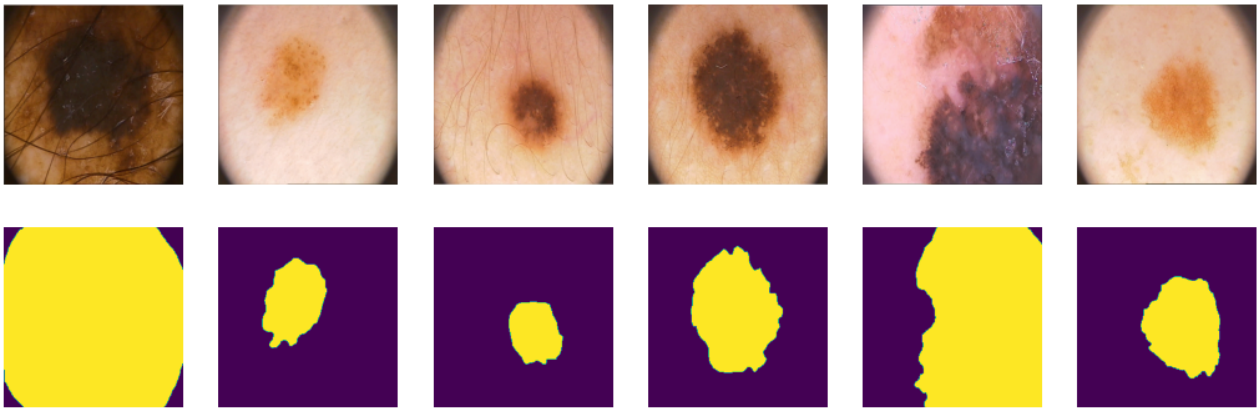
In [8]:
```
len(lesions)
```

Out[8]:
```
200
```

Чтобы убедиться, что все корректно, мы нарисуем несколько изображений

In [9]:
```
import matplotlib.pyplot as plt
from IPython.display import clear_output

plt.figure(figsize=(18, 6))
for i in range(6):
```

```
    plt.subplot(2, 6, i+1)
    plt.axis("off")
    plt.imshow(X[i])

    plt.subplot(2, 6, i+7)
    plt.axis("off")
    plt.imshow(Y[i])
plt.show();
```



Разделим наши 200 картинок на 100/50/50 для валидации и теста

In [10]:
```
ix = np.random.choice(len(X), len(X), False)
tr, val, ts = np.split(ix, [100, 150])
```

In [11]:
```
print(len(tr), len(val), len(ts))
```

```
100 50 50
```

PyTorch DataLoader

In [12]:
```
from torch.utils.data import DataLoader
batch_size = 25
data_tr = DataLoader(list(zip(np.rollaxis(X[tr], 3, 1), Y[tr, np.newaxis])),
                     batch_size=batch_size, shuffle=True)
data_val = DataLoader(list(zip(np.rollaxis(X[val], 3, 1), Y[val, np.newaxis])),
                      batch_size=batch_size, shuffle=True)
data_ts = DataLoader(list(zip(np.rollaxis(X[ts], 3, 1), Y[ts, np.newaxis])),
                     batch_size=batch_size, shuffle=True)
```

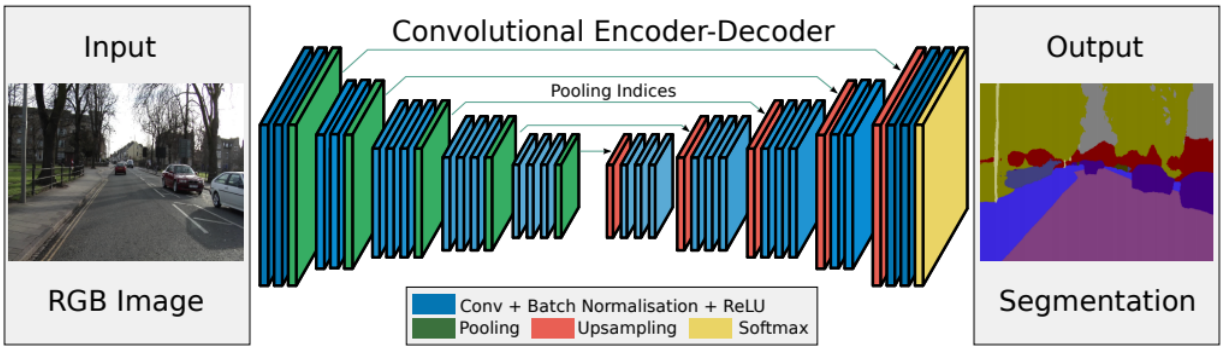
In [13]:
```
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

Реализация различных архитектур:

Ваше задание будет состоять в том, чтобы написать несколько нейросетевых архитектур для решения задачи семантической сегментации. Сравнить их по качеству на тесте и испробовать различные лосс функции для них.

SegNet [2 балла]



- Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). SegNet: A deep convolutional encoder-decoder architecture for image segmentation

Внимательно посмотрите из чего состоит модель и для чего выбраны те или иные блоки.

In [14]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import torch.optim as optim
from time import time
import pickle
import os
import pandas as pd

from matplotlib import rcParams
rcParams['figure.figsize'] = (15,4)
```

In [15]:
```
class SegNet(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        self.enc_conv0 = nn.Sequential(
                            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU()
                          )
        self.pool0 = torch.nn.MaxPool2d(2, 2, return_indices=True)   # 256 -> 128
        self.enc_conv1 = nn.Sequential(
                            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU()
                          )
        self.pool1 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 128 -> 64
        self.enc_conv2 = nn.Sequential(
                            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
```

```
                        nn.BatchNorm2d(256),
                        nn.ReLU(),
                        nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(),
                        nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU()
                    )
        self.pool2 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 64 -> 32
        self.enc_conv3 = nn.Sequential(
                        nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU()
                    )
        self.pool3 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 32 -> 16
        # bottleneck
        self.bottleneck_conv = nn.Sequential(
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        # pool
                        nn.Conv2d(512, 512, kernel_size=2, stride=2),
                        # upsampl
                        nn.ConvTranspose2d(512, 512, kernel_size=2, stride=2),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(),
                        nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU()
                    )
        # decoder (upsampling)
        self.upsample0 = torch.nn.MaxUnpool2d(2, 2) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
                        nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(),
                        nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(),
                        nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU()
                    )
        self.upsample1 = torch.nn.MaxUnpool2d(2, 2)  # 32 -> 64
```

```
        self.dec_conv1 = nn.Sequential(
                            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU()
                        )
        self.upsample2 = torch.nn.MaxUnpool2d(2, 2)   # 64 -> 128
        self.dec_conv2 = nn.Sequential(
                            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU()
                        )
        self.upsample3 = torch.nn.MaxUnpool2d(2, 2)   # 128 -> 256
        self.dec_conv3 = nn.Sequential(
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 1, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(1),
                            nn.ReLU()
                        )

    def forward(self, x):
        # encoder
        e0, indices0 = self.pool0(self.enc_conv0(x))
        e1, indices1 = self.pool1(self.enc_conv1(e0))
        e2, indices2 = self.pool2(self.enc_conv2(e1))
        e3, indices3 = self.pool3(self.enc_conv3(e2))

        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        d0 = self.dec_conv0(self.upsample0(b, indices3))
        d1 = self.dec_conv1(self.upsample1(d0, indices2))
        d2 = self.dec_conv2(self.upsample2(d1, indices1))
        d3 = self.dec_conv3(self.upsample3(d2, indices0))  # no activation
        return d3
```

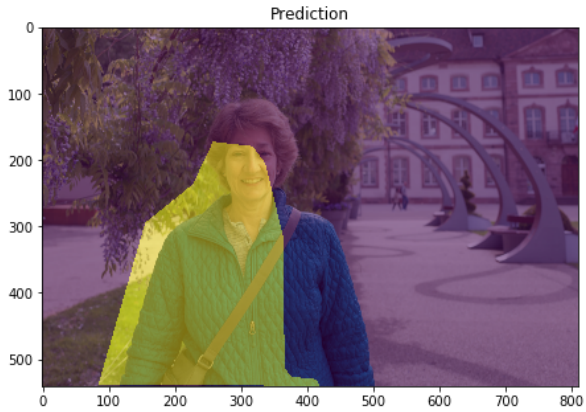
Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

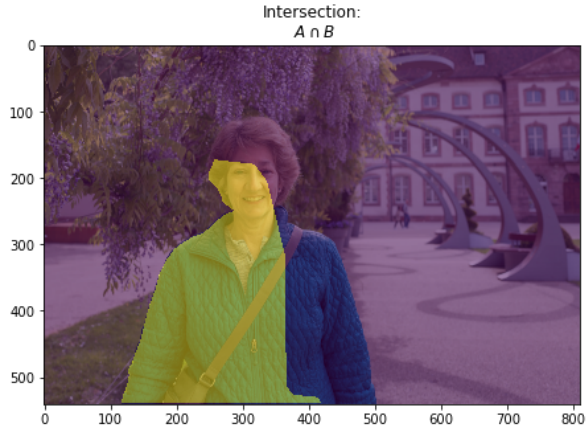
$I o U=\frac{\text {target } \cap \text { prediction }}{\text {target } \cup{prediction }}$

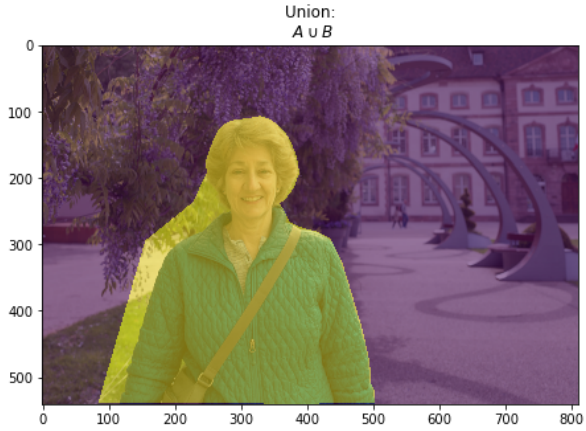
Пересечение (A ∩ B) состоит из пикселей, найденных как в маске предсказания, так и в основной маске истины, тогда как объединение (A ∪ B) просто состоит из всех пикселей, найденных либо в маске предсказания, либо в целевой маске.

To clarify this we can see on the segmentation:



And the intersection will be the following:





In [16]:
```python
def iou_pytorch(outputs: torch.Tensor, labels: torch.Tensor):
    # You can comment out this line if you are passing tensors of equal shape
    # But if you are passing output from UNet or something it will most probably
    # be with the BATCH x 1 x H x W shape
    outputs = outputs.squeeze(1).byte()  # BATCH x 1 x H x W => BATCH x H x W
    labels = labels.squeeze(1).byte()
    SMOOTH = 1e-8
    intersection = (outputs & labels).float().sum((1, 2))  # Will be zero if Truth=0 or
    union = (outputs | labels).float().sum((1, 2))         # Will be zzero if both are

    iou = (intersection + SMOOTH) / (union + SMOOTH)  # We smooth our devision to avoid

    thresholded = torch.clamp(20 * (iou - 0.5), 0, 10).ceil() / 10  # This is equal to

    return iou # thresholded  #
```

Функция потерь [1 балл]

Теперь не менее важным, чем построение архитектуры, является определение **оптимизатора** и **функции потерь.**

Функция потерь - это то, что мы пытаемся минимизировать. Многие из них могут быть использованы для задачи бинарной семантической сегментации.

Популярным методом для бинарной сегментации является *бинарная кросс-энтропия*, которая задается следующим образом:

$$\mathcal L_{BCE}(y, \hat y) = -\sum_i \left[y_i\log\sigma(\hat y_i) + (1-y_i)\log(1-\sigma(\hat y_i))\right].$$
где y это таргет желаемого результата и $\hat y$ является выходом модели. σ - это *логистическая* функция, который преобразует действительное число $\mathbb R$ в вероятность $[0,1]$.

Однако эта потеря страдает от проблем численной нестабильности. Самое главное, что $\lim_{x\rightarrow0}\log(x)=\infty$ приводит к неустойчивости в процессе оптимизации. Рекомендуется посмотреть следующее упрощение в Тарая функция эквивалентна и не так подвержена численной неустойчивости.

$$\mathcal L_{BCE} = \hat y - y\hat y + \log\left(1+\exp(-\hat y)\right).$$

In [17]:
```
def bce_loss(y_real, y_pred):
    # TODO
    # please don't use nn.BCELoss. write it from scratch
    return torch.mean(y_pred - y_real * y_pred + torch.log(1 + torch.exp(-y_pred)))
```

Тренировка [1 балл]

Мы определим цикл обучения в функции, чтобы мы могли повторно использовать его.

In [18]:
```
def train(model, opt, loss_fn, epochs, data_tr, data_val, scheduler=None):
    X_val, Y_val = next(iter(data_val))

    loss_history = []
    for epoch in range(epochs):
        tic = time()
        print('* Epoch %d/%d' % (epoch+1, epochs))

        avg_loss = 0
        model.train()  # train mode
        for X_batch, Y_batch in data_tr:
            # data to device
            X_batch = X_batch.to(device)
            Y_batch = Y_batch.to(device)

            # set parameter gradients to zero
            opt.zero_grad()

            # forward
            Y_pred = model(X_batch)
            loss = loss_fn(Y_batch, Y_pred) # forward-pass
            loss.backward()  # backward-pass
            opt.step()  # update weights
            if scheduler:
                scheduler.step()

            # calculate loss to show the user
            avg_loss += loss / len(data_tr)
        if scheduler:
                scheduler.step()
        toc = time()
        print('loss: %f' % avg_loss)
        loss_history.append(avg_loss)
```

```
        # show intermediate results
        model.eval()  # testing mode
        Y_hat = model(X_val.to(device)).detach().cpu().numpy() # detach and put into cp

        # Visualize tools
        clear_output(wait=True)
        for k in range(6):
            plt.subplot(3, 6, k+1)
            plt.imshow(np.rollaxis(X_val[k].numpy(), 0, 3), cmap='gray')
            plt.title('Real')
            plt.axis('off')

            plt.subplot(3, 6, k+7)
            plt.imshow(Y_hat[k, 0], cmap='gray')
            plt.title('Output')
            plt.axis('off')

            plt.subplot(3, 6, k+13)
            plt.imshow(Y_hat[k, 0] > 0.5, cmap='gray')
            plt.title('Mask')
            plt.axis('off')
        plt.suptitle('%d / %d - loss: %f' % (epoch+1, epochs, avg_loss))
        plt.show()
    return loss_history
```

Инференс [1 балл]

После обучения модели эту функцию можно использовать для прогнозирования сегментации на новых данных:

In [19]:
```
def predict(model, data):
    model.eval()  # testing mode
    Y_pred = [model(X_batch.to(device)).detach().cpu().numpy() for X_batch, _ in data]
    return np.array(Y_pred)
```

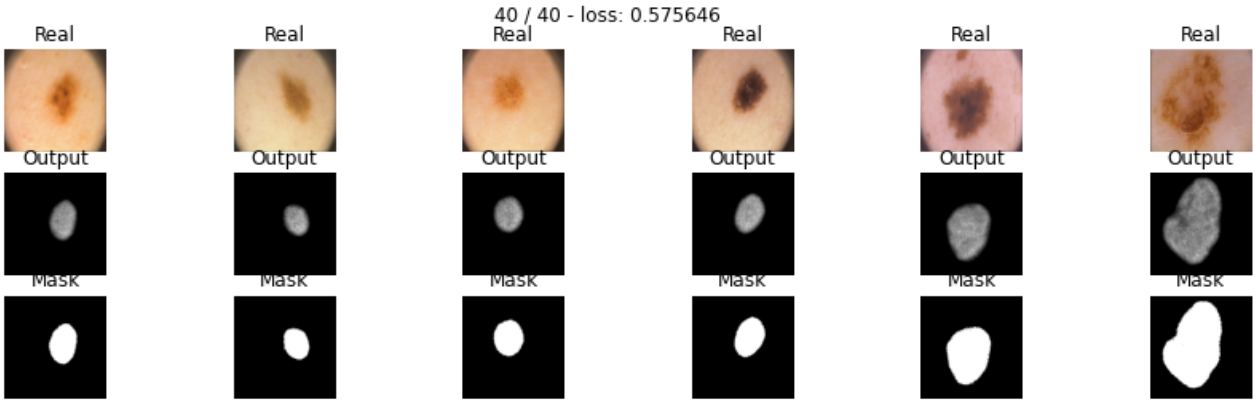
In [20]:
```
def score_model(model, metric, data):
    model.eval()  # testing mode
    scores = 0
    for X_batch, Y_label in data:
        Y_pred = torch.sigmoid(model(X_batch.to(device))) > 0.5  # was added sigmoid an
        scores += metric(Y_pred, Y_label.to(device)).mean().item()

    return scores/len(data)
```

Основной момент: обучение

Обучите вашу модель. Обратите внимание, что обучать необходимо до сходимости. Если указанного количества эпох (20) не хватило, попробуйте изменять количество эпох до сходимости алгоритма. Сходимость определяйте по изменению функции потерь на валидационной выборке. С параметрами оптимизатора можно спокойно играть, пока вы не найдете лучший вариант для себя.

In [21]:
```
model = SegNet().to(device)
```

In [22]:
```
max_epochs = 40
optimizer = optim.AdamW(model.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=15, gamma=0.5)
history = train(model, optimizer, bce_loss, max_epochs, data_tr, data_val, scheduler)
```



In [24]:
```
score = score_model(model, iou_pytorch, data_val)
score
```

Out[24]:
```
0.6612666845321655
```

In [27]:
```
# logging
name = "segnet_bce"
model_data = {
    "name": name,
    "history": history,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

In [34]:
```
# test
with open(os.path.join(data_path, name + ".pickle"), 'rb') as file:
    model_data_ = pickle.load(file)

assert model_data == model_data_
```

Ответьте себе на вопрос: не переобучается ли моя модель?

Дополнительные функции потерь [2 балла]

В данном разделе вам потребуется имплементировать две функции потерь: DICE и Focal loss. Если у вас что-то не учится, велика вероятность, что вы ошиблись или учите слишком мало, прежде чем бить тревогу попробуйте поперебирать различные варианты, убедитесь, что во всех других сетапах сетть достигает желанного результата. СПОЙЛЕР: учиться она будеет при всех лоссах предложенных в этом задании.

1. Dice coefficient: Учитывая две маски X и Y, общая метрика для измерения расстояния между этими двумя масками задается следующим образом:

$$D(X,Y)=\frac{2|X\cap Y|}{|X|+|Y|}$$

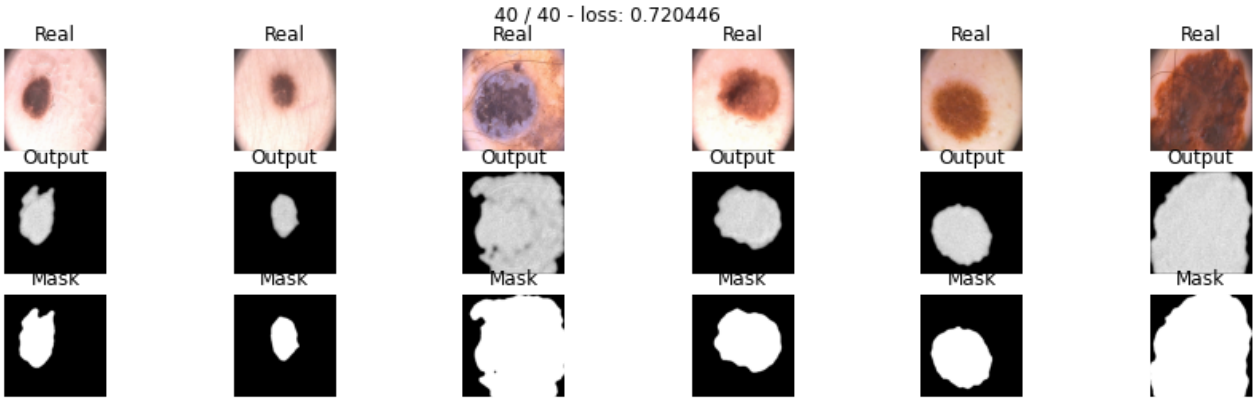
Эта функция не является дифференцируемой, но это необходимое свойство для градиентного спуска. В данном случае мы можем приблизить его с помощью:

$$\mathcal L_D(X,Y) = 1-\frac{1}{256 \times 256} \times \sum_i\frac{2X_iY_i}{X_i+Y_i}.$$
Не забудьте подумать о численной нестабильности, возникаемой в математической формуле.

In [36]:
```
def dice_loss(y_real, y_pred, eps = 1e-8):
    #num =
    #den =
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps)
    res = 1 - torch.mean(2 * (y_real * y_pred) / ((y_real + y_pred)))
    return res
```

Проводим тестирование:

In [37]:
```
model_dice = SegNet().to(device)

max_epochs = 40
optimizer = optim.AdamW(model_dice.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
history_dice = train(model_dice, optimizer, dice_loss, max_epochs, data_tr, data_val, s
```



In [38]:
```
score = score_model(model_dice, iou_pytorch, data_val)
score
```

Out[38]:
```
0.8040338158607483
```

In [39]:
```
# Logging
name = "segnet_dice"
model_data = {
    "name": name,
    "history": history_dice,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

2. Focal loss:

Окей, мы уже с вами умеем делать BCE loss:

$$\mathcal L_{BCE}(y, \hat y) = -\sum_i \left[y_i\log\sigma(\hat y_i) + (1-y_i)\log(1-\sigma(\hat y_i))\right].$$

Проблема с этой потерей заключается в том, что она имеет тенденцию приносить пользу классу **большинства** (фоновому) по отношению к классу **меньшинства** (переднему). Поэтому обычно применяются весовые коэффициенты к каждому классу:

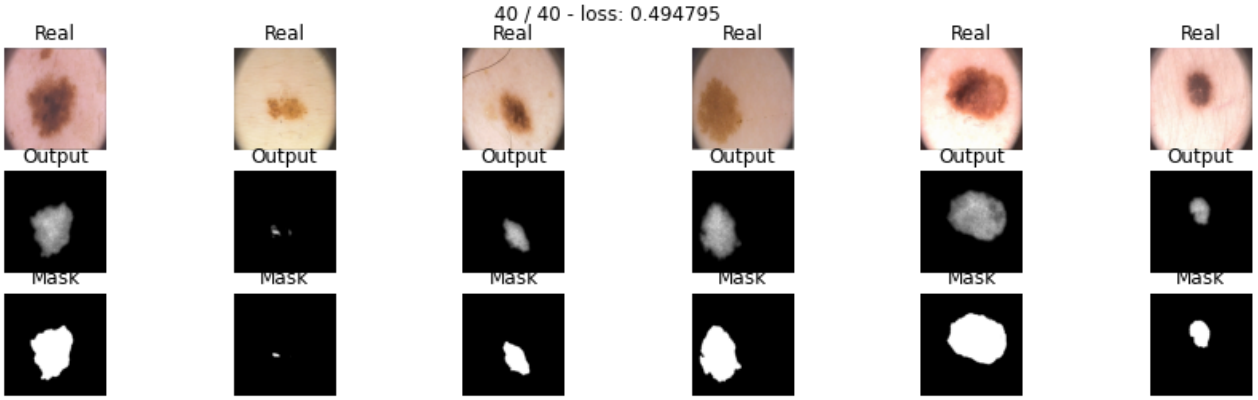
$$\mathcal L_{wBCE}(y, \hat y) = -\sum_i \alpha_i\left[y_i\log\sigma(\hat y_i) + (1-y_i)\log(1-\sigma(\hat y_i))\right].$$
Традиционно вес α_i определяется как обратная частота класса этого пикселя i, так что наблюдения миноритарного класса весят больше по отношению к классу большинства.

Еще одним недавним дополнением является взвешенный пиксельный вариант, которая взвешивает каждый пиксель по степени уверенности, которую мы имеем в предсказании этого пикселя.

$$\mathcal L_{focal}(y, \hat y) = -\sum_i \left[\left(1-\sigma(\hat y_i)\right)^\gamma y_i\log\sigma(\hat y_i) + (1-y_i)\log(1-\sigma(\hat y_i))\right].$$
Зафиксируем значение $\gamma=2$.

In [40]:
```
def focal_loss(y_real, y_pred, eps = 1e-7, gamma=2):
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps) # hint: torch.clamp
    loss = -torch.mean((1 - y_pred)**gamma * y_real * torch.log(y_pred) + (1 - y_real)
    return loss
```

In [41]:
```
model_focal = SegNet().to(device)

max_epochs = 40
optimizer = optim.Adam(model_focal.parameters(), lr=3e-4)
# scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.3)
history_focal = train(model_focal, optimizer, focal_loss, max_epochs, data_tr, data_val
```



In [42]:
```
score = score_model(model_focal, iou_pytorch, data_val)
score
```

Out[42]: 0.6813454031944275

In [43]:
```
# logging
name = "segnet_focal"
model_data = {
    "name": name,
    "history": history_focal,
    "score": score
```

```
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

[BONUS] Мир сегментационных лоссов [5 баллов]

В данном блоке предлагаем вам написать одну функцию потерь самостоятельно. Для этого необходимо прочитать статью и имплементировать ее. Кроме тако провести численное сравнение с предыдущими функциями. Какие варианты?

1) Можно учесть Total Variation 2) Lova 3) BCE но с Soft Targets (что-то типа label-smoothing для многоклассовой классификации) 4) Любой другой

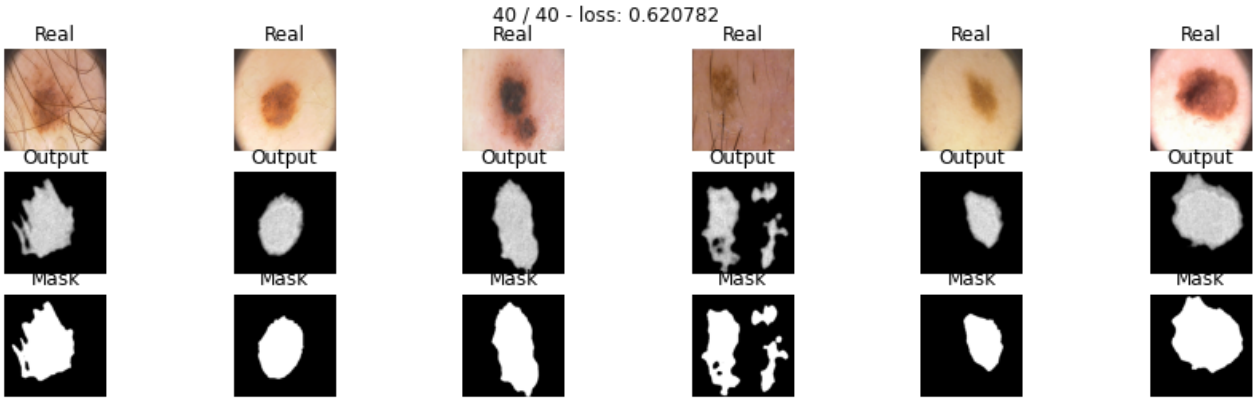
- Physiological Inspired Deep Neural Networks for Emotion Recognition". IEEE Access, 6, 53930-53943.
- Boundary loss for highly unbalanced segmentation
- Tversky loss function for image segmentation using 3D fully convolutional deep networks
- Correlation Maximized Structural Similarity Loss for Semantic Segmentation
- Topology-Preserving Deep Image Segmentation

Так как Тверский лосс очень похож на данные выше, то за него будет проставлено только 3 балла (при условии, если в модели нет ошибок при обучении). Постарайтесь сделать что-то интереснее.

A) Jaccard loss

In [44]:
```
def jaccard_loss(y_real, y_pred, eps = 1e-8):
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps)
    res = 1 - torch.sum(y_pred * y_real) / (torch.sum(y_pred) + torch.sum(y_real) - tor
    return res
```

In [45]:
```
model_jaccard = SegNet().to(device)

max_epochs = 40
optimizer = optim.AdamW(model_jaccard.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
history_jaccard = train(model_jaccard, optimizer, jaccard_loss, max_epochs, data_tr, da
```



In [46]:
```
score = score_model(model_jaccard, iou_pytorch, data_val)
score
```

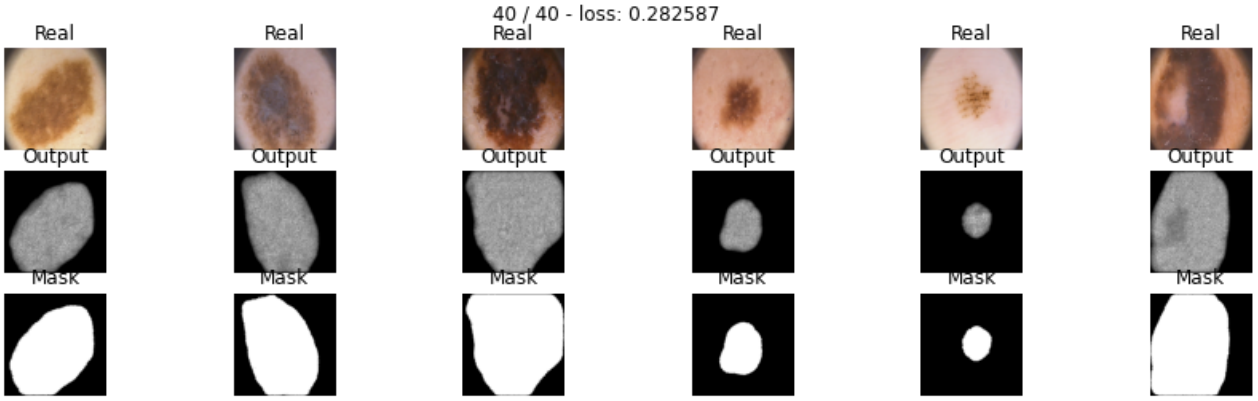
Out[46]:
```
0.7078787088394165
```

In [47]:
```
# Logging
name = "segnet_jaccard"
model_data = {
    "name": name,
    "history": history_jaccard,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

Б) Tversky loss

In [35]:
```
def tversky_loss(y_real, y_pred, beta=0.1, eps = 1e-8):
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps)
    res = 1 - torch.sum(y_pred * y_real) / (torch.sum(y_pred * y_real) +
                                            beta * torch.sum((1 - y_real) * y_pred) +
                                            (1 - beta) * torch.sum(y_real * (1 - y_pred
    return res
```

In [38]:
```
model_tversky = SegNet().to(device)

max_epochs = 40
optimizer = optim.AdamW(model_tversky.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
history_tversky = train(model_tversky, optimizer, tversky_loss, max_epochs, data_tr, da
```



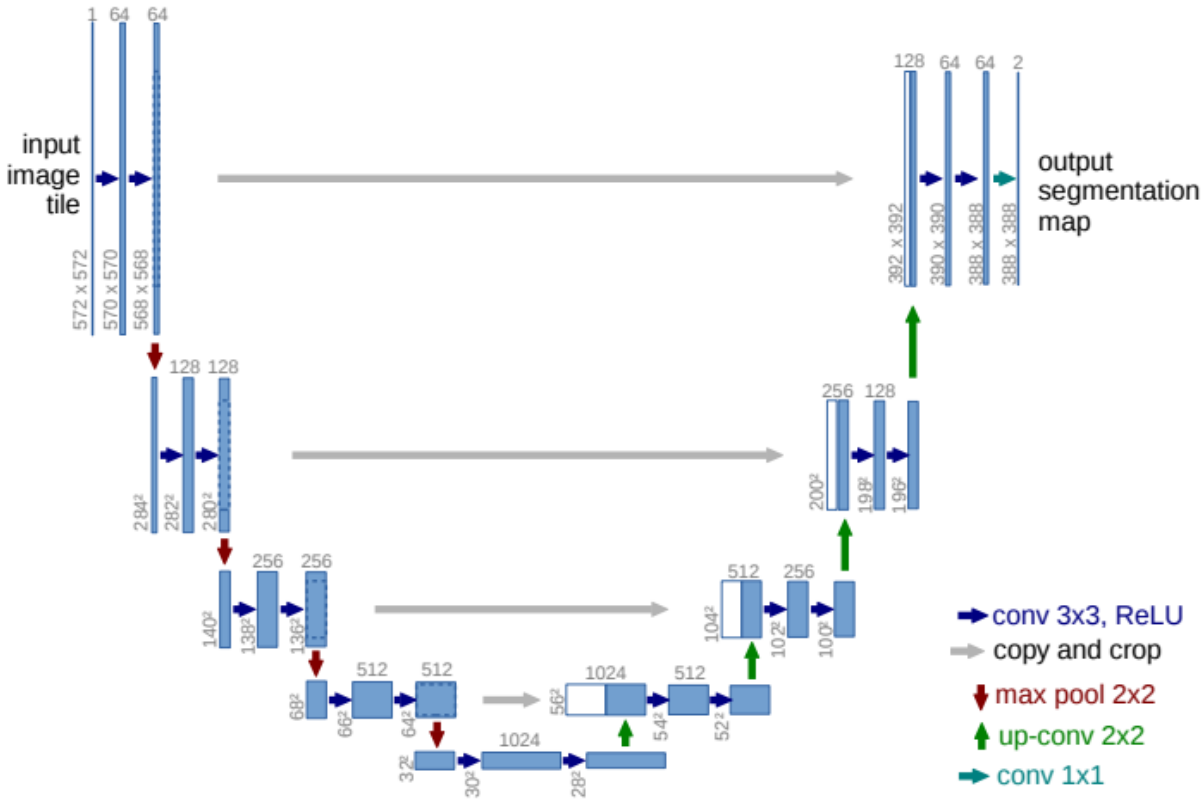
In [40]:
```
score = score_model(model_tversky, iou_pytorch, data_val)
score
```

Out[40]:
```
0.745946615934372
```

In [41]:
```
# logging
name = "segnet_tversky"
model_data = {
    "name": name,
    "history": history_tversky,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

U-Net [2 балла]

U-Net это архитектура нейронной сети, которая получает изображение и выводит его. Первоначально он был задуман для семантической сегментации (как мы ее будем использовать), но он настолько успешен, что с тех пор используется в других контекстах. Учитывая медицинское изображение, он выводит изображение в оттенках серого, представляющее вероятность того, что каждый пиксель является интересующей областью.



У нас в архитектуре все так же существует енкодер и декодер, как в **SegNet**, но отличительной особеностью данной модели являются skip-conenctions. Элементы соединяющие части декодера и енкодера. То есть для того чтобы передать на вход декодера тензор, мы конкатенируем симметричный выход с энкодера и выход предыдущего слоя декодера.

- Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

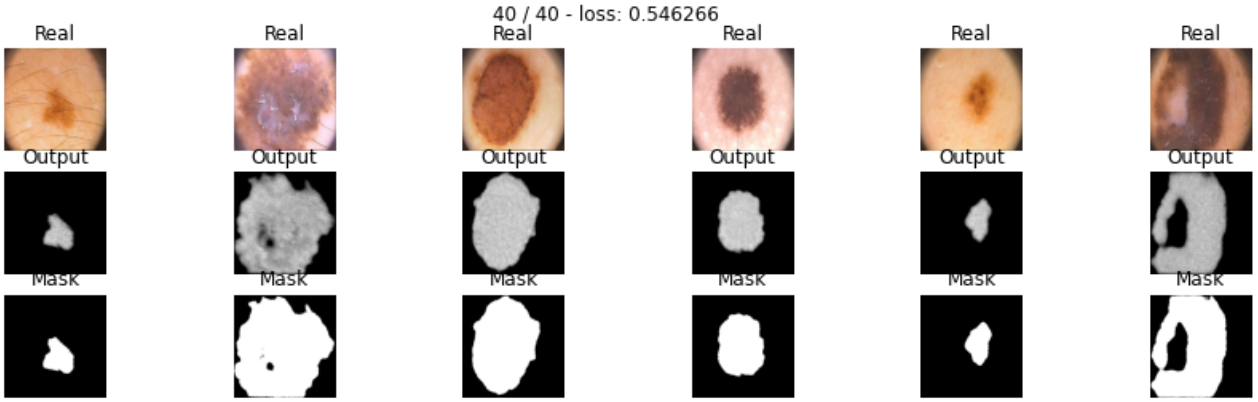
In [21]:
```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        self.enc_conv0 = nn.Sequential(
                         nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(64),
                         nn.ReLU(),
                         nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(64),
                         nn.ReLU()
                        )
        self.pool0 = torch.nn.MaxPool2d(2, 2, return_indices=True)   # 256 -> 128
        self.enc_conv1 = nn.Sequential(
                         nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(128),
                         nn.ReLU(),
                         nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(128),
                         nn.ReLU()
                        )
        self.pool1 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 128 -> 64
        self.enc_conv2 = nn.Sequential(
                         nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU(),
                         nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU(),
                         nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU()
                        )
        self.pool2 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 64 -> 32
        self.enc_conv3 = nn.Sequential(
                         nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU()
                        )
        self.pool3 = torch.nn.MaxPool2d(2, 2, return_indices=True)  # 32 -> 16
        # bottleneck
        self.bottleneck_conv = nn.Sequential(
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
```

```
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU()
                        )
        # decoder (upsampling)
        self.upsample0 = torch.nn.MaxUnpool2d(2, 2) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
                            nn.Conv2d(512*2, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU(),
                            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU(),
                            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU()
                        )
        self.upsample1 = torch.nn.MaxUnpool2d(2, 2)  # 32 -> 64
        self.dec_conv1 = nn.Sequential(
                            nn.Conv2d(256*2, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU()
                        )
        self.upsample2 = torch.nn.MaxUnpool2d(2, 2)   # 64 -> 128
        self.dec_conv2 = nn.Sequential(
                            nn.Conv2d(128*2, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU()
                        )
        self.upsample3 = torch.nn.MaxUnpool2d(2, 2)   # 128 -> 256
        self.dec_conv3 = nn.Sequential(
                            nn.Conv2d(64*2, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 1, kernel_size=1, stride=1),
                            nn.BatchNorm2d(1),
                            nn.ReLU()
                        )

    def forward(self, x):
```

```
        # encoder
        e0, indices0 = self.pool0(self.enc_conv0(x))
        e1, indices1 = self.pool1(self.enc_conv1(e0))
        e2, indices2 = self.pool2(self.enc_conv2(e1))
        e3, indices3 = self.pool3(self.enc_conv3(e2))

        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        # print(torch.cat([e3, b], dim=1).shape, indices3.shape)
        d0 = self.dec_conv0(self.upsample0(torch.cat([e3, b], dim=1), torch.cat([indice
        d1 = self.dec_conv1(self.upsample1(torch.cat([e2, d0], dim=1), torch.cat([indic
        d2 = self.dec_conv2(self.upsample2(torch.cat([e1, d1], dim=1), torch.cat([indic
        d3 = self.dec_conv3(self.upsample3(torch.cat([e0, d2], dim=1), torch.cat([indic
        return d3
```

In [22]:
```
unet_model = UNet().to(device)
```

In [23]:
```
history_unet = train(unet_model, optim.Adam(unet_model.parameters(), 1e-4), bce_loss, 4
```



In [25]:
```
score = score_model(unet_model, iou_pytorch, data_val)
score
```

Out[25]:
```
0.8241539597511292
```

In [26]:
```
# logging
name = "unet_bce"
model_data = {
    "name": name,
    "history": history_unet,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

Новая модель путем изменения типа пулинга:

Max-Pooling for the downsampling and **nearest-neighbor Upsampling** for the upsampling.

Down-sampling:

```
conv = nn.Conv2d(3, 64, 3, padding=1)
pool = nn.MaxPool2d(3, 2, padding=1)
```

Up-Sampling

```
upsample = nn.Upsample(32)
conv = nn.Conv2d(64, 64, 3, padding=1)
```

Замените max-pooling на convolutions с stride=2 и upsampling на transpose-convolutions с stride=2.

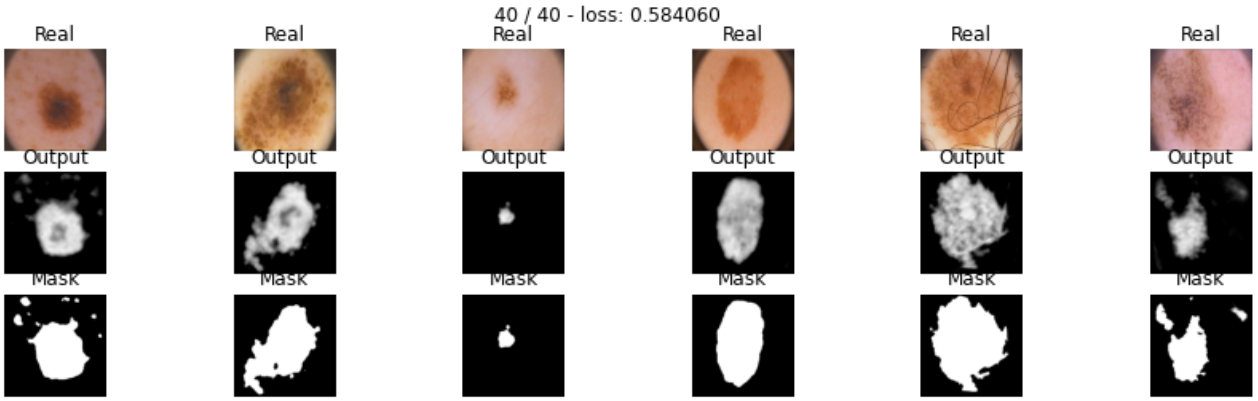
In [21]:
```
class UNet2(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        self.enc_conv0 = nn.Sequential(
                         nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(64),
                         nn.ReLU(),
                         nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(64),
                         nn.ReLU()
                        )
        self.pool0 = torch.nn.Conv2d(64, 64, kernel_size=2, stride=2, padding=0)   # 25
        self.enc_conv1 = nn.Sequential(
                         nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(128),
                         nn.ReLU(),
                         nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(128),
                         nn.ReLU()
                        )
        self.pool1 = torch.nn.Conv2d(128, 128, kernel_size=2, stride=2, padding=0)  # 1
        self.enc_conv2 = nn.Sequential(
                         nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU(),
                         nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU(),
                         nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(256),
                         nn.ReLU()
                        )
        self.pool2 = torch.nn.Conv2d(256, 256, kernel_size=2, stride=2, padding=0)  # 6
        self.enc_conv3 = nn.Sequential(
                         nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU(),
                         nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                         nn.BatchNorm2d(512),
                         nn.ReLU()
                        )
        self.pool3 = torch.nn.Conv2d(512, 512, kernel_size=2, stride=2, padding=0)  # 3
```

```
        # bottleneck
        self.bottleneck_conv = nn.Sequential(
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU(),
                            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(512),
                            nn.ReLU()
                        )
        # decoder (upsampling)
        self.upsample0 = torch.nn.ConvTranspose2d(512*2, 512*2, kernel_size=2, stride=2
        self.dec_conv0 = nn.Sequential(
                            nn.Conv2d(512*2, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU(),
                            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU(),
                            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(256),
                            nn.ReLU()
                        )
        self.upsample1 = torch.nn.ConvTranspose2d(256*2, 256*2, kernel_size=2, stride=2
        self.dec_conv1 = nn.Sequential(
                            nn.Conv2d(256*2, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU(),
                            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(128),
                            nn.ReLU()
                        )
        self.upsample2 = torch.nn.ConvTranspose2d(128*2, 128*2, kernel_size=2, stride=2
        self.dec_conv2 = nn.Sequential(
                            nn.Conv2d(128*2, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU()
                        )
        self.upsample3 = torch.nn.ConvTranspose2d(64*2, 64*2, kernel_size=2, stride=2)
        self.dec_conv3 = nn.Sequential(
                            nn.Conv2d(64*2, 64, kernel_size=3, stride=1, padding=1),
                            nn.BatchNorm2d(64),
                            nn.ReLU(),
                            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
```

```
                                nn.BatchNorm2d(64),
                                nn.ReLU(),
                                nn.Conv2d(64, 1, kernel_size=3, stride=1, padding=1),
                                nn.BatchNorm2d(1),
                                nn.ReLU()
                            )

    def forward(self, x):
        # encoder
        e0 = self.pool0(self.enc_conv0(x))
        e1 = self.pool1(self.enc_conv1(e0))
        e2 = self.pool2(self.enc_conv2(e1))
        e3 = self.pool3(self.enc_conv3(e2))

        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        # print(torch.cat([e3, b], dim=1).shape, indices3.shape)
        d0 = self.dec_conv0(self.upsample0(torch.cat([e3, b], dim=1)))
        d1 = self.dec_conv1(self.upsample1(torch.cat([e2, d0], dim=1)))
        d2 = self.dec_conv2(self.upsample2(torch.cat([e1, d1], dim=1)))
        d3 = self.dec_conv3(self.upsample3(torch.cat([e0, d2], dim=1)))  # no activatio
        return d3
```

In [22]:
```
unet2_model = UNet2().to(device)
```

In [23]:
```
optimizer = optim.AdamW(unet2_model.parameters(), lr=3e-4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
history_unet2 = train(unet2_model, optimizer, bce_loss, 40, data_tr, data_val, schedule
```



In [24]:
```
score = score_model(unet2_model, iou_pytorch, data_val)
score
```

Out[24]:
```
0.7235395610332489
```

In [25]:
```
# logging
name = "unet2_bce"
model_data = {
    "name": name,
    "history": history_unet2,
    "score": score
}
with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
    pickle.dump(model_data, file)
```

In [112…
```
# unet2_focal_model = UNet2().to(device)

# optimizer = optim.Adam(unet2_focal_model.parameters(), lr=3e-4)
# scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
# history_unet2_focal = train(unet2_focal_model, optimizer, focal_loss, 40, data_tr, da
```

In [113…
```
# score = score_model(unet2_focal_model, iou_pytorch, data_val)
```

In [114…
```
# # logging
# name = "unet2_focal"
# model_data = {
#     "name": name,
#     "history": history_unet2_focal,
#     "score": score
# }
# with open(os.path.join(data_path, name + ".pickle"), "wb") as file:
#     pickle.dump(model_data, file)
```

Сделайте вывод какая из моделей лучше

Вывод:

На мой взгляд, модель **model_unet** лучше

Отчет (6 баллов):

Ниже предлагается написать отчет о проделанно работе и построить графики для лоссов, метрик на валидации и тесте. Если вы пропустили какую-то часть в задании выше, то вы все равно можете получить основную часть баллов в отчете, если правильно зададите проверяемые вами гипотезы.

Аккуратно сравните модели между собой и соберите наилучшую архитектуру. Проверьте каждую модель с различными лоссами. Мы не ограничиваем вас в формате отчета, но проверяющий должен отчетливо понять для чего построен каждый график, какие выводы вы из него сделали и какой общий вывод можно сделать на основании данных моделей. Если вы захотите добавить что-то еще, чтобы увеличить шансы получения максимального балла, то добавляйте отдельное сравнение.

Дополнительные комментарии:

Пусть у вас есть N обученных моделей.

- Является ли отчетом N графиков с 1 линей? Да, но очень низкокачественным, потому что проверяющий не сможет сам сравнить их.
- Является ли отчетом 1 график с N линиями? Да, но скорее всего таким образом вы отразили лишь один эффект. Этого мало, чтобы сделать досточно суждений по поводу вашей работа.
- Я проверял метрики на трейне, и привел в результате таблицу с N числами, что не так? ключейвой момент тут, что вы измеряли на трейне ваши метрики, уверены ли вы, что

заивисмости останутся такими же на отложенной выборке?

- Я сделал отчет содержащий график лоссов и метрик, и у меня нет ошибок в основной части, но за отчет не стоит максимум, почему? Естестественно максимум баллов за отчет можно получить не за 2 графика (даже при условии их полной правильности). Проверяющий хочет видеть больше сравнений моделей, чем метрики и лоссы (особенно, если они на трейне).

Советы: попробуйте правильно поставить вопрос на который вы себе отвечаете и продемонстрировать таблицу/график, помогающий проверяющему увидеть ответ на этот вопрос. Пример: Ваня хочет узнать, с каким из 4-х лоссов модель (например, U-Net) имеет наилучшее качество. Что нужно сделать Ване? Обучить 4 одинаковых модели с разными лосс функциями. И измерить итоговое качество. Продемонстрировать результаты своих измерений и итоговый вывод. (warning: конечно же, это не идеально ответит на наш вопрос, так как мы не учитываем в экспериментах возможные различные типы ошибок, но для первого приближения этого вполне достаточно).

Примерное время на подготовку отчета 1 час, он содержит сравнеение метрик, график лоссов, выбор лучших моделей из нескольких кластеров и выбор просто лучшей модели, небольшой вывод по всему дз, возможно сравнение результирующих сегментаций, времени или числа параметров модели, проявляйте креативность.

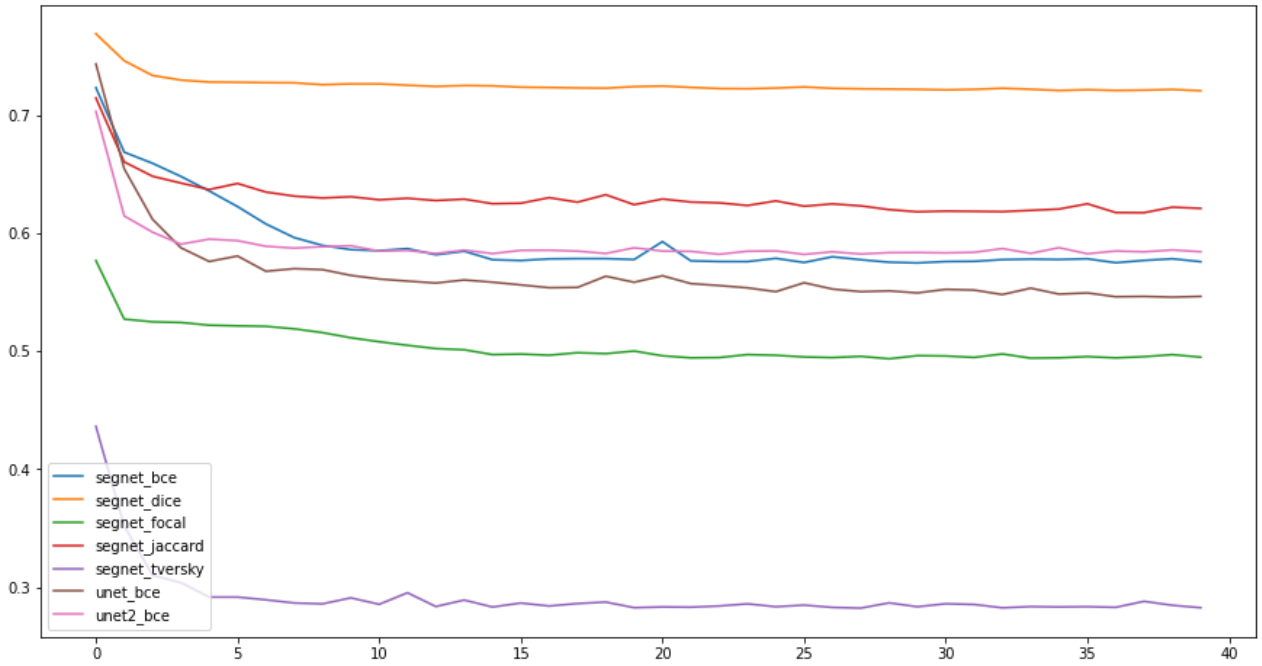
In [27]:
```
names = ["segnet_bce", "segnet_dice", "segnet_focal", "segnet_jaccard", "segnet_tversky
```

считываем логи

In [28]:
```
models_data = []
for name in names:
    with open(os.path.join(data_path, name + ".pickle"), 'rb') as file:
        model_data = pickle.load(file)
        models_data.append(model_data)
```

Сравниваем лоссы

In [29]:
```
plt.figure(figsize=(15, 8))
for model_data in models_data:
    plt.plot(model_data["history"], label=model_data["name"])
plt.legend()
plt.show()
```



Сравниваем скоры

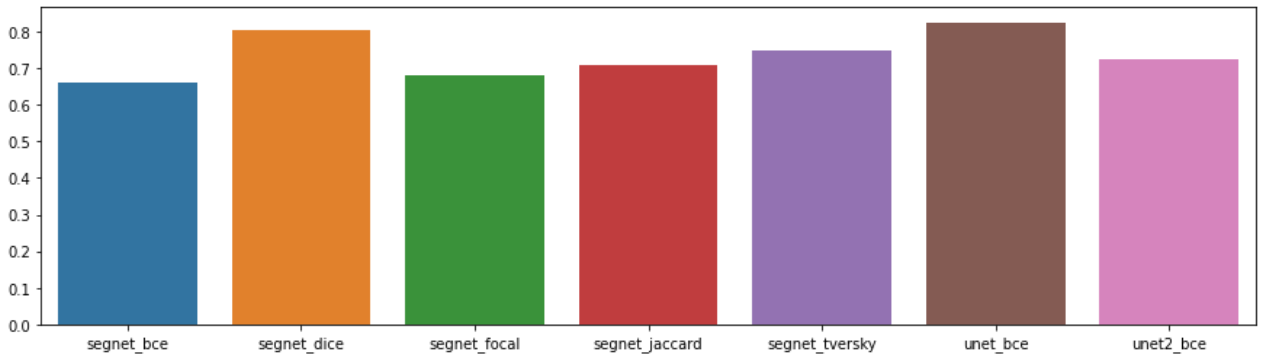
In [30]:
```
df = pd.DataFrame({model_data["name"]: model_data["score"] for model_data in models_dat
df
```

Out[30]:

	segnet_bce	segnet_dice	segnet_focal	segnet_jaccard	segnet_tversky	unet_bce	unet2_bce
score	0.661267	0.804034	0.681345	0.707879	0.745947	0.824154	0.72354

In [31]:
```
import seaborn as sns
```

In [32]:
```
g=sns.barplot(data=df, orient="v")
```



Выводы:

- наилучший скорр при сравнении на моделе **segnet** получился у **dice**
- наилучшей моделью на **BCE** по скору в данных конфигурациях показала себя **unet**
- абсолютным победителем также стал **unet** с функцией потерь **BCE**
- модели в целом обучались неплохо, границы достаточно четкие даже на **output**
- от **unet2** ожидалось большего, возможно, параметры можно подобрать и получше

- много зависит от подбора параметров, хорошо показало себя использование **scheduler**. На этапе исследовния и подбора периодически наблюдалось переобучение, особенно на **unet2**. С **scheduler** убирать переобучение получалось достаточно эффективно

In [ ]: