

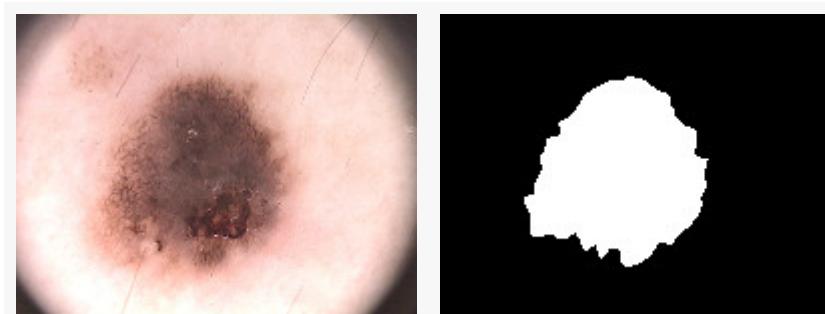
Мезга Александр
StepikID: 35630269

Ссылка на Google Colab:

https://colab.research.google.com/drive/1WG_nak3pMdMlf90tLAyXceYcatlCbWSh?usp=sharing

▼ Semantic Segmentation

1. Для начала мы скачаем датасет: [ADDI project](#).



2. Разархивируем .rar файл.

3. 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.

```
1 !nvidia-smi
```

```
↳ Sun Jun 7 19:04:07 2020
+-----+
| NVIDIA-SMI 440.82      Driver Version: 418.67      CUDA Version: 10.1 |
+-----+
| GPU  Name      Persistence-M  Bus-Id      Disp.A  Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0  Tesla P100-PCIE... Off    00000000:00:04.0 Off        0 |
| N/A   44C   P0    26W / 250W |     1MiB / 16280MiB |     0%   Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU  PID  Type  Process name          Usage        |
|-----+-----+-----+-----|
| No running processes found            |
+-----+
```

```
1 ! wget https://www.dropbox.com/s/4q6kwg8de56eqnc/PH2Dataset.rar
```

```
↳ --2020-06-07 19:04:10-- https://www.dropbox.com/s/4q6kwg8de56eqnc/PH2Dataset.rar
Resolving www.dropbox.com (www.dropbox.com)... 162.125.82.1, 2620:100:6032:1::a27d:5201
Connecting to www.dropbox.com (www.dropbox.com)|162.125.82.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/4q6kwg8de56eqnc/PH2dataset.rar [following]
--2020-06-07 19:04:10-- https://www.dropbox.com/s/raw/4q6kwg8de56eqnc/PH2Dataset.rar
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com/cd/0/inline/A5MtWBGmEJKY0wFj0t7Fd-sJtUqkyKeIm6TJQMyUkpF2o_q_yOX94OShO-1ZaFJDLgVK2yEc3QjORXRR_aqF6
--2020-06-07 19:04:10-- https://uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com/cd/0/inline/A5MtWBGmEJKY0wFj0t7Fd-sJtUqkyKeIm6TJQMyUkpF2o_q_yOX94OShO-1ZaFJDLgVK2yEc
Resolving uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com (uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com)... 162.125.82.15, 2620:100:6032:15::a27d:520f
Connecting to uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com (uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com)|162.125.82.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/A5M6SWCj2Q7-7Wx6UNLhTX0t4UQ0_GoIPNAe1Q45qm_v93ahFyIjpWXAM9KGV-k-ufjkP_SyoS7-dyY93VBGgFA1_21TdHlRpBnqjt4t7lRe9-grQ8Hk9B1_ce1Jv8iEZbgT8B4Me8rGedMYyF6y
--2020-06-07 19:04:11-- https://uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com/cd/0/inline2/A5M6SWCj2Q7-7Wx6UNLhTX0t4UQ0_GoIPNAe1Q45qm_v93ahFyIjpWXAM9KGV-k-ufjkP_S
Reusing existing connection to uce5da05595ad91a65bf136f8fbf.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 116457882 (111M) [application/rar]
Saving to: 'PH2Dataset.rar'

PH2Dataset.rar      100%[=====] 111.06M  13.0MB/s    in 8.4s

2020-06-07 19:04:20 (13.1 MB/s) - 'PH2Dataset.rar' saved [116457882/116457882]
```

```
1 get_ipython().system_raw("unrar x PH2Dataset.rar")
```

Структура 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\(\)](#).

```
1 images = []
2 lesions = []
3 from skimage.io import imread
4 import os
5 root = 'PH2Dataset'
6
7 for root, dirs, files in os.walk(os.path.join(root, 'PH2 Dataset images')):
8     if root.endswith('_Dermoscopic_Image'):
9         images.append(imread(os.path.join(root, files[0])))
10    if root.endswith('_lesion'):
11        lesions.append(imread(os.path.join(root, files[0])))
```

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

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

```
1 import numpy as np
2 X = np.array(X, np.float32)
3 Y = np.array(Y, np.float32)
4 print(f'Loaded {len(X)} images')
```

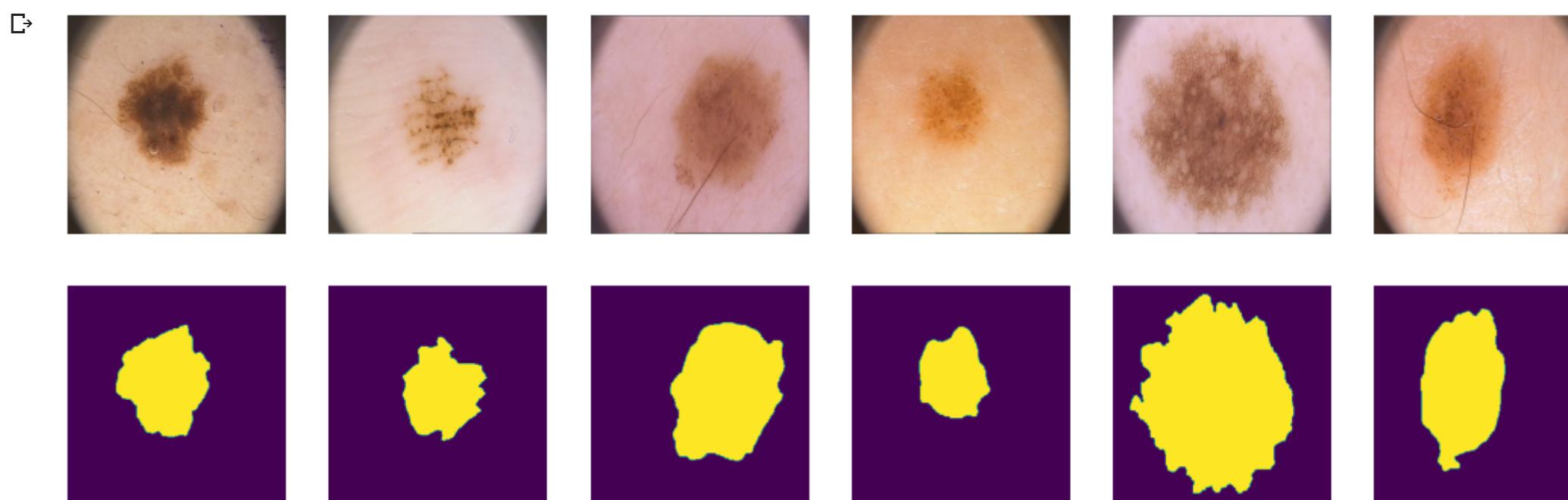
↳ Loaded 200 images

```
1 len(lesions)
```

↳ 200

Чтобы убедиться, что все корректно, мы нарисуем несколько изображений

```
1 import matplotlib.pyplot as plt
2 from IPython.display import clear_output
3
4 plt.figure(figsize=(18, 6))
5 for i in range(6):
6     plt.subplot(2, 6, i+1)
7     plt.axis("off")
8     plt.imshow(X[i])
9
10    plt.subplot(2, 6, i+7)
11    plt.axis("off")
12    plt.imshow(Y[i])
13 plt.show();
```



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

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

```
1 print(len(tr), len(val), len(ts))
```

↳ 100 50 50

▼ PyTorch DataLoader

```
1 from torch.utils.data import DataLoader
2 def set_batch_size(batch_size: int):
3     data_train = DataLoader(list(zip(np.rollaxis(X[tr], 3, 1), Y[tr], np.newaxis))),
4                             batch_size=batch_size, shuffle=True)
5     data_val = DataLoader(list(zip(np.rollaxis(X[val], 3, 1), Y[val], np.newaxis))),
6                           batch_size=batch_size, shuffle=True)
7     data_test = DataLoader(list(zip(np.rollaxis(X[ts], 3, 1), Y[ts], np.newaxis))),
8                           batch_size=batch_size, shuffle=True)
9     return data_train, data_val, data_test
```

```
1 import torch
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3 print(device)
```

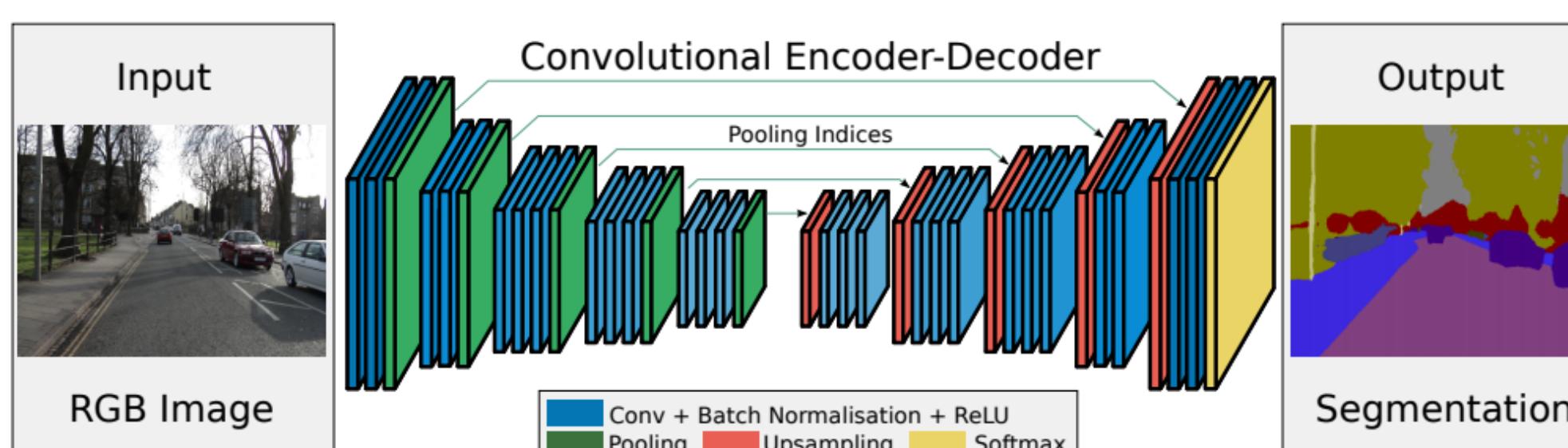
↳ cuda

```
1 scores_val = []
2 scores_test = []
```

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](#)

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 from torchvision import models
5 import torch.optim as optim
6 from time import time
7
8 from matplotlib import rcParams
9 rcParams['figure.figsize'] = (15,4)
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
```

```

07.06.2020 Mezga_[hw]semantic_segmentation.ipynb - Colaboratory
105
106
107     self.upsample3 = nn.Upsample(scale_factor=2, mode='bilinear') # nn.MaxUnpool2d(kernel_size=2, stride=2) # 12
108     self.dec_conv3 = nn.Sequential(
109         nn.Conv2d(in_channels=64, out_channels=1, kernel_size=3, padding=1),
110         nn.BatchNorm2d(1),
111         nn.ReLU(),
112
113         nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3, padding=1),
114         nn.BatchNorm2d(1),
115         nn.ReLU()
116     )
117
118     def forward(self, x):
119         # encoder
120         e0 = self.pool0(self.enc_conv0(x))
121         e1 = self.pool1(self.enc_conv1(e0))
122         e2 = self.pool2(self.enc_conv2(e1))
123         e3 = self.pool3(self.enc_conv3(e2))
124
125         # bottleneck
126         b = self.bottleneck_conv(e3)
127
128         # decoder
129         d0 = self.dec_conv0(self.upsample0(b))
130         d1 = self.dec_conv1(self.upsample1(d0))
131         d2 = self.dec_conv2(self.upsample2(d1))
132         d3 = self.dec_conv3(self.upsample3(d2)) # no activation
133
134     return d3

```

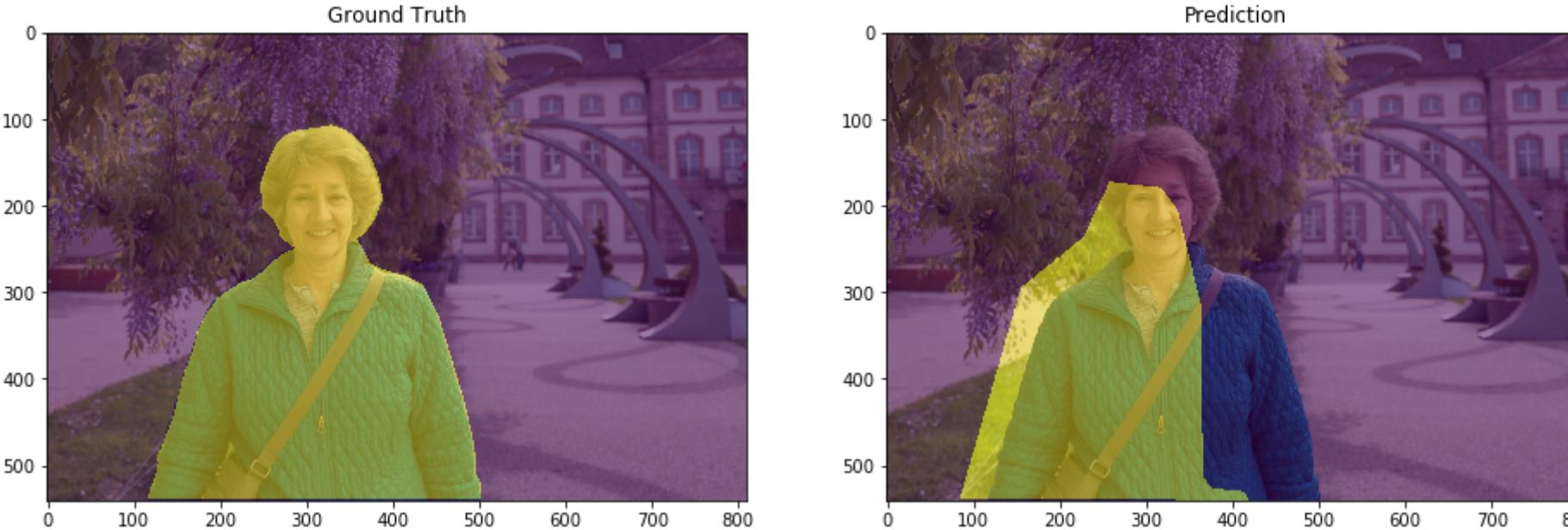
▼ Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

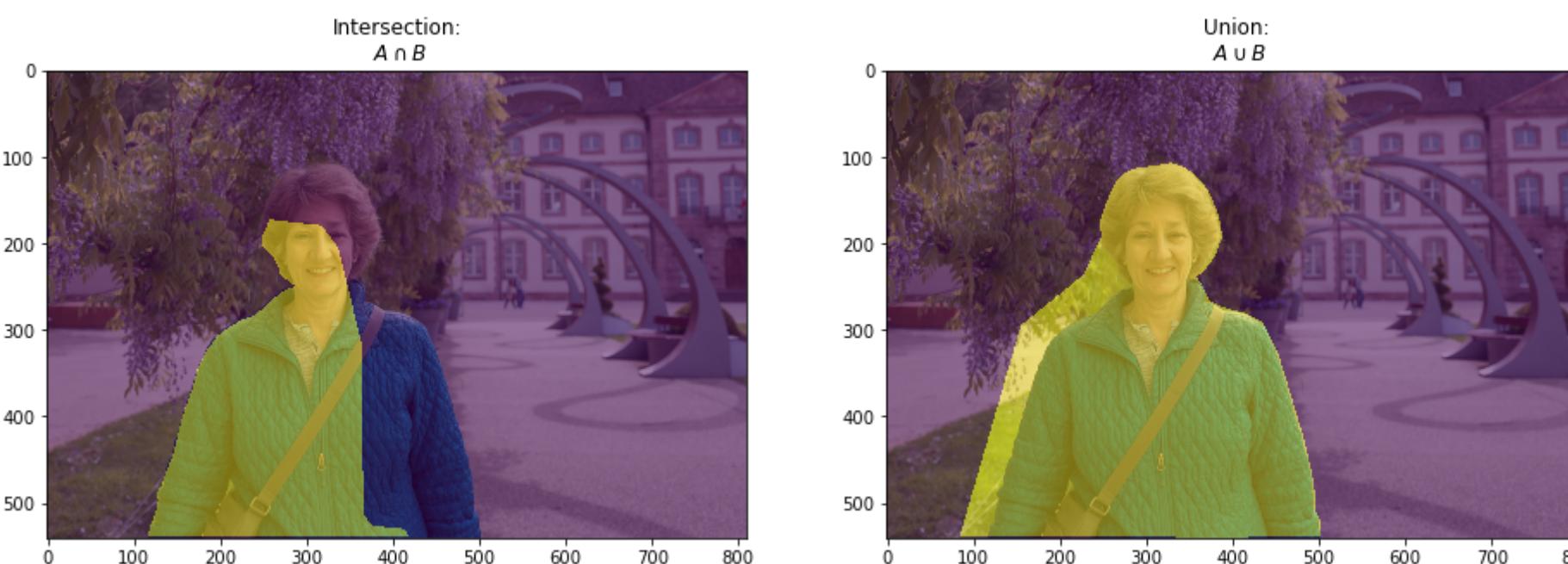
$$IoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup \text{prediction}}$$

Пересечение ($A \cap B$) состоит из пикселей, найденных как в маске предсказания, так и в основной маске истины, тогда как объединение ($A \cup B$) 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:



```

1 def iou_pytorch(outputs: torch.Tensor, labels: torch.Tensor):
2     # You can comment out this line if you are passing tensors of equal shape
3     # But if you are passing output from UNet or something it will most probably
4     # be with the BATCH x 1 x H x W shape
5     outputs = outputs.squeeze(1).byte() # BATCH x 1 x H x W => BATCH x H x W
6     labels = labels.squeeze(1).byte()
7     SMOOTH = 1e-8
8     intersection = (outputs & labels).float().sum((1, 2)) # Will be zero if Truth=0 or Prediction=0
9     union = (outputs | labels).float().sum((1, 2)) # Will be zzero if both are 0
10
11    iou = (intersection + SMOOTH) / (union + SMOOTH) # We smooth our devision to avoid 0/0
12
13    thresholded = torch.clamp(20 * (iou - 0.5), 0, 10).ceil() / 10 # This is equal to comparing with thresholds
14
15    return thresholded #

```

▼ функция лосса [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:

$$\mathcal{L}_{BCE}(y, \hat{y}) = - \sum_i [y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))]$$

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 $\lim_{x \rightarrow 0} \log(x) = \infty$ приводит к неустойчивости в процессе оптимизации. Рекомендуется посмотреть следующее [упрощение](#) в Тарая функция эквивалентна и не так подвержена численной неустойчивости.

$$\mathcal{L}_{BCE} = \hat{y} - y\hat{y} + \log(1 + \exp(-\hat{y})).$$

```
1 def bce_loss(y_real, y_pred, eps = 1e-8):
2     return (y_pred-(y_real*y_pred)+torch.log(1+torch.exp(-y_pred)+eps)).mean()
```

▼ Тренировка [1 балл]

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

```
1 def train(model, opt, loss_fn, epochs, data_tr, data_val, return_loss=False):
2     loss_arr = []
3     X_val, Y_val = next(iter(data_val))
4
5     for epoch in range(epochs):
6         tic = time()
7         print('* Epoch %d/%d' % (epoch+1, epochs))
8
9         avg_loss = 0
10        model.train() # train mode
11        for X_batch, Y_batch in data_tr:
12            # data to device
13            X_batch = X_batch.to(device)
14            Y_batch = Y_batch.to(device)
15
16            # set parameter gradients to zero
17            opt.zero_grad()
18
19            # forward
20            Y_pred = model(X_batch)
21            # loss = F.BCELoss(Y_batch, Y_pred)
22            # loss = torch.max(loss_fn(Y_pred, Y_batch), 1)
23            loss = loss_fn(Y_batch, Y_pred) # forward-pass
24            loss.backward() # backward-pass
25            opt.step() # update weights
26
27            # calculate loss to show the user
28            avg_loss += loss / len(data_tr)
29        toc = time()
30        print('loss: %f' % avg_loss)
31        loss_arr.append(avg_loss)
32
33        # show intermediate results
34        model.eval() # testing mode
35        Y_hat = model(X_val.to(device)).cpu().detach().numpy() # detach and put into cpu
36
37        # Visualize tools
38        clear_output(wait=True)
39        for k in range(6):
40            plt.subplot(2, 6, k+1)
41            plt.imshow(np.rollaxis(X_val[k].numpy(), 0, 3), cmap='gray')
42            plt.title('Real')
43            plt.axis('off')
44
45            plt.subplot(2, 6, k+7)
46            plt.imshow(Y_hat[k, 0], cmap='gray')
47            plt.title('Output')
48            plt.axis('off')
49            plt.suptitle('%d / %d - loss: %f' % (epoch+1, epochs, avg_loss))
50        plt.show()
51
52    if return_loss:
53        return loss_arr
```

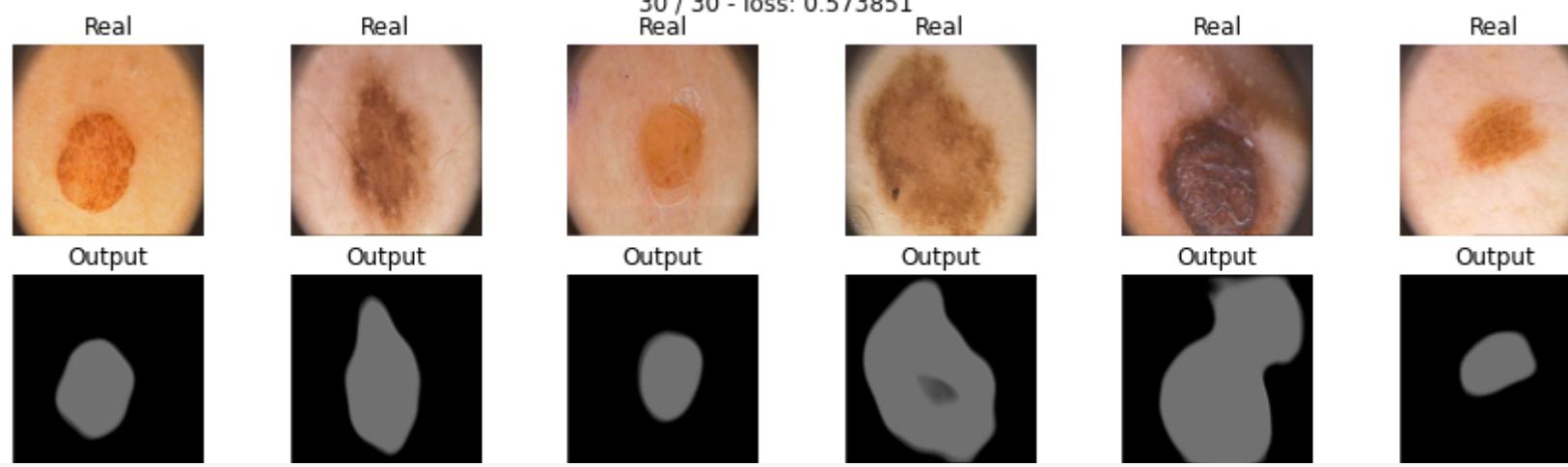
▼ Инференс [1 балл]

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

```
1 def predict(model, data):
2     model.eval() # testing mode
3     Y_pred = [X_batch for X_batch, _ in data]
4     return np.array(Y_pred)
5
6
7 def score_model(model, metric, data):
8     model.eval() # testing mode
9     scores = 0
10    for X_batch, Y_label in data:
11        Y_pred = model(X_batch.to(device))
12        scores += metric(Y_pred>0, Y_label.to(device)).mean().item()
13
14    return scores/len(data)
```

▼ ОСНОВНОЙ МОМЕНТ: обучение

```
1 import gc
2 gc.collect()
3
4 339
5
6 1 data_train, data_val, data_test = set_batch_size(10)
7
8 1 model_SegNet = SegNet().to(device)
9
10 1 max_epochs = 30
11 2 optimizer = optim.Adam(model_SegNet.parameters(), lr=0.0001)
12 3 loss_arr_model_SegNet = train(model_SegNet, optimizer, bce_loss, max_epochs,
13                                data_train, data_val, return_loss=True)
```



```

1 scores = score_model(model_SegNet, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(model_SegNet, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)

```

↳ /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
"See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.5980000138282776
Test Score = 0.6480000376701355

▼ Дополнительные функции лосса [2 балла]

В данном разделе вам потребуется имплементировать две функции потерь: DICE и Focal loss.

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:

$$\mathcal{L}_D(X, Y) = 1 - \frac{1}{256 \times 256} \times \sum_i \frac{2X_i Y_i}{X_i + Y_i}.$$

// Не забудьте подумать о численной нестабильности.

```

1 def dice_loss(y_real, y_pred, eps = 1e-8):
2     y_pred = torch.sigmoid(y_pred)
3     y_pred = y_pred.view(-1)
4     y_real = y_real.view(-1)
5
6     num = 2*torch.sum(y_real*y_pred)
7     den = torch.sum(y_real+y_pred)
8     res = (1 - num/(den + eps))
9     return res

```

Проводим тестирование:

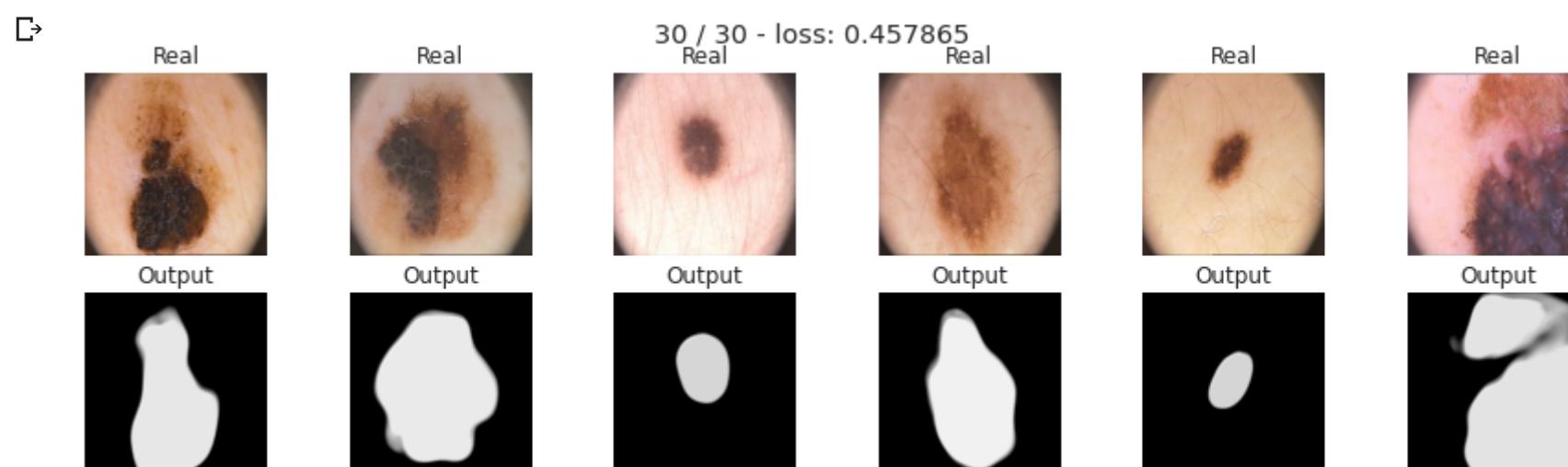
```
1 gc.collect()
```

↳ 89797

```

1 model_dice = SegNet().to(device)
2
3 max_epochs = 30
4 optimizer = optim.Adam(model_dice.parameters(), lr=0.0001)
5 loss_arr_model_dice = train(model_dice, optimizer, dice_loss, max_epochs,
6                               data_train, data_val, return_loss=True)

```



```

1 score = score_model(model_dice, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(model_dice, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)

```

↳ /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
"See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.74000004529953
Test Score = 0.7100000262260437

2. Focal loss:

Окей, мы уже с вами умеем делать BCE loss:

$$\mathcal{L}_{BCE}(y, \hat{y}) = - \sum_i [y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))].$$

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:

$$\mathcal{L}_{wBCE}(y, \hat{y}) = - \sum_i \alpha_i [y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))].$$

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.

$$\mathcal{L}_{focal}(y, \hat{y}) = - \sum_i [(1 - \sigma(\hat{y}_i))^{\gamma} y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))].$$

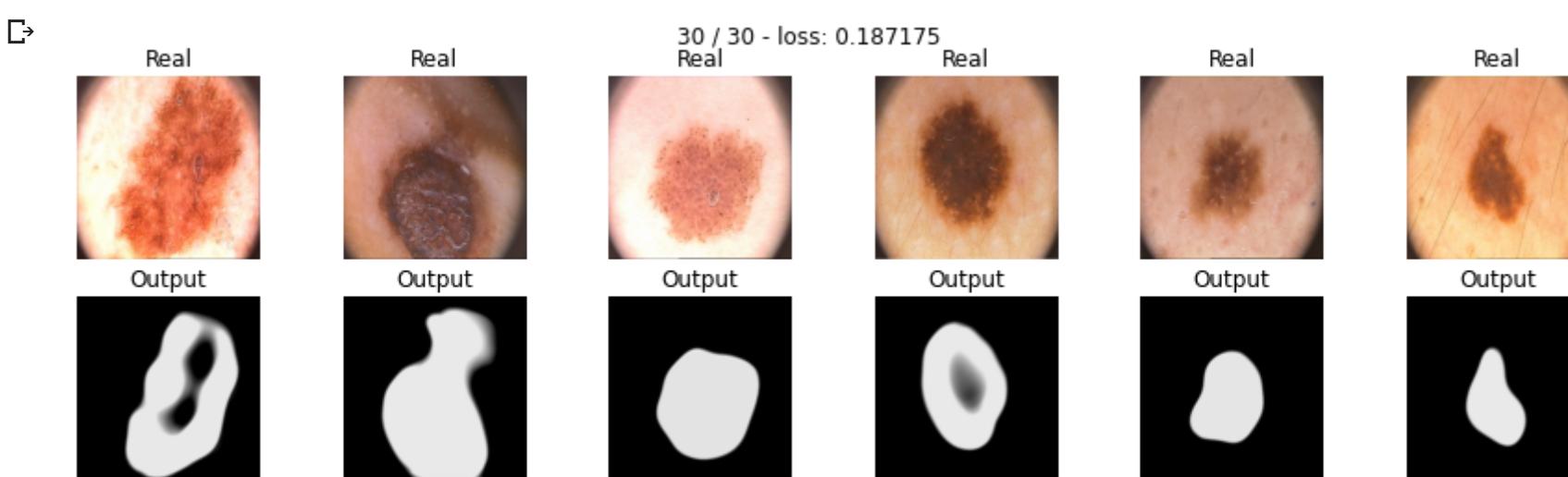
Задавшим значение $\nu = ?$

```
1 def focal_loss(y_real, y_pred, eps = 1e-8, gamma = 2, alpha=0.8, smooth=1):
2     y_pred = F.sigmoid(y_pred)
3
4     y_pred = y_pred.view(-1)
5     y_real = y_real.view(-1)
6
7     # BCE = F.binary_cross_entropy(y_pred, y_real, reduction='mean')
8     bce = bce_loss(y_real, y_pred)
9     bce_exp = torch.exp(-bce)
10    focal_loss = alpha * (1-bce_exp)**gamma * bce
11
12    return focal_loss
```

```
1 gc.collect()
```

```
65083
```

```
1 model_focal = SegNet().to(device)
2
3 max_epochs = 30
4 optimaizer = optim.Adam(model_focal.parameters(), lr=0.0001)
5 loss_arr_model_focal = train(model_focal, optimaizer, focal_loss, max_epochs, data_train, data_val, return_loss=True)
```



```
1 scores = score_model(model_focal, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(model_focal, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
  "See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.564000028371811
Test Score = 0.5660000503063202
```

▼ [BONUS] Мир сегментационных лоссов [5 баллов]

В данном блоке предлагаю написать вам 1 функцию потерь самостоятельно. Для этого необходимо прочитать статью и имплементировать ее. Кроме того 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 многослойной классификации) 4) Tversky loss 5) 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](#)

Exercise: Add the total variation term to the loss.

```
1 def tversky_loss(y_real, y_pred, smooth=1, alpha=0.5, beta=0.5):
2     y_pred = F.sigmoid(y_pred)
3
4     y_pred = y_pred.view(-1)
5     y_real = y_real.view(-1)
6
7     # True Positives, False Positives & False Negatives
8     TP = (y_pred * y_real).sum()
9     FP = ((1-y_real) * y_pred).sum()
10    FN = (y_real * (1-y_pred)).sum()
11
12    tversky = (TP + smooth) / (TP + alpha*FP + beta*FN + smooth)
13
14    return 1 - tversky
```

```
1 def tversky_focal_loss(y_real, y_pred, smooth=1, alpha=0.5, beta=0.5, gamma=1):
2     y_pred = F.sigmoid(y_pred)
3
4     y_pred = y_pred.view(-1)
5     y_real = y_real.view(-1)
6
7     # True Positives, False Positives & False Negatives
```

```

7 true positives, false positives & false negatives
8 TP = torch.sum(y_pred * y_real)
9 FP = torch.sum((1-y_real) * y_pred)
10 FN = torch.sum(y_real * (1-y_pred))
11
12 tversky = (TP + smooth) / (TP + alpha*FP + beta*FN + smooth)
13 focal_tversky = (1 - tversky)**gamma
14
15 return focal_tversky

```

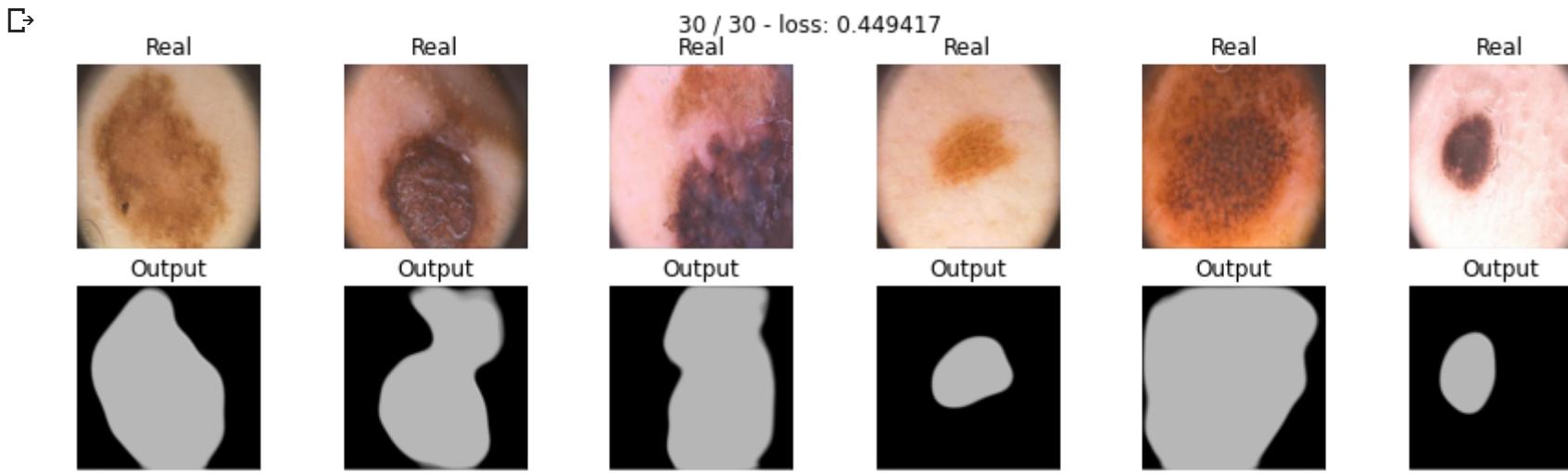
```
1 gc.collect()
```

65389

```

1 model_tversky = SegNet().to(device)
2
3 max_epochs = 30
4 optimaizer = optim.Adam(model_tversky.parameters(), lr=0.0001)
5 loss_arr_model_tversky = train(model_tversky, optimaizer, tversky_loss, max_epochs, data_train, data_val, return_loss)

```



```

1 scores = score_model(model_tversky, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(model_tversky, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)

```

/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
"See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.6740000247955322
Test Score = 0.7099999904632568

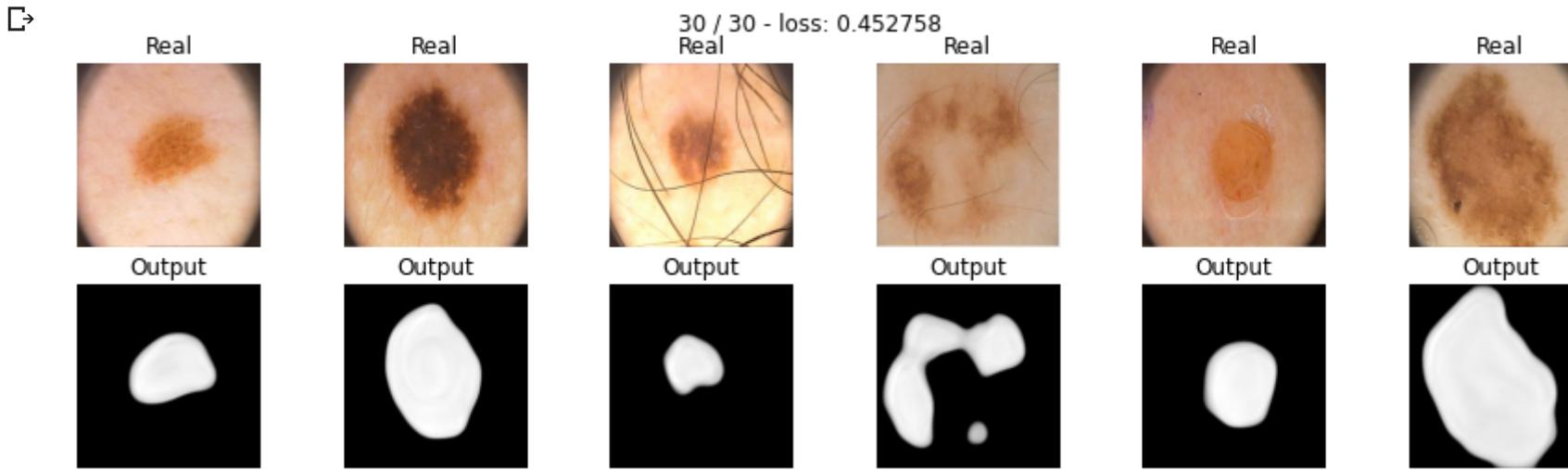
```
1 gc.collect()
```

64884

```

1 model_tversky_focal = SegNet().to(device)
2
3 max_epochs = 30
4 optimaizer = optim.Adam(model_tversky_focal.parameters(), lr=0.0001)
5 loss_arr_model_tversky_focal = train(model_tversky_focal, optimaizer,
6                                     tversky_focal_loss, max_epochs, data_train,
7                                     data_val, return_loss=True)

```



```

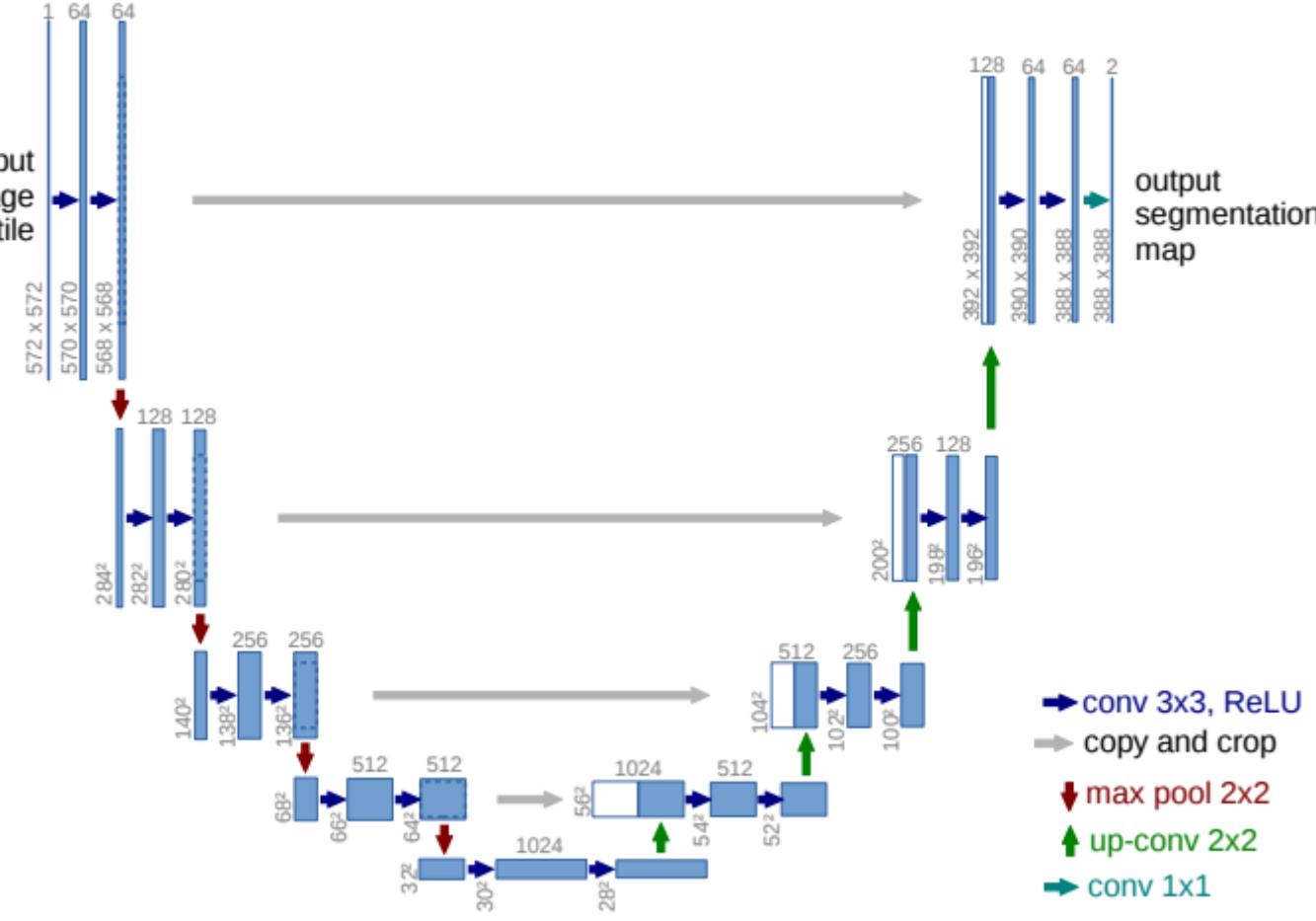
1 scores = score_model(model_tversky_focal, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(model_tversky_focal, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)

```

/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
"See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.6680000305175782
Test Score = 0.7360000371932983

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 особенностью 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.

```

1 class UNet(nn.Module):
2     def __init__(self):
3         super().__init__()
4
5         # encoder (downsampling)
6         self.enc_conv0 = nn.Sequential(
7             nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),
8             nn.BatchNorm2d(64),
9             nn.ReLU(),
10
11            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
12            nn.BatchNorm2d(64),
13            nn.ReLU(),
14        )
15        self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2) # 256 -> 128
16
17        self.enc_conv1 = nn.Sequential(
18            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
19            nn.BatchNorm2d(128),
20            nn.ReLU(),
21
22            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
23            nn.BatchNorm2d(128),
24            nn.ReLU(),
25        )
26        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128 -> 64
27
28        self.enc_conv2 = nn.Sequential(
29            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
30            nn.BatchNorm2d(256),
31            nn.ReLU(),
32
33            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
34            nn.BatchNorm2d(256),
35            nn.ReLU(),
36        )
37        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64 -> 32
38
39        self.enc_conv3 = nn.Sequential(
40            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1),
41            nn.BatchNorm2d(512),
42            nn.ReLU(),
43
44            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
45            nn.BatchNorm2d(512),
46            nn.ReLU(),
47        )
48        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32 -> 16
49 #####
50        # bottleneck
51        self.bottleneck_conv = nn.Sequential(
52            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3, padding=1),
53            nn.BatchNorm2d(1024),
54            nn.ReLU(),
55
56            nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, padding=1),
57            nn.BatchNorm2d(1024),
58            nn.ReLU(),
59        )
60 #####
61        # decoder (upsampling)
62        self.upsample0 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
63        self.dec_conv0 = nn.Sequential(
64            nn.Conv2d(in_channels=1024, out_channels=512, kernel_size=3, padding=1),
65            nn.BatchNorm2d(512),
66            nn.ReLU(),
67
68            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
69            nn.BatchNorm2d(512),
70            nn.ReLU(),
71        )
72        self.upsample1 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2) # 32 -> 64
73        self.dec_conv1 = nn.Sequential(
74            nn.Conv2d(512, 256, kernel_size=3, padding=1),
75            nn.BatchNorm2d(256),
76            nn.ReLU(),
77
78

```

```

79         nn.Conv2d(256, 256, kernel_size=3, padding=1),
80         nn.BatchNorm2d(256),
81         nn.ReLU(),
82     )
83     self.upsample2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2) # 64 -> 128
84     self.dec_conv2 = nn.Sequential(
85         nn.Conv2d(256, 128, kernel_size=3, padding=1),
86         nn.BatchNorm2d(128),
87         nn.ReLU(),
88
89         nn.Conv2d(128, 128, kernel_size=3, padding=1),
90         nn.BatchNorm2d(128),
91         nn.ReLU(),
92     )
93     self.upsample3 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2) # 128 -> 256
94     self.dec_conv3 = nn.Sequential(
95         nn.Conv2d(128, 64, kernel_size=3, padding=1),
96         nn.BatchNorm2d(64),
97         nn.ReLU(),
98
99         nn.Conv2d(64, 64, kernel_size=3, padding=1),
100        nn.BatchNorm2d(64),
101        nn.ReLU(),
102
103        nn.Conv2d(64, 1, kernel_size=3, padding=1),
104        nn.BatchNorm2d(1),
105        nn.ReLU(),
106    )
107
108    def forward(self, x):
109        # encoder
110        e0_conv = self.enc_conv0(x)
111        e0 = self.pool0(e0_conv)
112
113        e1_conv = self.enc_conv1(e0)
114        e1 = self.pool1(e1_conv)
115
116        e2_conv = self.enc_conv2(e1)
117        e2 = self.pool2(e2_conv)
118
119        e3_conv = self.enc_conv3(e2)
120        e3 = self.pool3(e3_conv)
121
122        # bottleneck
123        b = self.bottleneck_conv(e3)
124
125        # decoder
126        d0_up_cat = torch.cat((self.upsample0(b), e3_conv), 1)
127        d0 = self.dec_conv0(d0_up_cat)
128
129        d1_up_cat = torch.cat((self.upsample1(d0), e2_conv), 1)
130        d1 = self.dec_conv1(d1_up_cat)
131
132        d2_up_cat = torch.cat((self.upsample2(d1), e1_conv), 1)
133        d2 = self.dec_conv2(d2_up_cat)
134
135        d3_up_cat = torch.cat((self.upsample3(d2), e0_conv), 1)
136        d3 = self.dec_conv3(d3_up_cat)
137
138        return d3

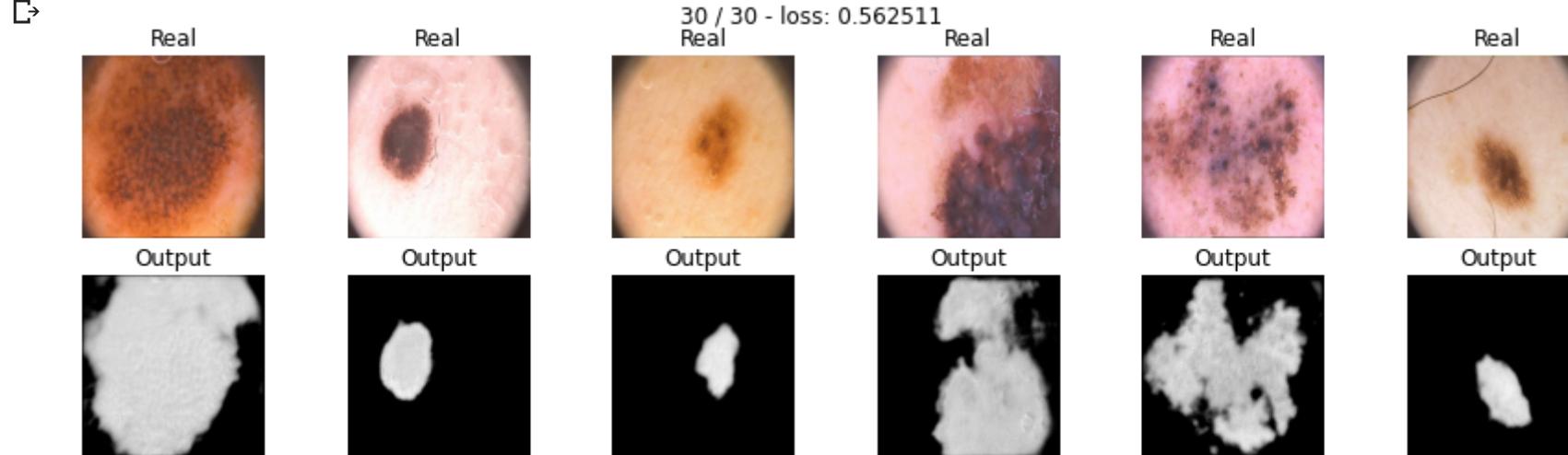
```

```
1 gc.collect()
```

↳ 64884

```
1 unet_model = UNet().to(device)
```

```
1 optimizer = optim.Adam(unet_model.parameters(), lr=0.0001)
2 loss_arr_unet_model = train(unet_model, optimizer, bce_loss, 30,
3                             data_train, data_val, return_loss=True)
```



```
1 scores = score_model(unet_model, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(unet_model, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

↳ Validation Score = 0.6540000319480896
Test Score = 0.7020000338554382

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)

1 class UNet2(nn.Module):
2     def __init__(self):
3         super().__init__()
4
5         # encoder (downsampling)
6         self.enc_conv0 = nn.Sequential(
7             nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),
8             nn.BatchNorm2d(64),
9             nn.ReLU(),
10
11            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
12            nn.BatchNorm2d(64),
13            nn.ReLU()
14        )
15        self.pool0 = nn.Conv2d(64, 64, kernel_size=2, stride=2)
16
17        self.enc_conv1 = nn.Sequential(
18            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
19            nn.BatchNorm2d(128),
20            nn.ReLU(),
21
22            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
23            nn.BatchNorm2d(128),
24            nn.ReLU()
25        )
26        self.pool1 = nn.Conv2d(128, 128, kernel_size=2, stride=2) # 128 -> 64
27
28        self.enc_conv2 = nn.Sequential(
29            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
30            nn.BatchNorm2d(256),
31            nn.ReLU(),
32
33            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
34            nn.BatchNorm2d(256),
35            nn.ReLU()
36        )
37        self.pool2 = nn.Conv2d(256, 256, kernel_size=2, stride=2) # 64 -> 32
38
39        self.enc_conv3 = nn.Sequential(
40            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1),
41            nn.BatchNorm2d(512),
42            nn.ReLU(),
43
44            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
45            nn.BatchNorm2d(512),
46            nn.ReLU()
47        )
48        self.pool3 = nn.Conv2d(512, 512, kernel_size=2, stride=2) # 32 -> 16
#####
# bottleneck
51        self.bottleneck_conv = nn.Sequential(
52            nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3, padding=1),
53            nn.BatchNorm2d(1024),
54            nn.ReLU(),
55
56            nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, padding=1),
57            nn.BatchNorm2d(1024),
58            nn.ReLU()
59        )
#####
# decoder (upsampling)
62        # self.upsample0 = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True) # 16 -> 32
63        self.upsample0 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
64        self.dec_conv0 = nn.Sequential(
65            nn.Conv2d(in_channels=1024, out_channels=512, kernel_size=3, padding=1),
66            nn.BatchNorm2d(512),
67            nn.ReLU(),
68
69            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
70            nn.BatchNorm2d(512),
71            nn.ReLU()
72        )
73        self.upsample1 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2) # 32 -> 64
74        self.dec_conv1 = nn.Sequential(
75            nn.Conv2d(512, 256, kernel_size=3, padding=1),
76            nn.BatchNorm2d(256),
77            nn.ReLU(),
78
79            nn.Conv2d(256, 256, kernel_size=3, padding=1),
80            nn.BatchNorm2d(256),
81            nn.ReLU()
82        )
83        self.upsample2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2) # 64 -> 128
84        self.dec_conv2 = nn.Sequential(
85            nn.Conv2d(256, 128, kernel_size=3, padding=1),
86            nn.BatchNorm2d(128),
87            nn.ReLU(),
88
89            nn.Conv2d(128, 128, kernel_size=3, padding=1),
90            nn.BatchNorm2d(128),
91            nn.ReLU()
92        )
93        self.upsample3 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2) # 128 -> 256
94        self.dec_conv3 = nn.Sequential(
95            nn.Conv2d(128, 64, kernel_size=3, padding=1),
96            nn.BatchNorm2d(64),
97            nn.ReLU(),
98
99            nn.Conv2d(64, 64, kernel_size=3, padding=1),
100           nn.BatchNorm2d(64),
101           nn.ReLU(),
102
103           nn.Conv2d(64, 1, kernel_size=3, padding=1),
104           nn.BatchNorm2d(1),
105           nn.ReLU()
106       )
107   )
108
109   def forward(self, x):
110       """
111       ...
112       """

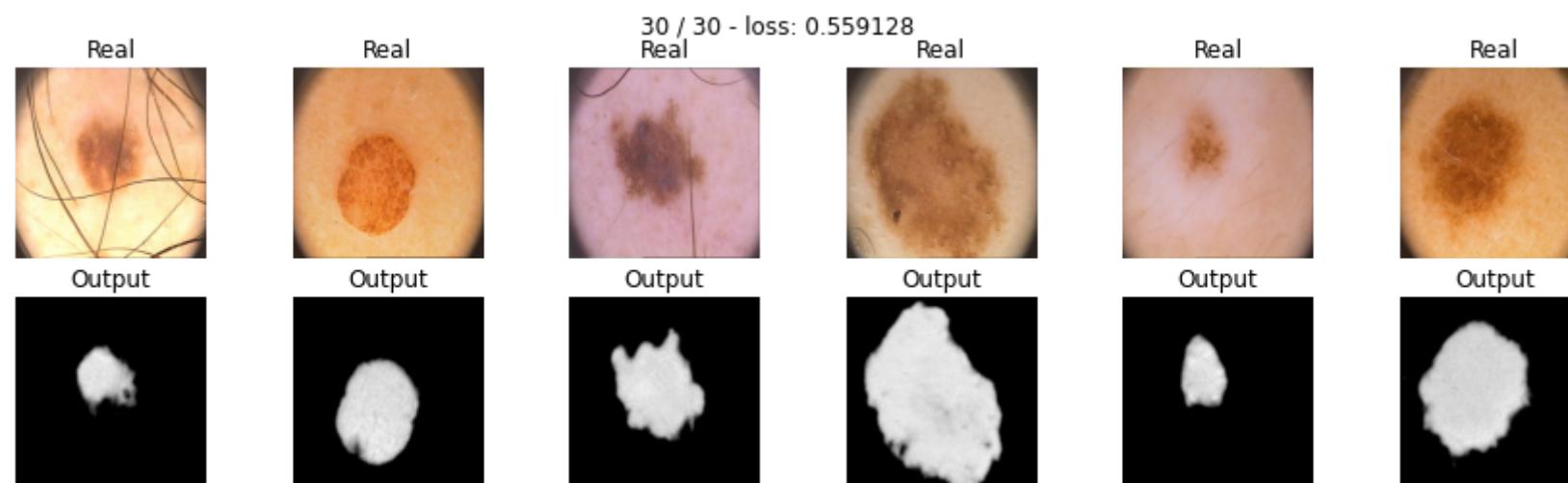
```

```
07.06.2020 Mezga_[hw]semantic_segmentation.ipynb - Colaboratory
110     # encoder
111     e0_conv = self.enc_conv0(x)
112     e0 = self.pool0(e0_conv)
113     e1_conv = self.enc_conv1(e0)
114     e1 = self.pool1(e1_conv)
115     e2_conv = self.enc_conv2(e1)
116     e2 = self.pool2(e2_conv)
117     e3_conv = self.enc_conv3(e2)
118     e3 = self.pool3(e3_conv)
119
120     # bottleneck
121     b = self.bottleneck_conv(e3)
122
123     # decoder
124     d0_up_cat = torch.cat((self.upsample0(b), e3_conv), 1)
125     d0 = self.dec_conv0(d0_up_cat)
126
127     d1_up_cat = torch.cat((self.upsample1(d0), e2_conv), 1)
128     d1 = self.dec_conv1(d1_up_cat)
129
130     d2_up_cat = torch.cat((self.upsample2(d1), e1_conv), 1)
131     d2 = self.dec_conv2(d2_up_cat)
132
133     d3_up_cat = torch.cat((self.upsample3(d2), e0_conv), 1)
134     d3 = self.dec_conv3(d3_up_cat)
135
136     return d3
```

```
1 gc.collect()
```

64884

```
1 unet2_model = UNet2().to(device)  
2
```



```
1 scores = score_model(unet2_model, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(unet2_model, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

→ Validation Score = 0.718000042438507
Test Score = 0.7320000290870666

- Yu, Fisher, and Vladlen Koltun. "[Multi-scale context aggregation by dilated convolutions](#)." arXiv preprint arXiv:1706.05587.

- пробуйте написать сеть DilatedUNet, которая использует в одной из предыдущих моделей dilated свертки.

```
2 def __init__(self):
3     super().__init__()
4     self.enc_conv0 = nn.Sequential(
5         nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
6         nn.BatchNorm2d(64),
7         nn.ReLU(),
8
9         nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
10        nn.BatchNorm2d(64),
11        nn.ReLU()
12    )
13    self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2) # 256 -> 128
14
15    self.enc_conv1 = nn.Sequential(
16        nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
17        nn.BatchNorm2d(128),
18        nn.ReLU(),
19
20        nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
21        nn.BatchNorm2d(128),
22        nn.ReLU()
23    )
24    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128 -> 64
25
26    self.enc_conv2 = nn.Sequential(
27        nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
28        nn.BatchNorm2d(256),
29        nn.ReLU(),
30
31        nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
32        nn.BatchNorm2d(256),
33        nn.ReLU()
```

```

34 )
35 self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64 -> 32
36
37 self.enc_conv3 = nn.Sequential(
38     nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
39     nn.BatchNorm2d(512),
40     nn.ReLU(),
41
42     nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
43     nn.BatchNorm2d(512),
44     nn.ReLU()
45 )
46
47 self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32 -> 16
48
49 ######
50
51 # bottleneck
52 self.bottleneck_conv = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
53
54 #####
55
56 # decoder (upsampling)
57 self.upsample0 = nn.Upsample(scale_factor=2, mode='bilinear') #nn.MaxUnpool2d(2, 2) # 16 -> 32
58 self.dec_conv0 = nn.Sequential(
59     nn.Conv2d(in_channels=512, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
60     nn.BatchNorm2d(256),
61     nn.ReLU(),
62
63     nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
64     nn.BatchNorm2d(256),
65     nn.ReLU()
66 )
67
68 self.upsample1 = nn.Upsample(scale_factor=2, mode='bilinear') #nn.MaxUnpool2d(kernel_size=2, stride=2) # 32 -> 64
69 self.dec_conv1 = nn.Sequential(
70     nn.Conv2d(in_channels=256, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
71     nn.BatchNorm2d(128),
72     nn.ReLU(),
73
74     nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
75     nn.BatchNorm2d(128),
76     nn.ReLU()
77 )
78
79 self.upsample2 = nn.Upsample(scale_factor=2, mode='bilinear') #nn.MaxUnpool2d(kernel_size=2, stride=2) # 64 -> 128
80 self.dec_conv2 = nn.Sequential(
81     nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
82     nn.BatchNorm2d(64),
83     nn.ReLU(),
84
85     nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
86     nn.BatchNorm2d(64),
87     nn.ReLU()
88 )
89
90 self.upsample3 = nn.Upsample(scale_factor=2, mode='bilinear') # nn.MaxUnpool2d(kernel_size=2, stride=2) # 128 -> 256
91 self.dec_conv3 = nn.Sequential(
92     nn.Conv2d(in_channels=64, out_channels=1, kernel_size=3, stride=1, padding=1, dilation=1),
93     nn.BatchNorm2d(1),
94     nn.ReLU(),
95
96     nn.Conv2d(in_channels=1, out_channels=1, kernel_size=3, stride=1, padding=1, dilation=1),
97     nn.BatchNorm2d(1),
98     nn.ReLU()
99 )
100
101 def forward(self, x):
102     # encoder
103     e0 = self.pool0(self.enc_conv0(x))
104     e1 = self.pool1(self.enc_conv1(e0))
105     e2 = self.pool2(self.enc_conv2(e1))
106     e3 = self.pool3(self.enc_conv3(e2))
107
108     # bottleneck
109     b = self.bottleneck_conv(e3)
110
111     # decoder
112     d0 = self.dec_conv0(self.upsample0(b))
113     d1 = self.dec_conv1(self.upsample1(d0))
114     d2 = self.dec_conv2(self.upsample2(d1))
115     d3 = self.dec_conv3(self.upsample3(d2)) # no activation
116
117     return d3
118

```

```

1 class DilatedUNet(nn.Module):
2     def __init__(self):
3         super().__init__()
4
5         # encoder (downsampling)
6         self.enc_conv0 = nn.Sequential(
7             nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
8             nn.BatchNorm2d(64),
9             nn.ReLU(),
10
11             nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1, dilation=1),
12             nn.BatchNorm2d(64),
13             nn.ReLU()
14         )
15         self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2) # 256 -> 128
16
17         self.enc_conv1 = nn.Sequential(
18             nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
19             nn.BatchNorm2d(128),
20             nn.ReLU(),
21
22             nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=1, padding=1, dilation=1),
23             nn.BatchNorm2d(128),
24             nn.ReLU()
25         )
26         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128 -> 64
27
28         self.enc_conv2 = nn.Sequential(

```

```

29     nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
30     nn.BatchNorm2d(256),
31     nn.ReLU(),
32
33     nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=1, dilation=1),
34     nn.BatchNorm2d(256),
35     nn.ReLU(),
36 )
37 self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64 -> 32
38
39 self.enc_conv3 = nn.Sequential(
40     nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
41     nn.BatchNorm2d(512),
42     nn.ReLU(),
43
44     nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
45     nn.BatchNorm2d(512),
46     nn.ReLU(),
47 )
48 self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32 -> 16
49 #####
50 # bottleneck
51 self.bottleneck_conv = nn.Sequential(
52     nn.Conv2d(in_channels=512, out_channels=1024, kernel_size=3, stride=1, padding=1, dilation=1),
53     nn.BatchNorm2d(1024),
54     nn.ReLU(),
55
56     nn.Conv2d(in_channels=1024, out_channels=1024, kernel_size=3, stride=1, padding=1, dilation=1),
57     nn.BatchNorm2d(1024),
58     nn.ReLU(),
59 )
60 #####
61 # decoder (upsampling)
62 self.upsample0 = nn.ConvTranspose2d(1024 , 512, kernel_size=2, stride=2)
63 self.dec_conv0 = nn.Sequential(
64     nn.Conv2d(in_channels=1024, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
65     nn.BatchNorm2d(512),
66     nn.ReLU(),
67
68     nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1, dilation=1),
69     nn.BatchNorm2d(512),
70     nn.ReLU(),
71 )
72 self.upsample1 = nn.ConvTranspose2d(512 , 256, kernel_size=2, stride=2) # 32 -> 64
73 self.dec_conv1 = nn.Sequential(
74     nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1, dilation=1),
75     nn.BatchNorm2d(256),
76     nn.ReLU(),
77
78     nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, dilation=1),
79     nn.BatchNorm2d(256),
80     nn.ReLU(),
81 )
82 self.upsample2 = nn.ConvTranspose2d(256 , 128, kernel_size=2, stride=2) # 64 -> 128
83 self.dec_conv2 = nn.Sequential(
84     nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1, dilation=1),
85     nn.BatchNorm2d(128),
86     nn.ReLU(),
87
88     nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, dilation=1),
89     nn.BatchNorm2d(128),
90     nn.ReLU(),
91 )
92 self.upsample3 = nn.ConvTranspose2d(128 , 64, kernel_size=2, stride=2) # 128 -> 256
93 self.dec_conv3 = nn.Sequential(
94     nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1, dilation=1),
95     nn.BatchNorm2d(64),
96     nn.ReLU(),
97
98     nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, dilation=1),
99     nn.BatchNorm2d(64),
100    nn.ReLU(),
101
102    nn.Conv2d(64, 1, kernel_size=3, stride=1, padding=1, dilation=1),
103    nn.BatchNorm2d(1),
104    nn.ReLU(),
105 )
106 )
107
108 def forward(self, x):
109     # encoder
110     e0_conv = self.enc_conv0(x)
111     e0 = self.pool0(e0_conv)
112     e1_conv = self.enc_conv1(e0)
113     e1 = self.pool1(e1_conv)
114     e2_conv = self.enc_conv2(e1)
115     e2 = self.pool2(e2_conv)
116     e3_conv = self.enc_conv3(e2)
117     e3 = self.pool3(e3_conv)
118
119     # bottleneck
120     b = self.bottleneck_conv(e3)
121
122     # decoder
123     d0_up_cat = torch.cat((self.upsample0(b), e3_conv), 1)
124     d0 = self.dec_conv0(d0_up_cat)
125     d1_up_cat = torch.cat((self.upsample1(d0), e2_conv), 1)
126     d1 = self.dec_conv1(d1_up_cat)
127     d2_up_cat = torch.cat((self.upsample2(d1), e1_conv), 1)
128     d2 = self.dec_conv2(d2_up_cat)
129     d3_up_cat = torch.cat((self.upsample3(d2), e0_conv), 1)
130     d3 = self.dec_conv3(d3_up_cat)
131
132     return d3

```

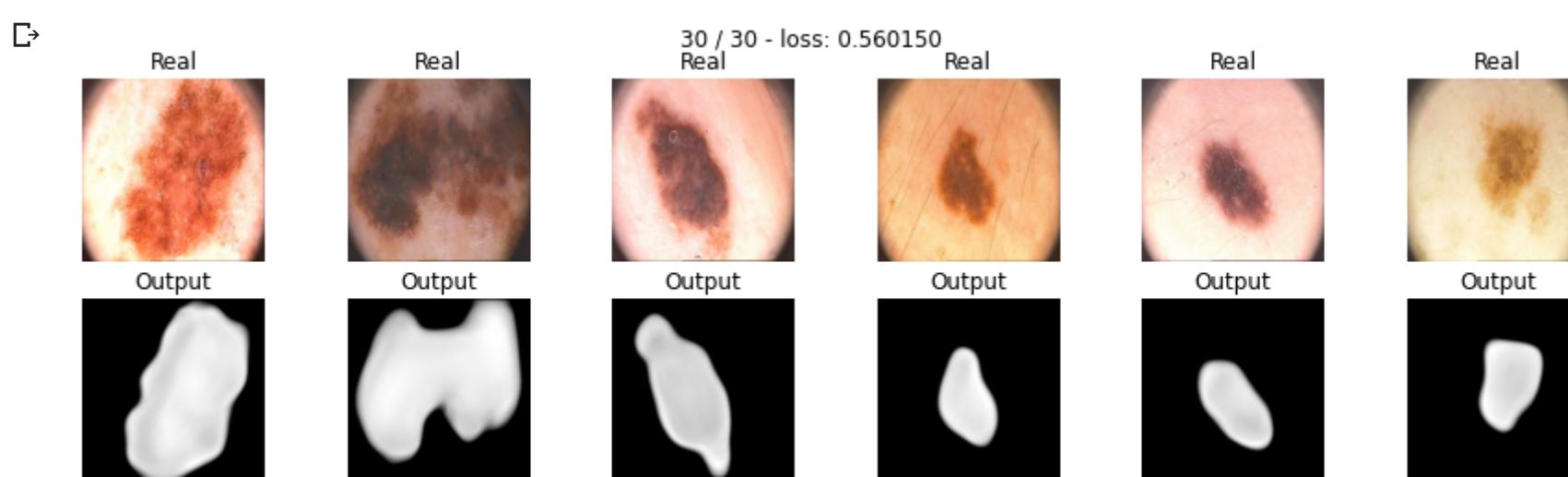
```
1 gc.collect()
```

64894

```
1 dilated_model_SegNet = DilatedSegNet().to(device)
```

```
1 optimizer = optim.Adam(dilated_model_SegNet.parameters(), lr=0.0001)
```

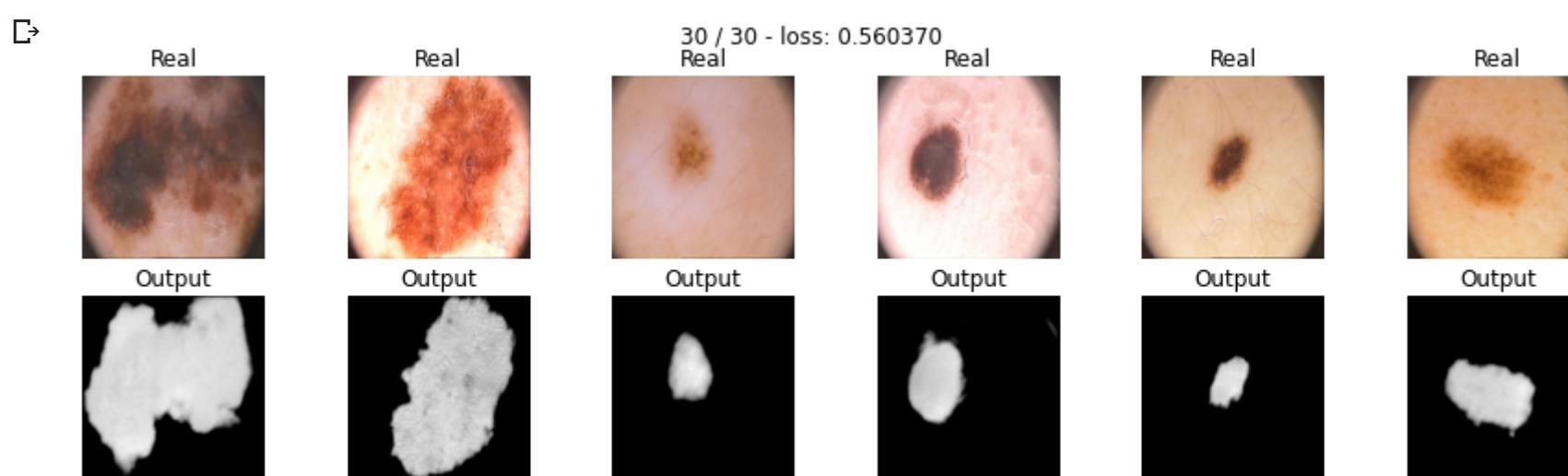
```
2 loss_arr_dilated_model_SegNet = train(model=dilated_model_SegNet, opt=optimizer,
3                                         loss_fn=bce_loss, epochs=30, data_tr=data_train,
4                                         data_val=data_val, return_loss=True)
```



```
1 scores = score_model(dilated_model_SegNet, iou_pytorch, data_val)
2 print('Validation Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(dilated_model_SegNet, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

```
USR/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changed to align_corners=False since 0.4.
  "See the documentation of nn.Upsample for details.".format(mode))
Validation Score = 0.6860000133514405
Test Score = 0.7379999995231629
```

```
1 dilated_model_UNet = DilatedUNet().to(device)
2 optimizer = optim.Adam(dilated_model_UNet.parameters(), lr=0.0001)
3 loss_arr_dilated_model_UNet = train(model=dilated_model_UNet, opt=optimizer,
4                                       loss_fn=bce_loss, epochs=30, data_tr=data_train,
5                                       data_val=data_val, return_loss=True)
```



```
1 scores = score_model(dilated_model_UNet, iou_pytorch, data_val)
2 print('Test Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(dilated_model_UNet, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

```
Test Score = 0.6700000166893005
Test Score = 0.7400000095367432
```

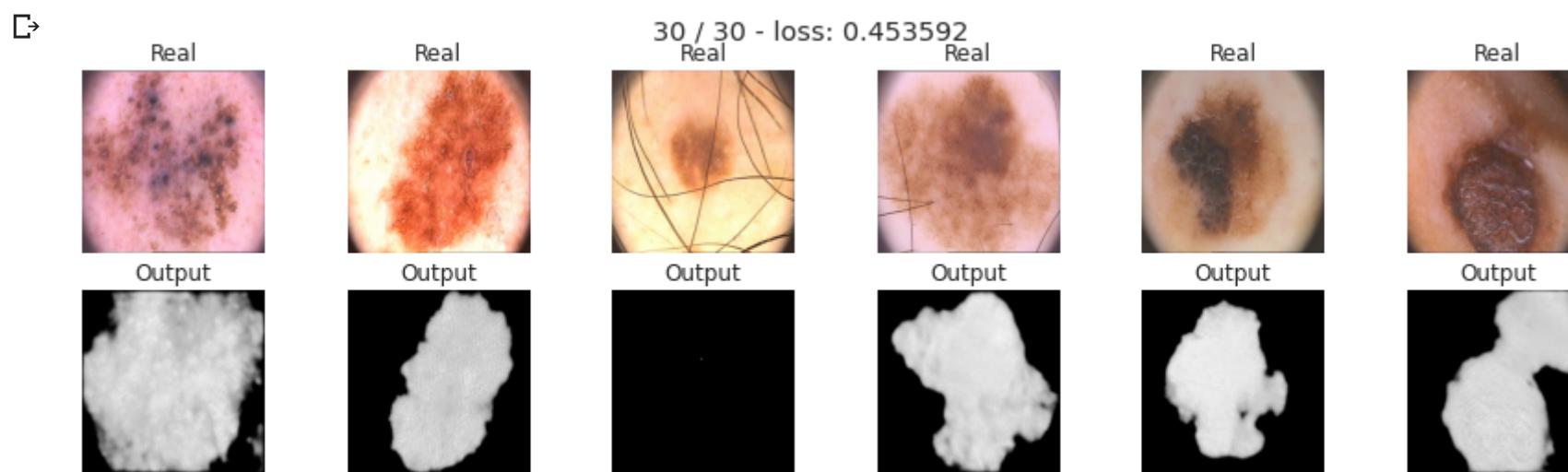
Протестируем также модель UNet с иными Loss функциями

UNet + Tversky

```
1 gc.collect()
```

```
2 19908
```

```
1 unet_model_tversky = UNet().to(device)
2 optimizer = optim.Adam(unet_model_tversky.parameters(), lr=0.0001)
3 loss_arr_unet_model_tversky = train(model=unet_model_tversky, opt=optimizer,
4                                       loss_fn=tversky_loss, epochs=30, data_tr=data_train,
5                                       data_val=data_val, return_loss=True)
```



```
1 scores = score_model(unet_model_tversky, iou_pytorch, data_val)
2 print('Test Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(unet_model_tversky, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)
```

```
Test Score = 0.6700000286102294
Test Score = 0.7400000333786011
```

UNet Dilated + Tversky

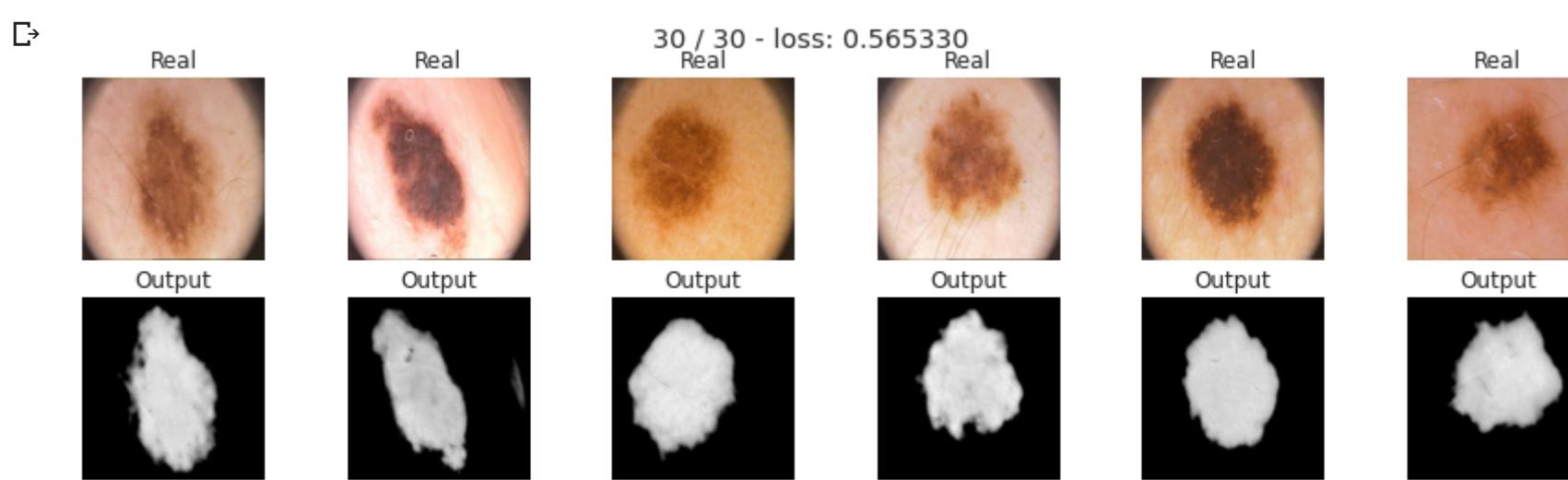
```
1 gc.collect()
```

↳ 19282

```

1 dilated_model_UNet_tversky = DilatedUNet().to(device)
2 optimizer = optim.Adam(dilated_model_UNet_tversky.parameters(), lr=0.0001)
3 loss_arr_dilated_model_UNet_tversky = train(model=dilated_model_UNet_tversky, opt=optimizer,
4                                              loss_fn=bce_loss, epochs=30, data_tr=data_train,
5                                              data_val=data_val, return_loss=True)

```



```

1 scores = score_model(dilated_model_UNet, iou_pytorch, data_val)
2 print('Test Score = ' + str(scores))
3 scores_val.append(scores)
4
5 scores = score_model(dilated_model_UNet, iou_pytorch, data_test)
6 print('Test Score = ' + str(scores))
7 scores_test.append(scores)

```

↳ Test Score = 0.6700000166893005
Test Score = 0.74000004529953

▼ Отчет (5 баллов):

Ниже предлагается написать отчет о проделанной работе и построить графики для лоссов, метрик на валидации и teste.

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.

```

1 x = np.arange(30)
2 loss_arr = [loss_arr_model_SegNet, loss_arr_model_dice, loss_arr_model_focal,
3             loss_arr_model_tversky, loss_arr_model_tversky_focal,
4             loss_arr_unet_model, loss_arr_unet2_model,
5             loss_arr_dilated_model_SegNet, loss_arr_dilated_model_UNet,
6             loss_arr_unet_model_tversky, loss_arr_dilated_model_UNet_tversky
7         ]

```

```

1 models = [model_SegNet, model_dice, model_focal, model_tversky,
2            model_tversky_focal, unet_model, unet2_model, dilated_model_SegNet,
3            dilated_model_UNet, unet_model_tversky, dilated_model_UNet_tversky]
4 names = ['SegNet+BCE', 'SegNet+Dice', 'SegNet+Focal', 'SegNet+Tversky',
5           'SegNet+Tversky Focal', 'UNet+BCE', 'UNet_2+BCE', 'dilatedSegNet+BCE',
6           'dilatedUNet+BCE', 'UNet+Tversky', 'dilatedUNet+Tversky']

```

```

1 from tqdm import tqdm
2 from matplotlib import pyplot as plt
3 import seaborn as sns
4 sns.set(style="whitegrid")
5
6 scores_test_checkout = []
7 scores_val_checkout = []
8 # _, _, data_test = set_batch_size(8)
9 for i in tqdm(range(len(names))):
10    scores_test_checkout.append(score_model(models[i], iou_pytorch, data_test))
11    scores_val_checkout.append(score_model(models[i], iou_pytorch, data_val))

```

↳ 0%| 0/11 [00:00<?, ?it/s]/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py:2973: UserWarning: Default upsampling behavior when mode=bilinear is changing to align_corners. See the documentation of nn.Upsample for details.".format(mode))
100%|██████████| 11/11 [00:09<00:00, 1.20it/s]

```

1 def plot_loss(loss_arr, names):
2     fig, ax = plt.subplots(figsize=(15,10))
3     for i in range(len(names)):
4         ax.plot(loss_arr[i], label=names[i])
5     plt.ylabel('loss')
6     plt.xlabel('epochs')
7
8     leg = ax.legend()

```

```

1 def plot_score(scores_test, scores_val, names):
2     # Create two subplots and unpack the output array immediately
3     f, (ax1, ax2) = plt.subplots(2, figsize=(20,10))
4     ax1.bar(names, scores_test, width=0.5)
5     ax1.set_title("Score evaluation on test dataset with epochs=30")
6     plt.ylabel('score')
7
8     ax2.bar(names, scores_val, width=0.5)
9     ax2.set_title("Score evaluation on validation dataset with epochs=30")
10    plt.ylabel('score')
11    plt.show()
12
13
14

```

В ходе выполнения задания были реализованы и протестированы следующие архитектуры:

- SegNet + BCE Loss
- SegNet + Dice Loss
- SegNet + Focal Loss
- SegNet + Tversky Loss
- SegNet + Tversky Focal Loss

- UNet + BCE Loss
- UNet_2 + BCE Loss
- dilatedSegNet + BCE Loss
- dilatedUNet + BCE Loss
- UNet + Tversky Loss
- dilatedUNet + Tversky Loss

Интерпретируя результаты стоит учесть, что датасет состоит из 200 экземпляров, чего недостаточно для полноценного и объективного анализа.

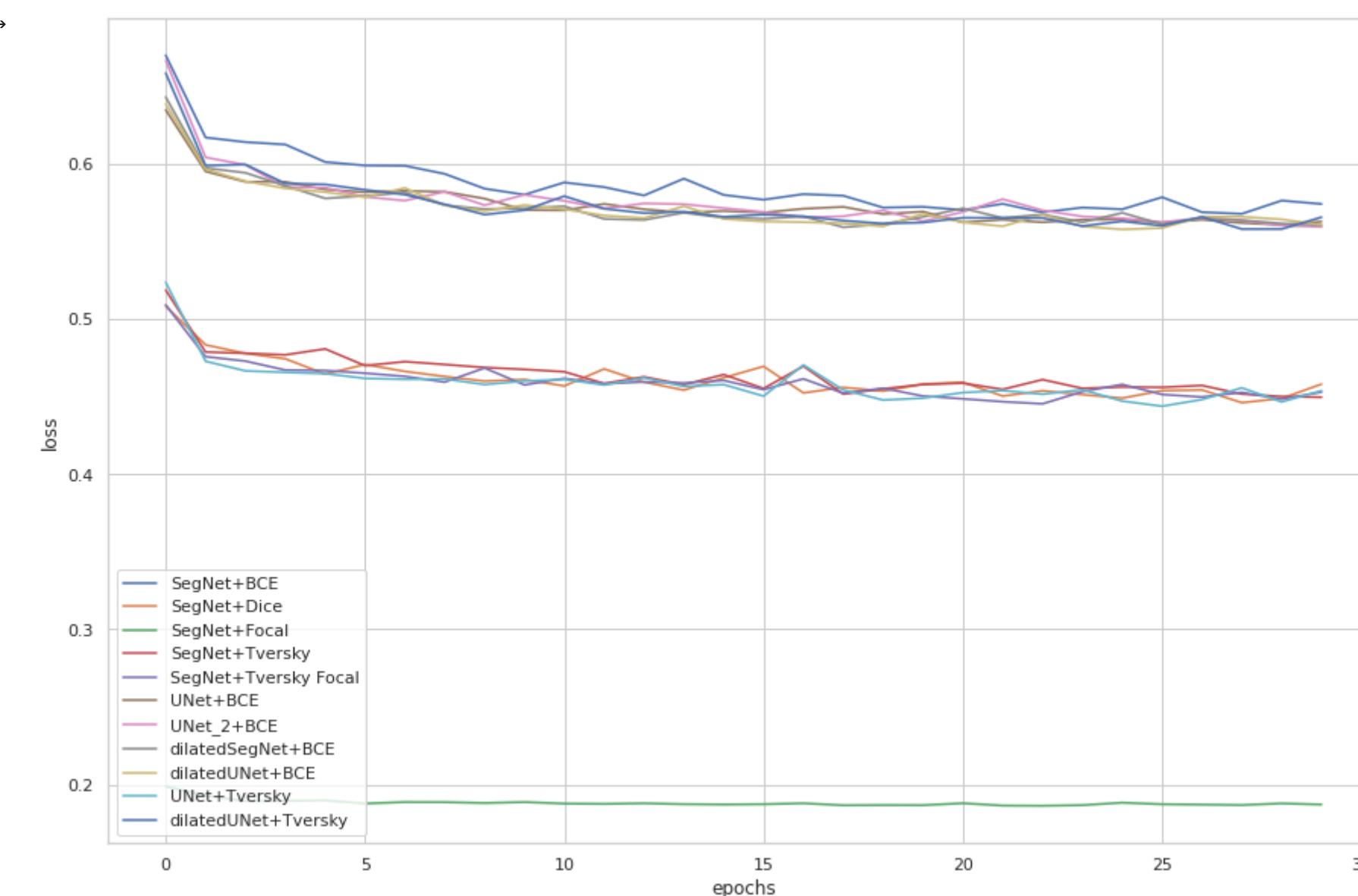
В ходе работы были проведены эксперименты с различными связками архитектур нейронных сетей и loss функций. Ввиду малого размера датасета результаты были нестабильны и часто существенно отличались, что обусловлено начальной инициализации.

Для решения этой проблемы были подобраны параметры обучения.

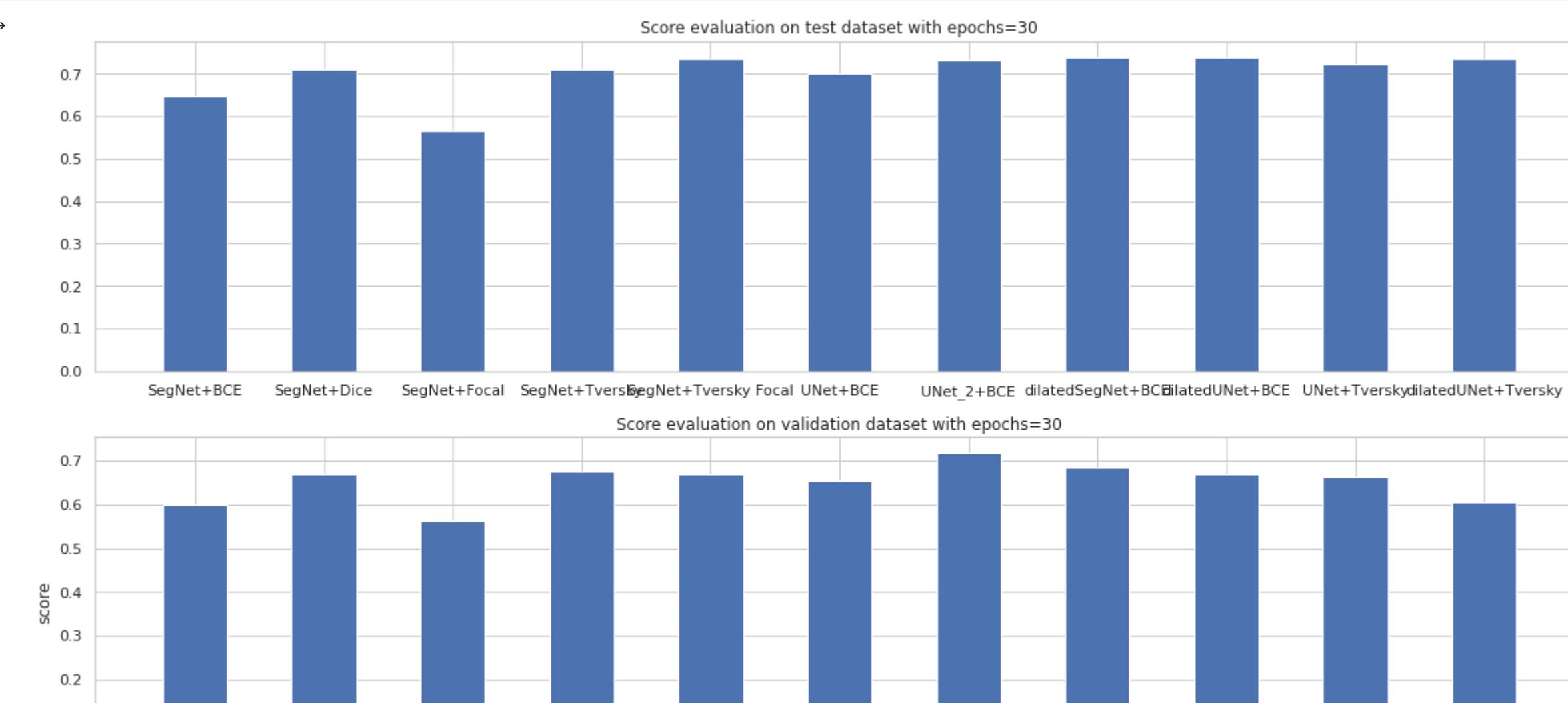
Параметры обучения:

- Размер батчей был ограничен 10, чтобы избежать переполнения памяти
- Количество эпох было выбрано равным 30, что обусловлено экспериментами, которые показали, что повышение их количества не слишком существенно влияет на конечный результат, а понижение приводит к нестабильным результатам
- В качестве оптимизатора используется Adam с параметром learning rate равным 0.0001, что обеспечивает повышение точности работы модели при сохранении удовлетворительной скорости обучения

```
1 plot_loss(loss_arr, names)
```



```
1 plot_score(scores_test_checkout, scores_val_checkout, names)
```



Выделим ТОП-3 архитектур

для **тестовой** части датасета.

```
1 top3_test = sorted(zip(scores_test_checkout, names), reverse=True)[:3]
2 for i, item in enumerate(top3_test):
3     print(str(i+1) + ' ' + str(item[1]) + '\nTest score = ' + str(item[0]))
```

```
1) dilatedUNet+BCE
Test score = 0.7400000333786011
2) dilatedSegNet+BCE
Test score = 0.737999995231629
3) SegNet+Tversky Focal
Test score = 0.7360000371932983
```

Для **валидационной** части датасета.

```
1 top3_val = sorted(zip(scores_val_checkout, names), reverse=True)[:3]
2 for i, item in enumerate(top3_val):
```

```
3     print(str(i+1) + ' ' + str(item[1]) + '\nTest score = ' + str(item[0]))
```

1) UNet_2+BCE
Test score = 0.7180000066757202
2) dilatedSegNet+BCE
Test score = 0.6859999895095825
3) SegNet+Tversky
Test score = 0.6740000247955322

Таким образом, на тестовой выборке наибольшую точность показала связка **Dilated UNet + BCE Loss** при текущем обучении моделей. Несмотря на нестабильность результатов при иных попытках обучения, которые не вошли в финальный вариант, стоит отметить стабильное попадание в ТОП-3 моделей с Dilated свертками и архитектуры UNet. Что касается выбора Loss функции, четкого фаворита выделить не удалось ввиду вышеупомянутой нестабильности. Тем не менее, архитектуры с BCE всегда занимали места в ТОП-3.

Ввиду таких результатов, для сегментации биомедицинских снимков в первую очередь следует пробовать применять модель UNet с различными параметрами.