



Deep Learning School

Школа глубокого обучения ФПМИ МФТИ

Домашнее задание. Базовый поток. Весна 2021

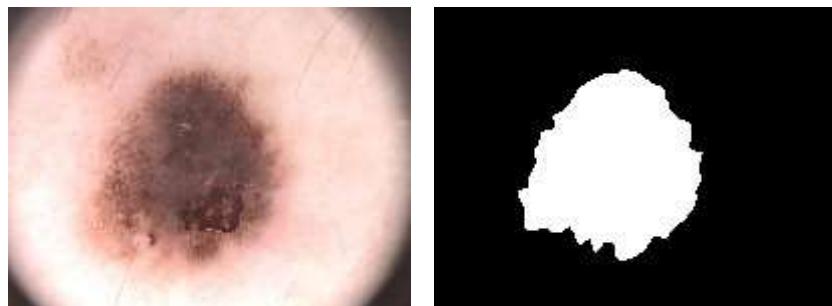
Сегментация изображений

В этом задании вам предстоит решить задачу сегментации медицинских снимков. Часть кода с загрузкой данных написана за вас. Всю содержательную сторону вопроса вам нужно заполнить самостоятельно. Задание оценивается из 15 баллов.

Обратите внимание, что отчёт по заданию стоит целых 6 баллов. Он вынесен в отдельный пункт в конце тетради. Это сделано для того, чтобы тетрадь была оформлена как законченный документ о проведении экспериментов. Неотъемлемой составляющей отчёта является ответ на следующие вопросы:

- Что было сделано? Что получилось реализовать, что не получилось?
- Какие результаты ожидалось получить?
- Какие результаты были достигнуты?
- Чем результаты различных подходов отличались друг от друга и от бейзлайна (если таковой присутствует)?

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 [62]:

```
#! wget https://www.dropbox.com/s/k88qukc20Ljnbuo/PH2Dataset.rar
#! wget https://www.dropbox.com/s/8LqrLoi0mxj2acu/PH2Dataset.rar
!gdown https://drive.google.com/uc?id=1D5kZvea81dGRnmCXXoLNJ4TL-MDksti7 -O PH2Dataset.rar

#альтернативная ссылка на данные: https://www.dropbox.com/s/8LqrLoi0mxj2acu/PH2Dataset.rar
```

Downloading...

From: <https://drive.google.com/uc?id=1D5kZvea81dGRnmCXXoLNJ4TL-MDksti7>
To: /content/PH2Dataset.rar
100% 116M/116M [00:00<00:00, 163MB/s]

In [63]:

```
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/
    ...
    ...
```

Здесь X.bmp — изображение, которое нужно сегментировать, X_lesion.bmp — результат сегментации.

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

In [64]:

```
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 256×256 пикселей.

Для изменения размера изображений можно использовать

`skimage.transform.resize()`. Эта функция также автоматически нормализует изображения в диапазоне [0, 1].

In [65]:

```

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 [66]:

```

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 [67]:

```
len(lesions)
```

Out[67]:

200

Чтобы убедиться, что все корректно, мы нарисуем несколько изображений

In [68]:

```

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 обучения, валидации и теста

соответственно

```
In [69]: ix = np.random.choice(len(X), len(X), False)
tr, val, ts = np.split(ix, [100, 150])
```

```
In [70]: print(len(tr), len(val), len(ts))
```

100 50 50

PyTorch DataLoader

```
In [71]: import torch
from torch.utils.data import DataLoader
batch_size = 20
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)
```

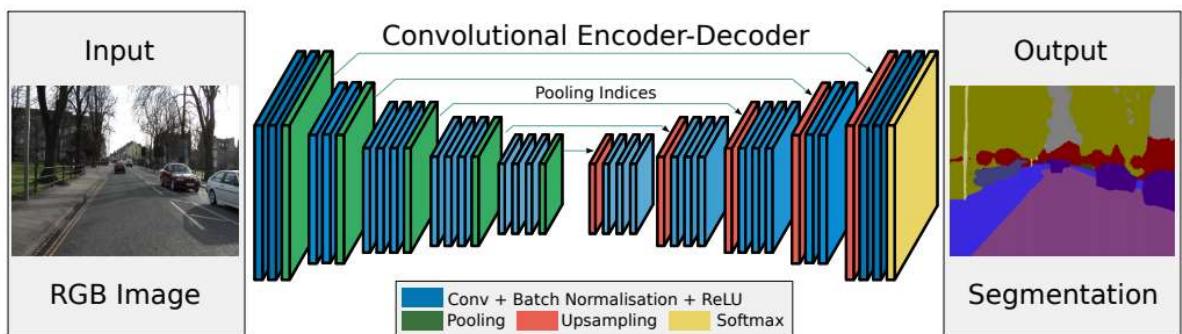
```
In [72]: import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

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 [73]:  
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  
  
from matplotlib import rcParams  
rcParams['figure.figsize'] = (15,4)
```

```
In [13]:  
class SegNet(nn.Module):  
    def __init__(self):  
        super().__init__()  
  
        # encoder (downsampling)  
        # Each enc_conv/dec_conv block should look like this:  
        # nn.Sequential(  
        #     nn.Conv2d(...),  
        #     ... (2 or 3 conv layers with relu and batchnorm),  
        # )  
        # Энкодер построен на основе VGG16  
        self.enc_conv0 = nn.Sequential(  
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),  
            nn.BatchNorm2d(64),  
            nn.ReLU()  
        )  
        self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 2  
        self.enc_conv1 = nn.Sequential(  
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),  
            nn.BatchNorm2d(128),  
            nn.ReLU()  
        )  
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 1  
        self.enc_conv2 = nn.Sequential(  
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),  
            nn.BatchNorm2d(256),  
            nn.ReLU()  
        )  
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 64  
        self.enc_conv3 = nn.Sequential(  
            nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),  
            nn.BatchNorm2d(512),  
            nn.ReLU()  
        )  
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 32  
  
        # bottleneck  
        # Болтник - как бы мостик между энкодером и декодером  
        self.enc_conv4 = nn.Sequential(  
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),  
            nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),  
            nn.BatchNorm2d(512),  
            nn.ReLU()  
        )  
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True)  
        self.upsample4 = nn.MaxUnpool2d(kernel_size=2)
```

```

self.dec_conv4 = nn.Sequential(
    nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
    # но сумы последние 3 свёртки+пулинг энкодера + первые анпулинг+3 свёртки де
)

# decoder (upsampling)
# декодер делаем симметрично энкодеру
self.upsample0 = nn.MaxUnpool2d(kernel_size=2) # 16 -> 32
self.dec_conv0 = nn.Sequential(
    nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=512, out_channels=256, kernel_size=3, padding=1),
    nn.BatchNorm2d(256),
    nn.ReLU()
)
self.upsample1 = nn.MaxUnpool2d(kernel_size=2) # 32 -> 64
self.dec_conv1 = nn.Sequential(
    nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=256, out_channels=128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128),
    nn.ReLU()
)
self.upsample2 = nn.MaxUnpool2d(kernel_size=2) # 64 -> 128
self.dec_conv2 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU()
)
self.upsample3 = nn.MaxUnpool2d(kernel_size=2) # 128 -> 256
self.dec_conv3 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
    nn.Conv2d(in_channels=64, out_channels=1, kernel_size=3, padding=1),
    #nn.Softmax()
)

def forward(self, x):
    # связываем блоки слоев между собой:
    # encoder
    e0,ind_e0 = self.pool0(self.enc_conv0(x))
    e1,ind_e1 = self.pool1(self.enc_conv1(e0))
    e2,ind_e2 = self.pool2(self.enc_conv2(e1))
    e3,ind_e3 = self.pool3(self.enc_conv3(e2))

    # bottleneck
    e4,ind_e4 = self.pool4(self.enc_conv4(e3))

    d4 = self.dec_conv4(self.upsample4(e4,ind_e4))

    # decoder
    d0 = self.dec_conv0(self.upsample0(d4,ind_e3))
    d1 = self.dec_conv1(self.upsample1(d0,ind_e2))
    d2 = self.dec_conv2(self.upsample2(d1,ind_e1))
    d3 = self.dec_conv3(self.upsample3(d2,ind_e0)) # no activation
    return d3

```

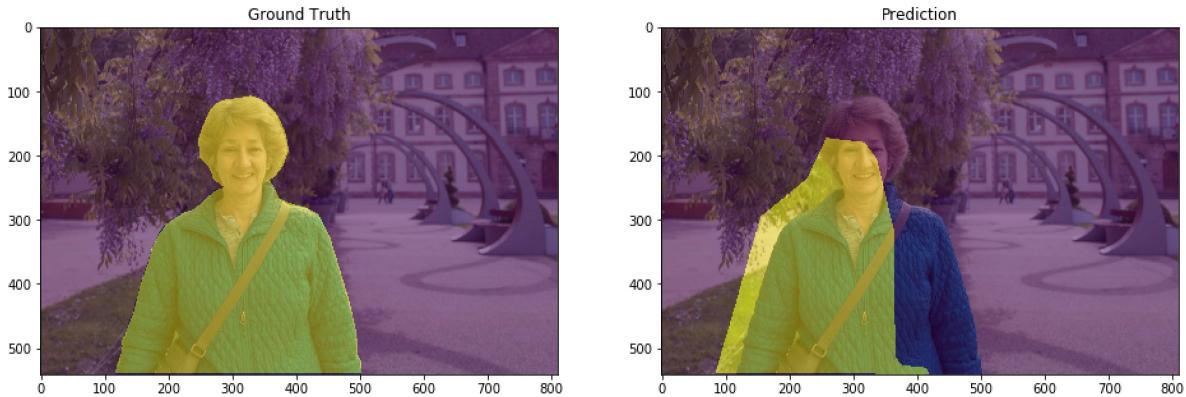
Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

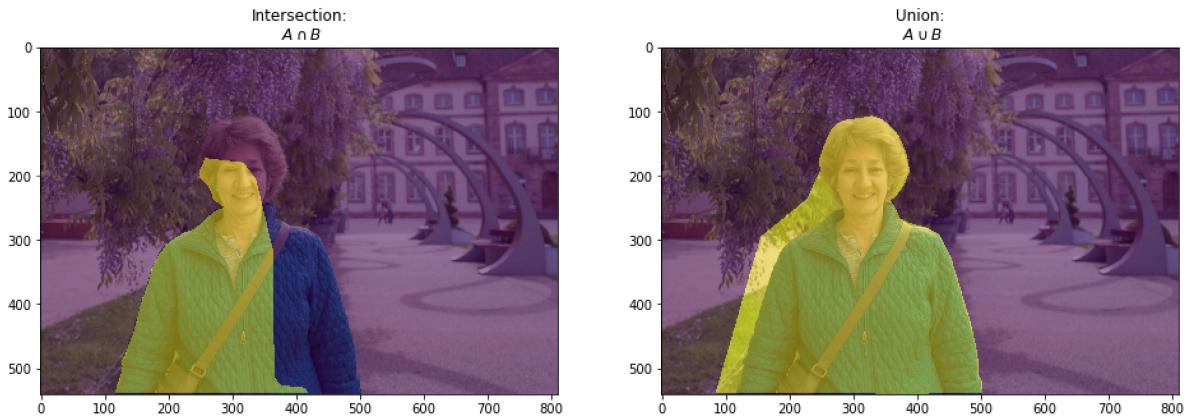
$$IoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup \text{prediction}}$$

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.

Для примера посмотрим на истину (слева) и предсказание (справа):



Тогда пересечение и объединение будет выглядеть так:



In [74]:

```
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
    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 saying if iou > 0.5 then 1 else 0
    return thresholded #
```

ФУНКЦИЯ ПОТЕРЬ [1 балл]

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](#). Эта функция эквивалентна первой и не так подвержена численной неустойчивости:

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

```
In [75]: def bce_loss(y_pred, y_real):
    #y_pred = torch.clamp(y_pred, 0, 1)
    loss = (y_pred - y_pred * y_real + torch.log(1 + torch.exp(-y_pred))).mean()
    return loss

# TODO
# Please don't use nn.BCELoss. write it from scratch
```

Тренировка [1 балл]

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

```
In [76]: def train(model, opt, loss_fn, epochs, data_tr, data_val):
    X_val, Y_val = next(iter(data_val))
    losses_train = []
    losses_val = []
    scores_train = []
    scores_val = []

    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:
            X_batch, Y_batch = X_batch.to(device), Y_batch.to(device) # data to device

            opt.zero_grad() # set parameter gradients to zero

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

            avg_loss += loss.item()

        losses_train.append(avg_loss / len(data_tr))
        scores_train.append(score(Y_val, Y_pred))

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

            Y_pred = model(X_batch.to(device))
            loss = loss_fn(Y_pred, Y_batch)

            avg_loss += loss.item()

        losses_val.append(avg_loss / len(data_val))
        scores_val.append(score(Y_val, Y_pred))

    return losses_train, losses_val, scores_train, scores_val
```

```

# calculate loss to show the user
    avg_loss += loss / len(data_tr)
toc = time()
print('loss: %f' % avg_loss)

# show intermediate results
model.eval() # testing mode
avg_loss_val = 0
#Y_hat = Y_pred.cpu().detach().numpy()

with torch.no_grad():
    Y_hat = model(X_val.to(device)).detach().cpu()

    loss = loss_fn(Y_hat, Y_val)
    avg_loss_val += loss / len(data_val)

toc = time()
print('loss_val: %f' % avg_loss_val)
losses_val.append(avg_loss_val)

avg_score_val = score_model(model, iou_pytorch, data_val)
scores_val.append(avg_score_val)

torch.cuda.empty_cache()

# Visualize tools
clear_output(wait=True)
for k in range(6):
    plt.subplot(2, 6, k+1)
    plt.imshow(np.rollaxis(X_val[k].numpy(), 0, 3), cmap='gray')
    plt.title('Real')
    plt.axis('off')

    plt.subplot(2, 6, k+7)
    plt.imshow(Y_hat[k, 0], cmap='gray')
    plt.title('Output')
    plt.axis('off')
plt.suptitle('%d / %d - loss: %f' % (epoch+1, epochs, avg_loss))
plt.show()

```

Инференс [1 балл]

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

In [77]:

```

def predict(model, data):
    model.eval() # testing mode
    Y_pred = [ X_batch for X_batch, _ in data]
    return np.array(Y_pred)

```

In [78]:

```

def score_model(model, metric, data):
    model.eval() # testing mode
    scores = 0
    for X_batch, Y_label in data:
        with torch.no_grad():
            Y_pred = torch.sigmoid(model(X_batch.to(device)))

```

```

Y_pred = torch.where(Y_pred >= 0.5, 1, 0)
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 []:

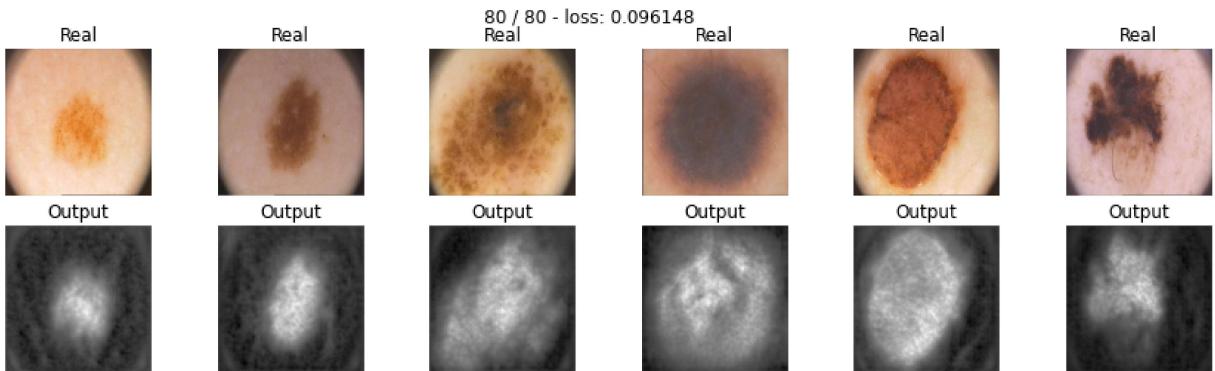
```
model = SegNet().to(device)
```

In []:

```

max_epochs = 80
#optim = torch.optim.SGD(model.parameters(), lr=1e-5, momentum=0.9, weight_decay=1e-5)
optim = torch.optim.Adam(model.parameters(), lr=3e-4, betas=(0.9, 0.999))
#optim = torch.optim.Adamax(model.parameters(), lr=0.00005)
train(model, optim, bce_loss, max_epochs, data_tr, data_val)

```



In []:

```
score_model(model, iou_pytorch, data_val)
```

Out[]:

```
0.7650000254313151
```

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 эпох, прежде чем бить тревогу попробуйте перебрать различные варианты и убедитесь, что во всех других сетапах сеть достигает желанного результата. СПОЙЛЕР: учиться она будет 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, возникающей в математической формуле.

In [79]:

```
def dice_loss(y_pred, y_real):
    smooth=1e-8

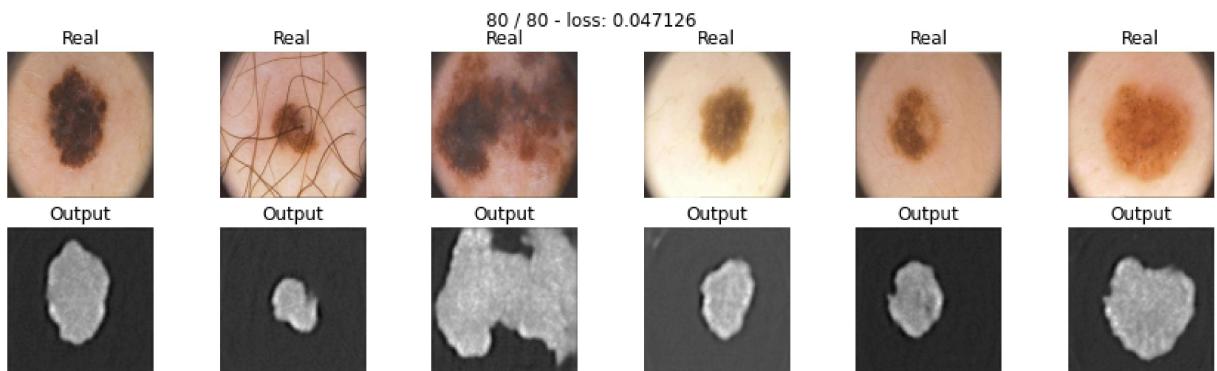
    y_pred = torch.sigmoid(y_pred)
    #y_pred = torch.where(y_pred >= 0.5, 1 , 0)
    #y_pred = torch.clamp(y_pred, 0, 1)
    intersection = torch.sum(y_pred * y_real, (1, 2, 3))
    num = 2 * intersection + smooth
    den = torch.sum(y_real, (1, 2, 3)) + torch.sum(y_pred, (1, 2, 3)) + smooth
    res = 1 - torch.mean(num / den)
    return res
```

Проводим тестирование:

In []:

```
model_dice = SegNet().to(device)

max_epochs = 80
optimaizer = optim.Adam(model_dice.parameters(), lr=1e-4, betas=(0.9, 0.99))
train(model_dice, optimaizer, dice_loss, max_epochs, data_tr, data_val)
```



In []:

```
score_model(model_dice, iou_pytorch, data_val)
```

Out[]:

```
0.7116667032241821
```

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.

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

Зафиксируем значение $\gamma = 2$.

In [80]:

```
def focal_loss(y_pred, y_real, eps = 1e-8, gamma = 2):
    alpha = 0.6
    y_pred = torch.sigmoid(y_pred)
    y_real = y_real.view(-1)
    y_pred = y_pred.view(-1)

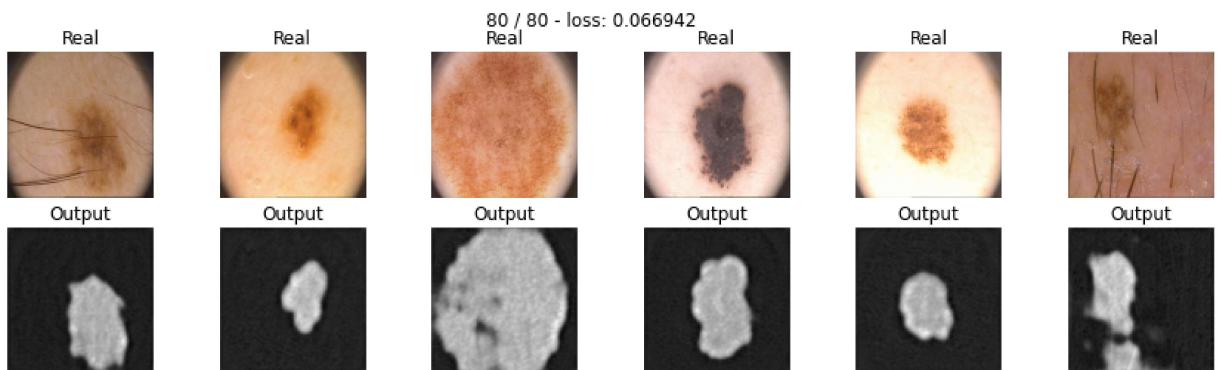
    BCE = bce_loss(y_pred, y_real)
    BCE_EXP = torch.exp(-BCE)
    focal_loss = alpha * (1-BCE_EXP)**gamma * BCE
    #y_pred = torch.sigmoid(y_pred) # hint: torch.clamp
    #y_pred = torch.clamp(min=eps)

    #Loss = y_real*torch.log(y_pred) + (1 - y_real) * torch.log(1 - y_pred)
    #Loss = Loss * (1 - y_pred)**gamma
    #your_loss = Loss.mean()
    your_loss = focal_loss.mean()
    return your_loss
```

In [60]:

```
model_focal = SegNet().to(device)

max_epochs = 80
optimaizer = optim.Adam(model_focal.parameters(), lr=1e-4, betas=(0.9, 0.999))
train(model_focal, optimaizer, focal_loss, max_epochs, data_tr, data_val)
```



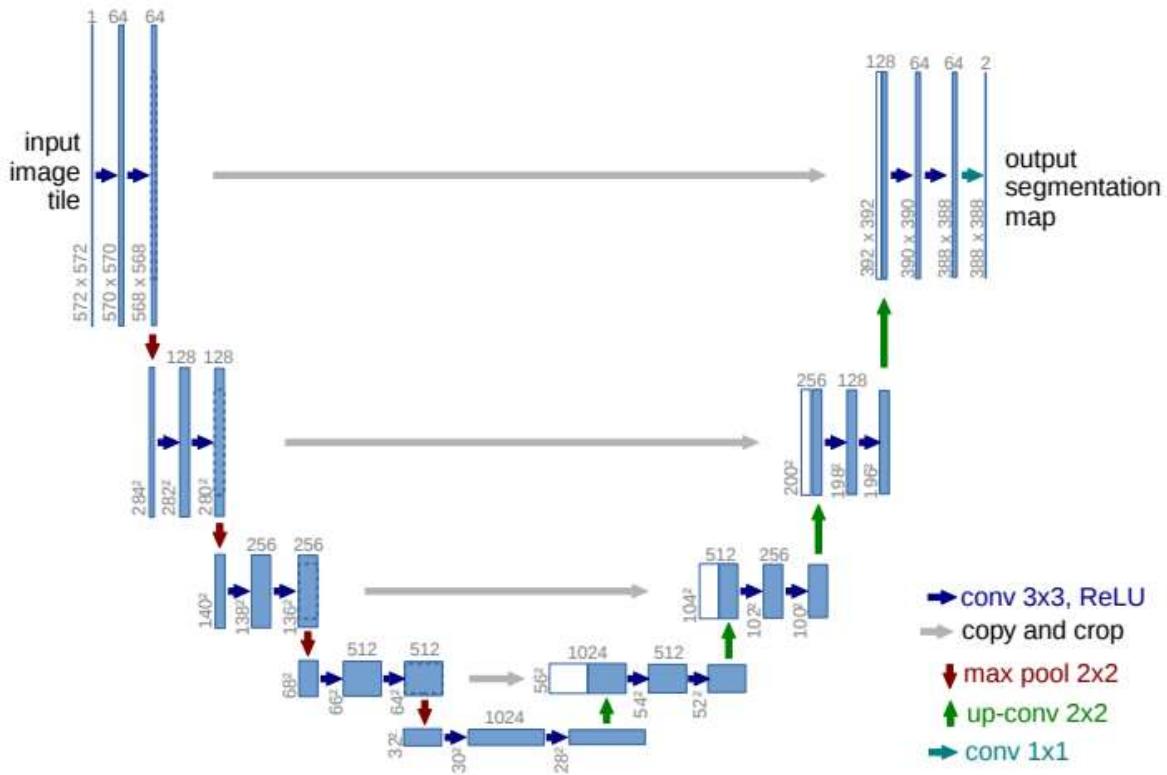
In [61]:

```
score_model(model_focal, iou_pytorch, data_val)
```

Out[61]: 0.7566666801770529

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. Получая на вход медицинское изображение, он выведет изображение в оттенках серого, где интенсивность каждого пикселя зависит от вероятности того, что этот пиксель принадлежит интересующей нас области.



У нас в архитектуре все так же существует энкодер и декодер, как в **SegNet**, но отличительной особенностью данной модели являются *skip-conenctions*, соединяющие части декодера и энкодера. 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 [85]:

```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        # Each enc_conv/dec_conv block should Look Like this:
        # nn.Sequential(
        #     nn.Conv2d(...),
        #     ... (2 or 3 conv Layers with relu and batchnorm),
        # )
        self.enc_conv0 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
```

```

        nn.ReLU()
    )
    self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 2
    self.enc_conv1 = nn.Sequential(
        nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
        nn.BatchNorm2d(128),
        nn.ReLU()
    )
    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 1
    self.enc_conv2 = nn.Sequential(
        nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
        nn.BatchNorm2d(256),
        nn.ReLU()
    )
    self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 64
    self.enc_conv3 = nn.Sequential(
        nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU()
    )
    self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True) # 32

    # bottleneck
    self.enc_conv4 = nn.Sequential(
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
        nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
        nn.BatchNorm2d(512),
        nn.ReLU()
    )
    self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True)
    self.upsample4 = nn.MaxUnpool2d(kernel_size=2)
    self.dec_conv4 = nn.Sequential(
        nn.ConvTranspose2d(in_channels=512, out_channels=512, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=512, out_channels=512, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=512, out_channels=512, kernel_size=2, stride=2),
        # no сумы последние 3 свёртки+пулинг энкодера + первые анпулинг+3 свёртки де
    )

```

decoder (upsampling)

```

    self.upsample0 = nn.MaxUnpool2d(kernel_size=2) # 16 -> 32
    self.dec_conv0 = nn.Sequential(
        nn.ConvTranspose2d(in_channels=512, out_channels=512, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=512, out_channels=512, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=2, stride=2),
        nn.BatchNorm2d(256),
        nn.ReLU()
    )
    self.upsample1 = nn.MaxUnpool2d(kernel_size=2) # 32 -> 64
    self.dec_conv1 = nn.Sequential(
        nn.ConvTranspose2d(in_channels=256, out_channels=256, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=256, out_channels=256, kernel_size=2, stride=2),
        nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=2, stride=2),
        nn.BatchNorm2d(128),
        nn.ReLU()
    )
    self.upsample2 = nn.MaxUnpool2d(kernel_size=2) # 64 -> 128
    self.dec_conv2 = nn.Sequential(
        nn.ConvTranspose2d(in_channels=128, out_channels=128, kernel_size=2, stride=2),

```

```

nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=2, stride=2
nn.BatchNorm2d(64),
nn.ReLU()
)
self.upsample3 = nn.MaxUnpool2d(kernel_size=2) # 128 -> 256
self.dec_conv3 = nn.Sequential(
nn.ConvTranspose2d(in_channels=64, out_channels=64, kernel_size=2, stride=2,
nn.ConvTranspose2d(in_channels=64, out_channels=1, kernel_size=2, stride=2,
)

def forward(self, x):
# encoder
e0,ind_e0 = self.pool0(self.enc_conv0(x))
e1,ind_e1 = self.pool1(self.enc_conv1(e0))
e2,ind_e2 = self.pool2(self.enc_conv2(e1))
e3,ind_e3 = self.pool3(self.enc_conv3(e2))

# bottleneck
e4,ind_e4 = self.pool4(self.enc_conv4(e3))

d4 = self.upsample4(e4,ind_e4)
x = torch.cat([d4, e4], dim=1)
d4 = self.dec_conv4(x)

# decoder
d0 = self.upsample0(d4,ind_e3)
x = torch.cat([d0, e3], dim=1)
d0 = self.dec_conv0(x)
d1 = self.upsample1(d0,ind_e2)
x = torch.cat([d1, e2], dim=1)
d1 = self.dec_conv1(x)
d2 = self.upsample2(d1,ind_e1)
x = torch.cat([d2, e1], dim=1)
d2 = self.dec_conv2(x)
d3 = self.upsample3(d2,ind_e0)
x = torch.cat([d3, e0], dim=1)
d3 = self.dec_conv3(x) # no activation
return d3

```

In [86]: unet_model_bce = UNet().to(device)

In [87]: train(unet_model_bce, optim.Adam(unet_model_bce.parameters()), bce_loss, 20, data_tr
* Epoch 1/20

```

-----  

RuntimeError                                     Traceback (most recent call last)  

<ipython-input-87-ad51c373782a> in <module>()  

----> 1 train(unet_model_bce, optim.Adam(unet_model_bce.parameters()), bce_loss, 20,  

data_tr, data_val)  

<ipython-input-76-8ccc4bc0888d> in train(model, opt, loss_fn, epochs, data_tr, data_<br/>
val)  

18  

19         # forward  

---> 20         Y_pred = model(X_batch.to(device))  

21         loss  = loss_fn(Y_pred, Y_batch)  

22         # forward-pass  

/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py in _call_impl(sel<br/>
f, *input, **kwargs)
```

```

1100             if not (self._backward_hooks or self._forward_hooks or self._forward_
_pre_hooks or _global_backward_hooks
1101                     or _global_forward_hooks or _global_forward_pre_hooks):
-> 1102                 return forward_call(*input, **kwargs)
1103             # Do not call functions when jit is used
1104             full_backward_hooks, non_full_backward_hooks = [], []

```

```

<ipython-input-85-5c7f0dcf17e6> in forward(self, x)
99
100         d4 = self.upsample4(e4, ind_e4)
--> 101         x = torch.cat([d4, e4])
102         d4 = self.dec_conv4(x)
103

```

`RuntimeError: Sizes of tensors must match except in dimension 0. Expected size 16 but got size 8 for tensor number 1 in the list.`

In []: `score_model(unet_model_bce, iou_pytorch, data_val)`

Теперь проверьте модель UNet с функцией потерь FocalLoss.

In []:

Сделайте вывод, какая из моделей лучше.

Отчет (6 баллов)

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

Неотъемлемой составляющей отчёта является ответ на следующие вопросы:

- Что было сделано? Что получилось реализовать, что не получилось?
- Какие результаты ождалось получить?
- Какие результаты были достигнуты?
- Чем результаты различных подходов отличались друг от друга и от бейзлайна (если таковой присутствует)?

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 достаточно суждений по поводу вашей работы.
- Я проверял метрики на трейне, и привел в результате таблицу с N числами, что не так? Ключевой момент тут, что вы измеряли на трейне ваши метрики, уверены ли вы, что зависимости 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? Естественно 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 сравнение 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 []:

При выполнении домашнего задания, создал модель Segnet и испытал на трех лосс-функциях. Найлучший результат показала BCE_loss - 76%, на втором месте по качеству focal_loss - 75%, ну и последнее место - Dice_loss с 71%. При испытании модели пробовал менять следующие параметры: Размер батча(с 25 до 20), Число эпох (пробовал от 20 до 80), Оптимизаторы(SGD , Adamax и Adam. Последний показал существенно лучший результат и потому продолжил далее экспериментировать с параметрами этого оптимизатора (LR и Betas) Лучшее -подобранные оставил в коде в последней редакции. Кроме того у focal_loss подбирал параметр alpha от 0,1 до 1. Лучший результат модель показала при значении 0.6 К сожалению из-за того, что не уложился в положенное время, не успеваю физически выполнить U-net модель(недоделал немного) и также не успеваю оформить в виде графиков отчет.

В целом задание было сложнее чем предыдущие, но гораздо интереснее, особенно, когда достигаешь определенных успехов при написании кода... Благодарю за понимание!