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.

```
# !gdown https://www.dropbox.com/s/8lqrloi0mxj2acu/PH2Dataset.rar -O
PH2Dataset.rar

# 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/
    ...
    ...
```

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

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

for root, dirs, files in os.walk(os.path.join("drive/MyDrive",
'PH2Dataset')):
    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 $[0,1]$.

```
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]
```

```
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
```

```
X.shape, Y.shape
```

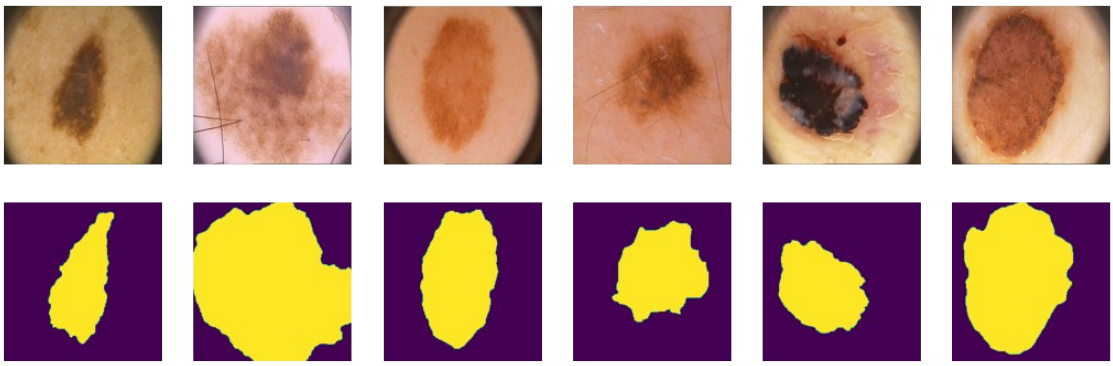
```
((200, 256, 256, 3), (200, 256, 256))
```

```
len(lesions)
```

```
200
```

Чтобы убедиться, что все корректно, мы нарисуем несколько изображений

```
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

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

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

```
100 50 50
```

PyTorch DataLoader

```
from torch.utils.data import DataLoader
batch_size = 10 # при 25 OutOfMemoryError: CUDA out of memory. Tried
to allocate 200.00 MiB (GPU 0; 14.76 GiB total capacity; 13.53 GiB
already allocated; 63.75 MiB free; 13.63 GiB reserved in total by
PyTorch) If reserved memory is >> allocated memory try setting
max_split_size_mb to avoid fragmentation.  See documentation for
Memory Management and PYTORCH_CUDA_ALLOC_CONF
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)

del X
del Y

import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# device = "cpu"
torch.cuda.empty_cache()
torch.cuda.memory_summary(device=device, abbreviated=False)
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.

```
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
```

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

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

```
        nn.ReLU(inplace=True),
        nn.Conv2d(512, 512, kernel_size=3, padding=1, stride=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True)
        )
    self.pool3 = nn.MaxPool2d(2,2,return_indices=True) # 32 -> 16

    # bottleneck
    self.bottleneck_conv = nn.Sequential(
        nn.Conv2d(512, 1024, kernel_size=1, padding=0, stride=1),
        nn.BatchNorm2d(1024),
        nn.ReLU(inplace=True),
        nn.Conv2d(1024, 512, kernel_size=1, padding=0, stride=1),
        nn.BatchNorm2d(512),
        nn.ReLU(inplace=True)
        )

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

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

    def forward(self, x):
        # encoder
        e0,ind = self.pool0(self.enc_conv0(x))
        e1,ind1 = self.pool1(self.enc_conv1(e0))
        e2,ind2 = self.pool2(self.enc_conv2(e1))
        e3,ind3 = self.pool3(self.enc_conv3(e2))

        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        d0 = self.dec_conv0(self.upsample0(b,ind3))
        d1 = self.dec_conv1(self.upsample1(d0,ind2))
        d2 = self.dec_conv2(self.upsample2(d1,ind1))
        d3 = self.dec_conv3(self.upsample3(d2,ind))
        # d3 =   # no activation
        return d3
```

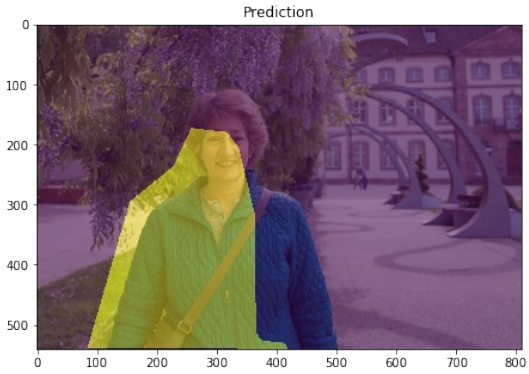
Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

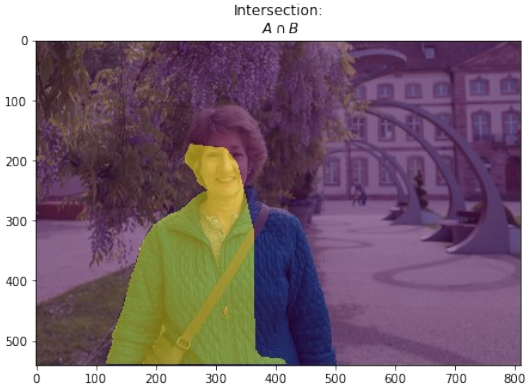
$$IoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup prediction}$$

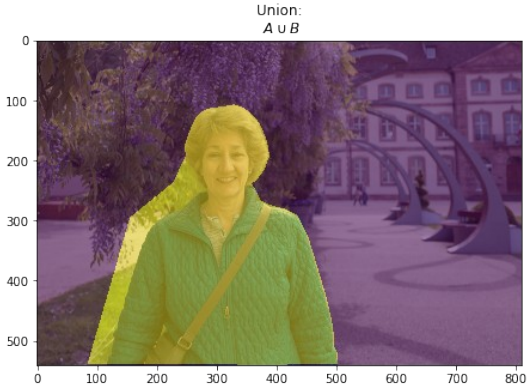
Пересечение (A ∩ B) состоит из пикселей, найденных как в маске предсказания, так и в основной маске истины, тогда как объединение (A ∪ B) просто состоит из всех пикселей, найденных либо в маске предсказания, либо в целевой маске.

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



Тогда пересечение и объединение будет выглядеть так:






```
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 > 0.5).byte()  # BATCH x 1 x H x W => BATCH x H
x W
    labels = (labels).byte()
    SMOOTH = 1e-8
    intersection = (outputs & labels).float().sum((1, 2))  # Will be
zero if Truth=0 or Prediction=0
    union = (outputs | labels).float().sum((1, 2))         # Will be
zzero if both are 0

    iou = (intersection + SMOOTH) / (union + SMOOTH)  # We smooth our
devision to avoid 0/0

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

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

где y это таргет желаемого результата и $\hat{y}$ является выходом модели. σ - это *логистическая* функция, который преобразует действительное число R в вероятность $[0, 1]$.

Однако эта потеря страдает от проблем численной нестабильности. Самое главное, что $\lim_{x \to 0} \log(x) = \infty$ приводит к неустойчивости в процессе оптимизации. Рекомендуется посмотреть следующее упрощение. Эта функция эквивалентна первой и не так подвержена численной неустойчивости:

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

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

def score_model(model, metric, data):
    model.eval()  # testing mode
    scores = 0
    for X_batch, Y_label in data:
        Y_pred = model(X_batch.cuda())
        scores += metric(Y_pred.to("cpu"),
Y_label.to("cpu")).mean().item()

    return scores/len(data)
```

Тренировка [1 балл]

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

```
def train(model, opt, loss_fn, epochs, data_tr, data_val):
    loses = []
    scores = {
```

```
        "train": [],
        "val": []
    }
    X_val, Y_val = next(iter(data_val))
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(opt, 'min')
    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
            # print(Y_batch.shape)
            Y_pred = model(X_batch)
            # print(Y_pred.shape)
            loss = loss_fn(Y_batch, Y_pred) # forward-pass
            loss.backward()  # backward-pass
            opt.step()  # update weights

            # calculate loss to show the user
            avg_loss += loss / len(data_tr)
        scheduler.step(loss)
        toc = time()

        # show intermediate results
        model.eval()  # testing mode
        # print(Y_pred.shape, Y_val.shape)

        Y_hat = (model(X_val.cuda()).to("cpu").detach()) > 0.5 #
detach and put into cpu
        X_hat = Y_val.to("cpu").detach()
        scores["val"].append(score_model(model, iou_pytorch,
data_val))
        scores["train"].append(score_model(model, iou_pytorch,
data_tr))
        # Visualize tools
        clear_output(wait=True)
        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(X_hat[k, 0], cmap='gray')
            plt.title('real')
            plt.axis('off')
        plt.suptitle('%d / %d - loss: %f' % (epoch+1, epochs,
avg_loss))
        plt.show()
        print('loss:', avg_loss, toc - tic)
        loses.append(float(avg_loss.detach()))
    return loses, scores
```

Инференс [1 балл]

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

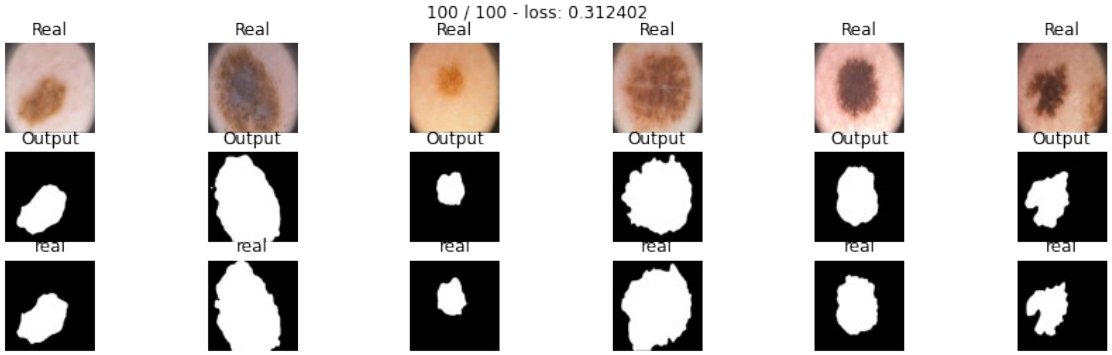
```
def predict(model, data):
    model.eval()  # testing mode
    Y_pred = [X_batch for X_batch, _ in data]
    return np.array(Y_pred)

def score_model(model, metric, data):
    model.eval()  # testing mode
    scores = 0
    for X_batch, Y_label in data:
        Y_pred = model(X_batch.cuda())
        scores += metric(Y_pred.to("cpu"),
Y_label.to("cpu")).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.

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

max_epochs = 100
optim_ = optim.Adam(model.parameters(), lr=1e-4, amsgrad=True)
```

```
loss = bce_loss
segnet_bce_loses, segnet_bce_scores = train(model, optim_, loss,
max_epochs, data_tr, data_val)
```




```
loss: tensor(0.3124, device='cuda:0', grad_fn=<AddBackward0>)
3.2246367931365967

score_model(model, iou_pytorch, data_val)

0.8483749985694885
```

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:

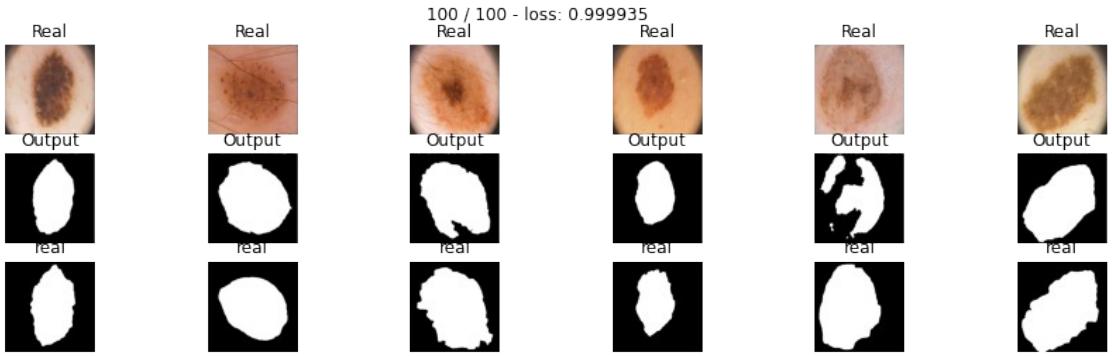
$$L_D(X,Y)=1-\frac{1}{256\times 256}\times\frac{\sum_i 2X_iY_i}{\sum_i (X_i+Y_i)}.$$

Не забудьте подумать о численной нестабильности, возникающей в математической формуле.

```
def dice_loss(y_real, y_pred):
    num = 1 / (256 * 256)
    den = torch.mean(2 * y_pred * y_real) / torch.mean(y_pred +
y_real)
    res = 1 - den * num
    return res
```

Проводим тестирование:

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

max_epochs = 100
optimaizer = optim.Adam(model_dice.parameters())
loses, scores = train(model_dice, optimaizer, dice_loss, max_epochs,
data_tr, data_val)
```




```
loss: tensor(0.9999, device='cuda:0', grad_fn=<AddBackward0>)
3.1933984756469727

score_model(model_dice, iou_pytorch, data_val)

0.7693515419960022

segnet_dice_scores = scores
segnet_dice_loses = loses
```

2. Focal loss:

Окей, мы уже с вами умеем делать BCE loss:

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

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:

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

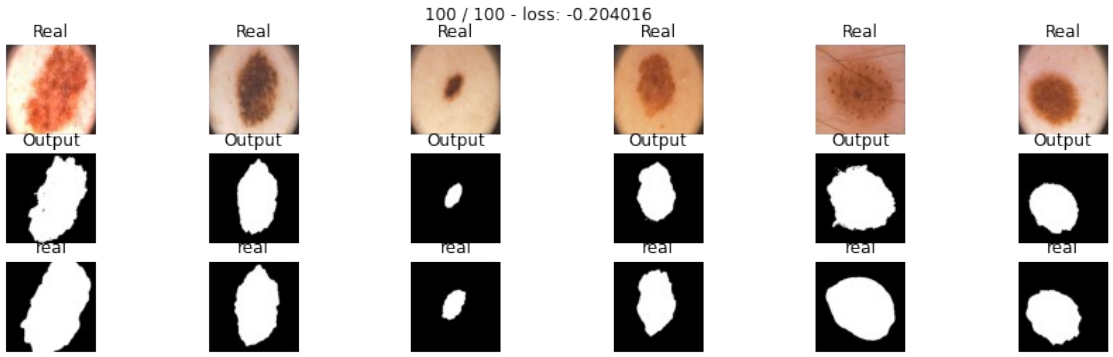
Традиционно вес α_i определяется как обратная частота класса этого пикселя i, так что наблюдения миноритарного класса весят больше по отношению к классу большинства.

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

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

Зафиксируем значение $\gamma = 2$.

```
def focal_loss(y_real, y_pred, eps = 1e-8, gamma = 2):
    p = torch.sigmoid(y_pred)
    loss = y_pred - y_real * y_pred + torch.log(1 + torch.exp(-
y_pred)) * ((1 - p) ** gamma)
    return torch.mean(loss)

model_focal = SegNet().to(device)

max_epochs = 100
optimaizer = optim.Adam(model_focal.parameters())
loses, scores = train(model_focal, optimaizer, focal_loss, max_epochs,
data_tr, data_val)
```




```
loss: tensor(-0.2040, device='cuda:0', grad_fn=<AddBackward0>)
3.195136070251465

 score_model(model_focal, iou_pytorch, data_val)

0.7833827853202819

segnet_focal_losss = loses
segnet_focal_scores = scores
```

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 особеностью 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.

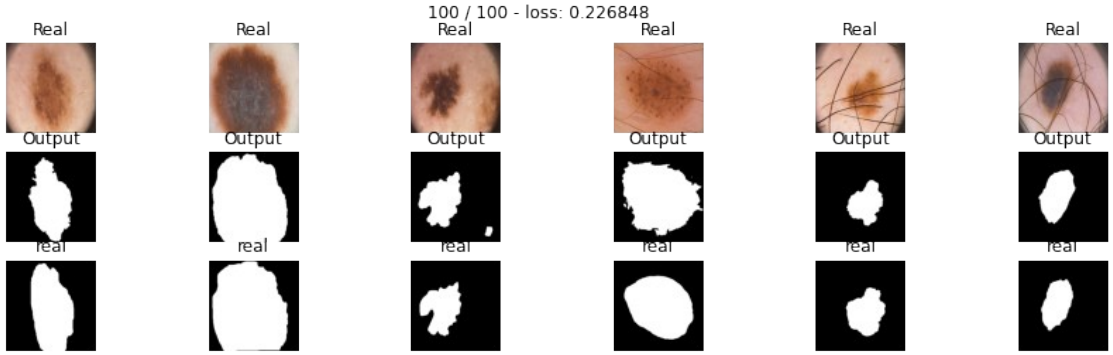
```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

        # 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(3, 64, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
            )
        self.pool0 = nn.MaxPool2d(kernel_size=2, stride=2) # 256 ->
128
        self.enc_conv1 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1,
bias=True),
```

```
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
            )
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128 -> 64
        self.enc_conv2 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
            )
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64 -> 32
        self.enc_conv3 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
            )
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32 -> 16

        # bottleneck
        self.bottleneck_conv = nn.Sequential(
            nn.Conv2d(512, 1024, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(1024),
            nn.ReLU(inplace=True),
            nn.Conv2d(1024, 1024, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(1024),
            nn.ReLU(inplace=True)
            )

        # decoder (upsampling)
        self.upsample0 = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(1024, 512, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
```

```
        ) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
            nn.Conv2d(1024, 512, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
            )
        self.upsample1 = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        ) # 32 -> 64
        self.dec_conv1 = nn.Sequential(
            nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
            )
        self.upsample2 = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        ) # 64 -> 128
        self.dec_conv2 = nn.Sequential(
            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
            )
        self.upsample3 = nn.Sequential(
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(64),
```

```
            nn.ReLU(inplace=True)
        ) # 128 -> 256
        self.dec_conv3 = nn.Sequential(
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1,
bias=True),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
            )
        self.out = nn.Sequential(
            nn.Conv2d(64, 1, kernel_size=1, stride=1, padding=0),
            nn.BatchNorm2d(1)
        )

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

        # print(e3.shape)
        # bottleneck
        b = self.bottleneck_conv(self.pool3(e3))
        # print(b.shape)
        # decoder

        d0 = self.dec_conv0(torch.cat((e3, self.upsample0(b)), dim=1))
        d1 = self.dec_conv1(torch.cat((e2, self.upsample1(d0)),
dim=1))
        d2 = self.dec_conv2(torch.cat((e1, self.upsample2(d1)),
dim=1))
        d3 = self.dec_conv3(torch.cat((e0, self.upsample3(d2)),
dim=1))

        return self.out(d3) # no activation

# torch.cuda.empty_cache()

unet_model_bce = UNet().to(device)
```

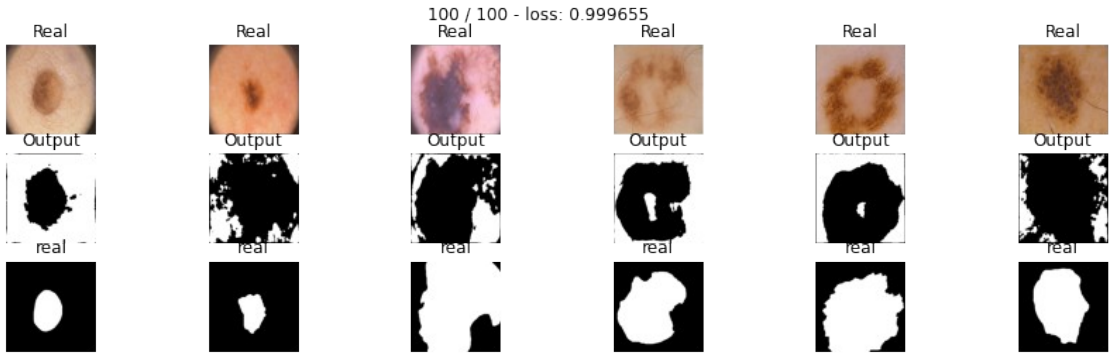
```
loses, scores = train(unet_model_bce,
optim.Adam(unet_model_bce.parameters()), bce_loss, 100, data_tr,
data_val)
```



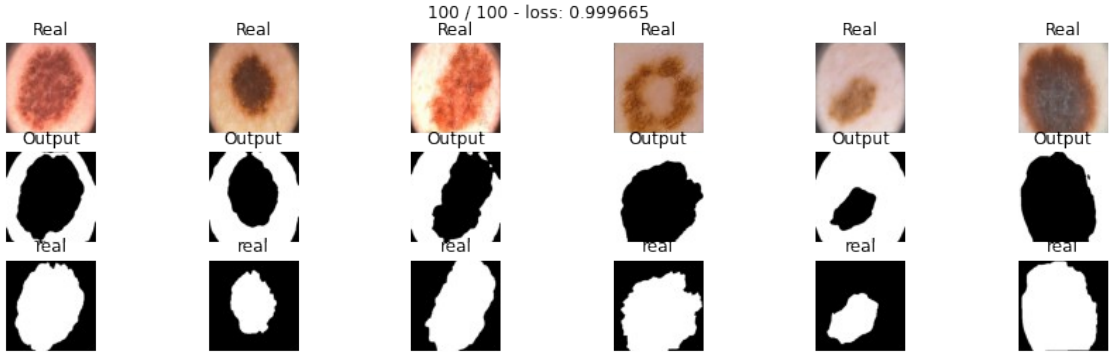

```
loss: tensor(0.2268, device='cuda:0', grad_fn=<AddBackward0>)
7.160990476608276

score_model(unet_model_bce, iou_pytorch, data_val)

0.8284999847412109

unet_bce_loses = loses
unet_bce_scores = scores
```

Теперь проверьте модель UNet с функцией потерь FocalLoss.

```
loses, scores = train(unet_model_bce,
optim.Adam(unet_model_bce.parameters()), dice_loss, 100, data_tr,
data_val)
```




```
loss: tensor(0.9997, device='cuda:0', grad_fn=<AddBackward0>)
7.170672655105591

unet_dice_loses = loses
unet_dice_scores = scores

usen_focal_loses, unet_focal_scores = train(unet_model_bce,
optim.Adam(unet_model_bce.parameters()), dice_loss, 100, data_tr,
data_val)
```

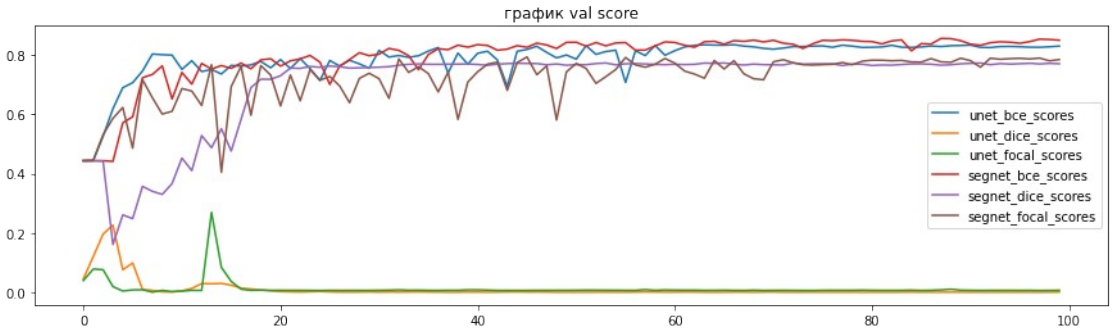



```
loss: tensor(0.9997, device='cuda:0', grad_fn=<AddBackward0>)
7.169709205627441
```

Сделайте вывод, какая из моделей лучше.

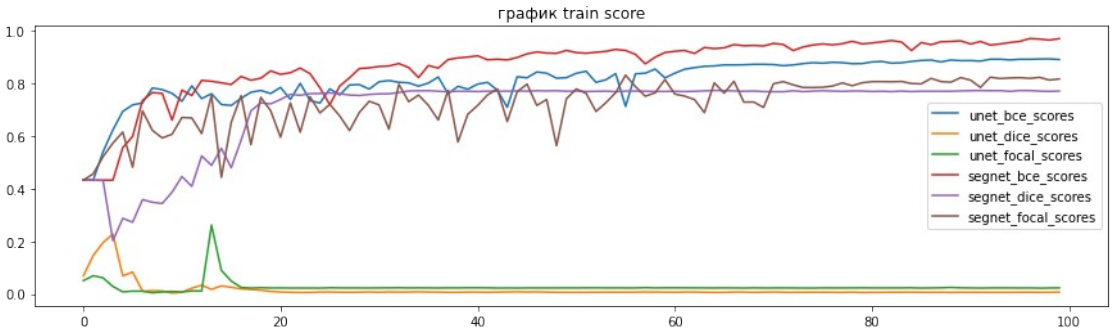
```
plt.title("график val score")
plt.plot(unet_bce_scores["val"])
plt.plot(unet_dice_scores["val"])
plt.plot(unet_focal_scores["val"])
plt.plot(segnet_bce_scores["val"])
plt.plot(segnet_dice_scores["val"])
plt.plot(segnet_focal_scores["val"])
plt.legend(['unet_bce_scores', 'unet_dice_scores',
"unet_focal_scores", "segnet_bce_scores", "segnet_dice_scores",
"segnet_focal_scores"])
```

```
<matplotlib.legend.Legend at 0x7fdc562d5c10>
```



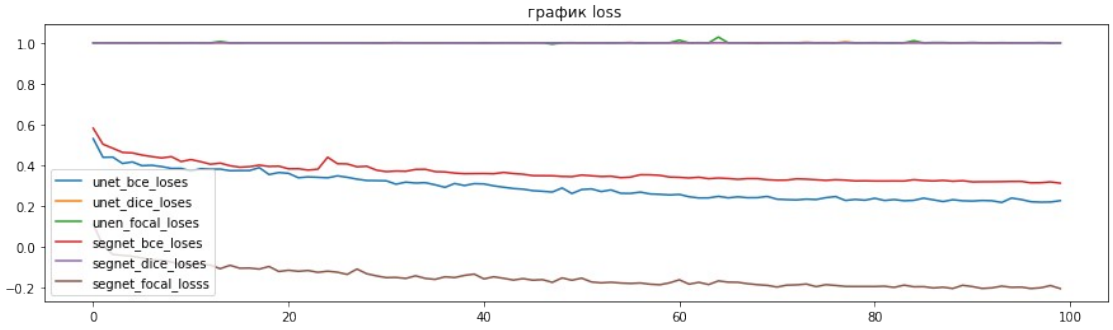

```
plt.title("график train score")
plt.plot(unet_bce_scores["train"])
plt.plot(unet_dice_scores["train"])
plt.plot(unet_focal_scores["train"])
plt.plot(segnet_bce_scores["train"])
plt.plot(segnet_dice_scores["train"])
plt.plot(segnet_focal_scores["train"])
plt.legend(['unet_bce_scores', 'unet_dice_scores',
"unet_focal_scores", "segnet_bce_scores", "segnet_dice_scores",
"segnet_focal_scores"])
```

```
<matplotlib.legend.Legend at 0x7fdc50e3f4c0>
```




```
plt.title("график loss ")
plt.plot(unet_bce_loses)
plt.plot(unet_dice_loses)
plt.plot(usen_focal_loses)
plt.plot(segnet_bce_loses)
plt.plot(segnet_dice_loses)
plt.plot(segnet_focal_losss)
plt.legend(['unet_bce_loses', 'unet_dice_loses', "unen_focal_loses",
"segnet_bce_loses", "segnet_dice_loses", "segnet_focal_losss"])
```

```
<matplotlib.legend.Legend at 0x7fdc518b2c70>
```



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.

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)?

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 итогововое 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.

```
max(segnet_bce_scores["val"])
```

```
0.8543593287467957
```

Отчёт

Что было сделано:

- Реализована модель SegNet
- Реализована модель Unet
- BCELoss
- Функция для тренировки модели
- scheduler
- График лосса
- График score

Что не получилось реализовать:

- dice_loss, значение всегда близко к 0.9999, а с Unet цвета меняются местами
- focal_loss, просто работает не так, как нужно :(

Какие результаты ожидалось получить:

- Планировалось выполнить задание за пару дней, но в итоге это заняло намного больше времени, чем ожидалось
- Больше всего времени заняла разработка SegNet и функций лосса dice и focal (лоссы сделать так и не получилось)
- Лучший результат я ожидал получить от модели Unet с функцией лосса focal

Какие результаты были достигнуты:

- Реализованы все функции, кроме dice_loss и focal_loss
- лучшее значение score выдала модель SegNet с функцией лосса bce (~0.85)

Чем результаты различных подходов отличались друг от друга:

- среднее время обучения одного поколения SegNet составило ~3.1 секунды
- среднее время обучения одного поколения Unet составило ~7.1 секунды
- среднее значение dice лосса примерно равно единице (косяк)
- среднее значение focal лосса меньше нуля (косяк)

- средняя точность SegNet меньше средней точности Unet (косяк, скорее всего)

Вывод:

Мне удалось реализовать архитектуры SegNet и Unet с лоссом bce, но не удалось реализовать их с лоссами dice и focal (к сожалению, я не знаю почему функции лосса выдают неверный результат), лучшей точность показала, как видно на первом графике, модель SegNet с функцией лосса bce (~0.85), скорее всего, это связано с неправильной реализацией модели Unet, но ошибку так и не удалось выявить. Также, как видно на втором графике, корректно работает только функция лосса bce, функция лосса dice показывает результат приближённый к 1, а функция focal показывает результат меньше нуля. Основное время заняла реализация функций лосса и модели SegNet, реализация Unet заняла относительно мало времени.