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)?

Stepik: Shkarbanenko Mikhail 537953169

Telegram: @IInfiniteTsukuyomi

---

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.

Я использовал Kaggle notebook для работы, так как Kaggle предоставляет GPU на более длительный срок, чем Colab.

```
#!wget -c https://www.dropbox.com/s/k88qukc20ljnbuo/PH2Dataset.rar
```

```
!ls
```

```
__notebook_source__.ipynb
```

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()`

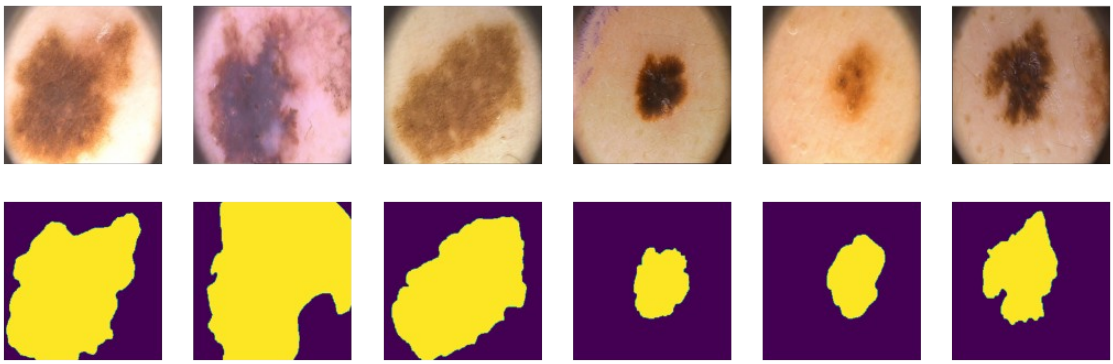
```
import pickle

images = []
lesions = []
from skimage.io import imread
import os
from pathlib import Path
root = '../input/ph2dataset/PH2Dataset/'

for d in Path(root+'PH2 Dataset images').iterdir():
    for imd in Path(d).iterdir():
        if str(imd).endswith('_Dermoscopic_Image'):
            images.append(imread(next(imd.iterdir())))
        if str(imd).endswith('_lesion'):
            lesions.append(imread(next(imd.iterdir())))
```

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

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 = 25
data_tr = DataLoader(list(zip(np.rollaxis(X[tr], 3, 1), Y[tr,
np.newaxis])),
                     batch_size=batch_size, shuffle=True,
num_workers=2)
data_val = DataLoader(list(zip(np.rollaxis(X[val], 3, 1), Y[val,
np.newaxis])),
                      batch_size=batch_size, shuffle=True,
num_workers=2)
data_ts = DataLoader(list(zip(np.rollaxis(X[ts], 3, 1), Y[ts,
np.newaxis])),
```

```
                    batch_size=batch_size, shuffle=True,
num_workers=2)
```

```
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.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import torch.optim as optim
from time import time
import copy

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

class SegNet(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder
        self.enc_conv0 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
```

```
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )
        self.pool0 = nn.MaxPool2d(2, stride=2, return_indices=True) #
256 -> 128

        self.enc_conv1 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )
        self.pool1 = nn.MaxPool2d(2, stride=2, return_indices=True) #
128 -> 64

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

        self.enc_conv3 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
        )
        self.pool3 = nn.MaxPool2d(2, stride=2, return_indices=True) #
32 -> 16

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

        # decoder (upsampling)
```

```
        self.upsample0 = nn.MaxUnpool2d(2, stride=2) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
            nn.Conv2d(512, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )

        self.upsample1 = nn.MaxUnpool2d(2, stride=2) # 32 -> 64
        self.dec_conv1 = nn.Sequential(
            nn.Conv2d(256, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )

        self.upsample2 = nn.MaxUnpool2d(2, stride=2) # 64 -> 128
        self.dec_conv2 = nn.Sequential(
            nn.Conv2d(128, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )

        self.upsample3 = nn.MaxUnpool2d(2, stride=2) # 128 -> 256
        self.dec_conv3 = nn.Sequential(
            nn.Conv2d(64, 1, kernel_size=3, stride=1, padding=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, i0 = self.pool0(self.enc_conv0(x))
        e1, i1 = self.pool1(self.enc_conv1(e0))
        e2, i2 = self.pool2(self.enc_conv2(e1))
        e3, i3 = self.pool3(self.enc_conv3(e2))

        # bottleneck
        b = self.bottleneck_conv(e3)
```

```
        # decoder

        d0 = self.dec_conv0(self.upsample0(b, i3))
        d1 = self.dec_conv1(self.upsample1(d0, i2))
        d2 = self.dec_conv2(self.upsample2(d1, i1))
        d3 = self.dec_conv3(self.upsample3(d2, i0)) # no activation

        return d3
```

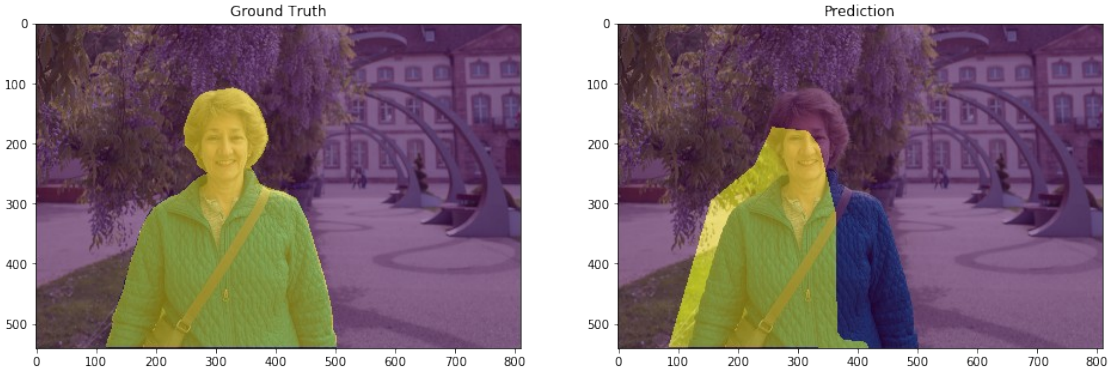
Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

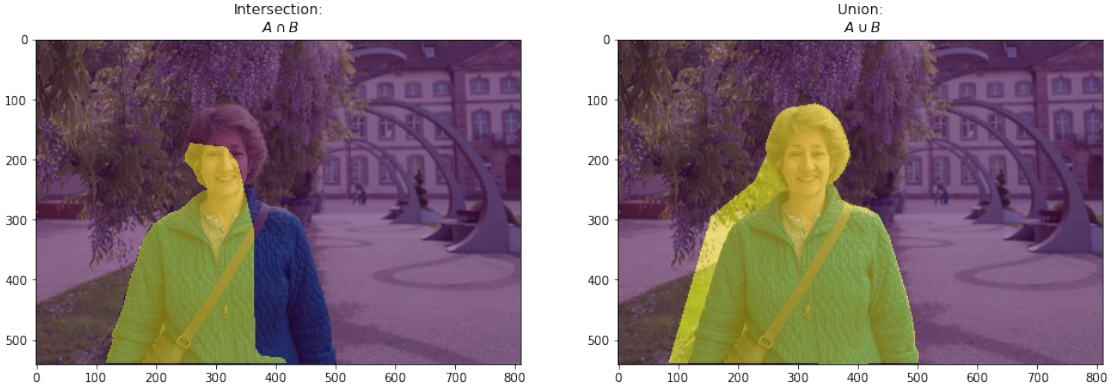
$$IoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup prediction}$$

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.squeeze(1).byte()  # BATCH x 1 x H x W => BATCH
x H x W
    labels = labels.squeeze(1).byte()
    SMOOTH = 1e-8
    intersection = (outputs & labels).float().sum((1, 2))  # Will be
zero if Truth=0 or 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:

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

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

Тренировка [1 балл]

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

```
def train(model, opt, loss_fn, metric_fn, epochs, data_tr, data_val):

    torch.cuda.empty_cache()

    X_val, Y_val = next(iter(data_val))
    train_losses = []
    val_losses = []
    train_metric_values = []
    val_metric_values = []
    best_val_score = 0
    best_model = None
    best_model_path = 'best_model.pt'

    for epoch in range(epochs):

        tic = time()
        print('* Epoch %d/%d' % (epoch+1, epochs))
        avg_loss = 0
        model.train() # train mode

        for X_batch, Y_batch in data_tr:

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

            # set parameter gradients to zero
            opt.zero_grad()

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

            # calculate loss to show the user
            avg_loss += loss / len(data_tr)

        train_losses.append(avg_loss.item())
        train_metric_values.append(score_model(model, metric_fn,
data_tr))
```

```
        model.eval()  # validation mode
        avg_loss = 0

        for X_val, Y_val in data_val:
            with torch.no_grad():
                Y_hat = model(X_val.to(device)).detach().cpu() #
detach and put into cpu
                loss = loss_fn(Y_hat, Y_val) # forward-pass
                avg_loss += loss / len(data_val)

        val_losses.append(avg_loss.item())
        val_metric_values.append(score_model(model, metric_fn,
data_val))

        if val_metric_values[epoch] > best_val_score:
            best_val_score = val_metric_values[epoch]
            torch.save(model.state_dict(), best_model_path)

        # 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()

    if model._get_name() == 'SegNet':
        best_model = SegNet().to(device)
    elif model._get_name() == 'UNet':
        best_model = UNet().to(device)
    elif model._get_name() == 'UNet2':
        best_model = UNet2().to(device)

best_model.load_state_dict(torch.load('/kaggle/working/best_model.pt')
)
    model = best_model

    model_data_dict = {
        'model_name' : model._get_name(),
        'loss_name' : loss_fn.__name__,
```

```
        'model' : model,
        'num_epochs' : epochs,
        'train_losses' : train_losses,
        'val_losses' : val_losses,
        'train_metric_values' : train_metric_values,
        'val_metric_values' : val_metric_values
    }

    torch.cuda.empty_cache()

    return  model_data_dict
```

Инференс [1 балл]

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

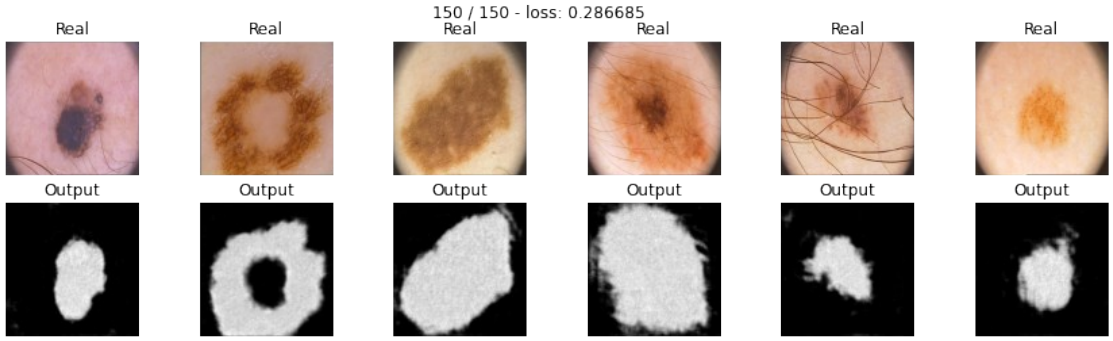
```
def predict(model, data):
    model.eval()  # testing mode
    Y_pred = [ X_batch for X_batch, _ in data]
    return np.array(Y_pred)

def score_model(model, metric, data):
    model.eval()  # testing mode
    scores = 0
    for X_batch, Y_label in data:
      with torch.no_grad():
        Y_pred = model(X_batch.to(device))
        Y_pred = Y_pred > 0.15 # выходы модели приводятся к маске из 0
и 1
        scores += metric(Y_pred, Y_label.to(device)).mean().item()

    return scores/len(data)
```

Основной момент: обучение

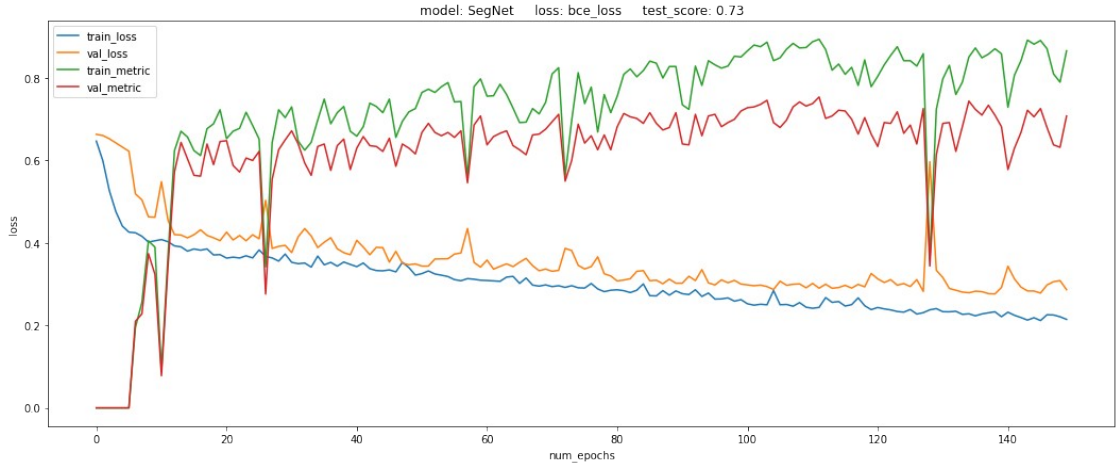
Обучите вашу модель. Обратите внимание, что обучать необходимо до сходимости. Если указанного количества эпох (20) не хватило, попробуйте изменять количество эпох до сходимости алгоритма. Сходимость определяйте по изменению функции потерь на валидационной выборке. С параметрами оптимизатора можно спокойно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

```
import pandas as pd
import seaborn as sns

def create_model(model, loss_fn, max_epochs=150):
    optim = torch.optim.AdamW(model.parameters(), lr=1e-3,
weight_decay=5e-2)
    model_data = train(model, optim, loss_fn, iou_pytorch, max_epochs,
```

```
data_tr, data_val)
    return model_data

def loss_plot(model_data):
    x_values = list(range(model_data['num_epochs']))
    model_name, loss_name, test_score = model_data['model_name'],
model_data['loss_name'], round(model_data['test_score'], 2)
    title = f'model: {model_name}     loss: {loss_name}
test_score: {test_score}'
    plt.figure(figsize=(18,7))
    plt.plot(x_values, model_data['train_losses'], label='train_loss')
    plt.plot(x_values, model_data['val_losses'], label='val_loss')
    plt.plot(x_values, model_data['train_metric_values'],
label='train_metric')
    plt.plot(x_values, model_data['val_metric_values'],
label='val_metric')
    plt.xlabel('num_epochs')
    plt.ylabel('loss')
    plt.title(title)
    plt.legend()
    plt.show()

models_data = {}

model, loss_fn = SegNet().to(device), bce_loss
model_name, loss_name = model._get_name(), bce_loss.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



Ответьте себе на вопрос: не переобучается ли моя модель?

Датасет маленький и вполне возможно, что модель может просто его запомнить и соответстенно переобучиться. Индикатором переобучения может служить медленно растущищй зазор между скор кривыми для val и train датасетов. Я экспериментировал с числом эпох и, как мне кажется, нашел оптимальное число - 150, позволяющее сохранить баланс между underfiting и overfiting.

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:

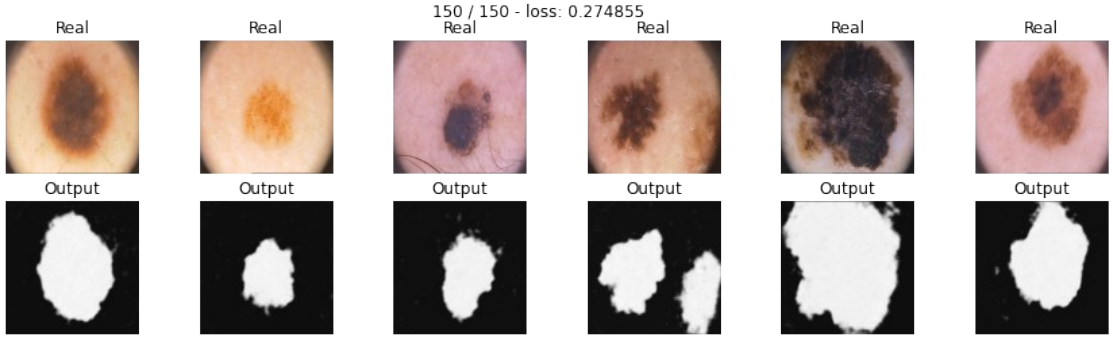
$$D(X,Y)=\frac{2|X\cap Y|}{|X|+|Y|}$$

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

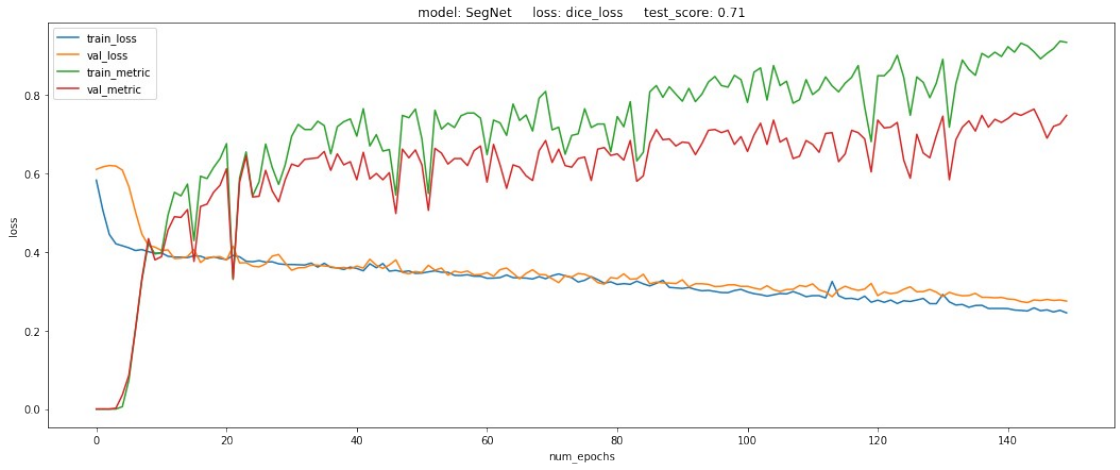
$$L_D(X,Y)=1-\frac{1}{256\times 256}\times\frac{\sum_i 2X_iY_i}{\sum_i X_i+Y_i}.$$

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

```
def dice_loss(y_pred, y_real):

    smooth = 1e-8
    real = y_real.squeeze(1)
    pred = y_pred.sigmoid().squeeze(1)
    res = 1 - ((2 * torch.sum(pred * real) + smooth) / (torch.sum(pred
+ real) + smooth))#/(256*256)

    return res

model, loss_fn = SegNet().to(device), dice_loss
model_name, loss_name = model._get_name(), loss_fn.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



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:

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

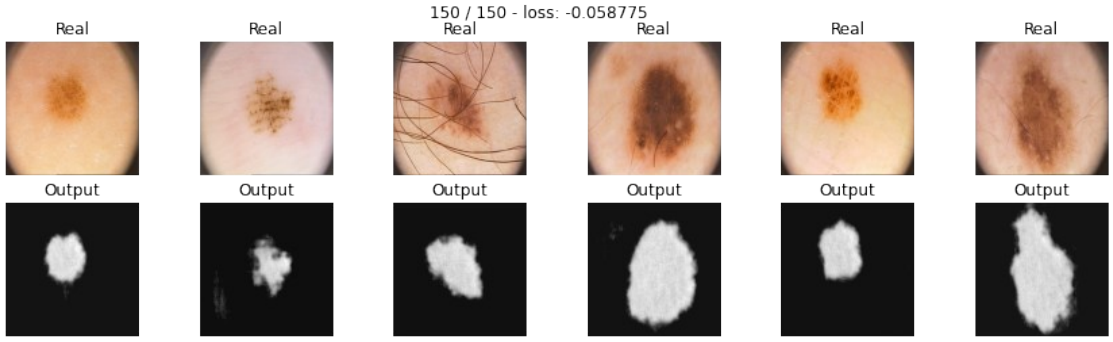
Традиционно вес α_i определяется как обратная частота класса этого пикселя i, так что наблюдения миноритарного класса весят больше по отношению к классу большинства.

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

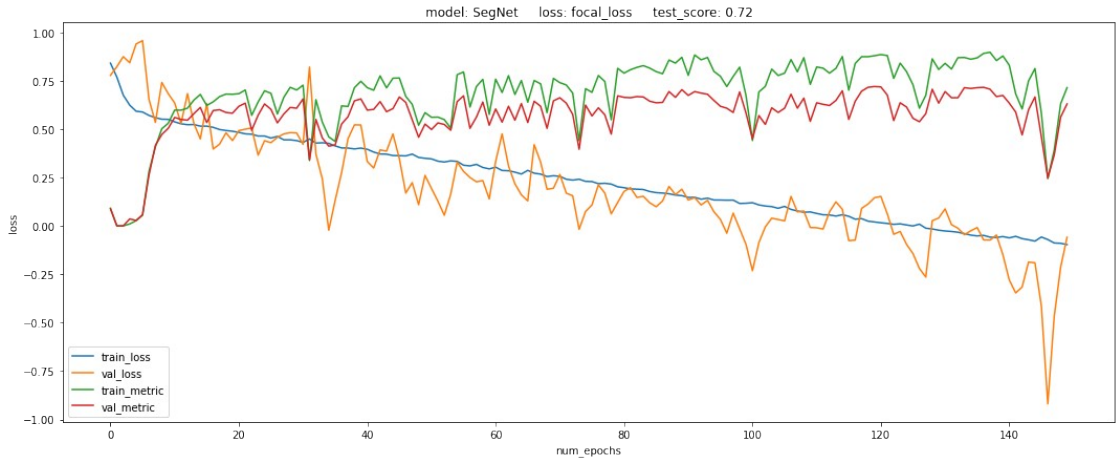
$$L_{focal}(y,\hat{y}) = -\sum_i \left[\left(1-\sigma(\hat{y}_i)\right)^{\gamma} y_i \log \sigma(\hat{y}_i) + (1-y_i) \log\left(1-\sigma(\hat{y}_i)\right)\right].$$

Зафиксируем значение $\gamma=2$.

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

model, loss_fn = SegNet().to(device), focal_loss
model_name, loss_name = model._get_name(), loss_fn.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



[BONUS] Мир сегментационных лоссов [5 баллов]

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

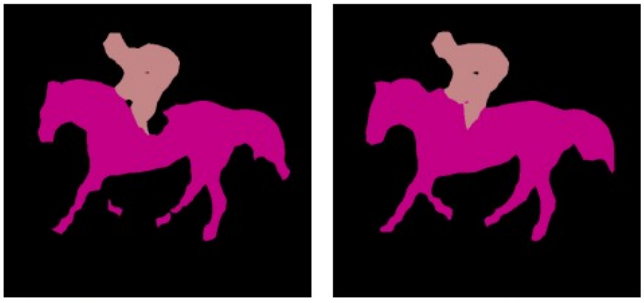
1) Можно учесть Total Variation 2) Lova 3) BCE но с Soft Targets (что-то типа label-smoothing для многослассовой классификации) 4) Любой другой

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

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

Я создал свою лосс функцию на основе статьи Correlation Maximized Structural Similarity Loss for Semantic Segmentation. Прикрепляю картинку из статьи, иллюстрирующую почему выбор пал именно на нее.

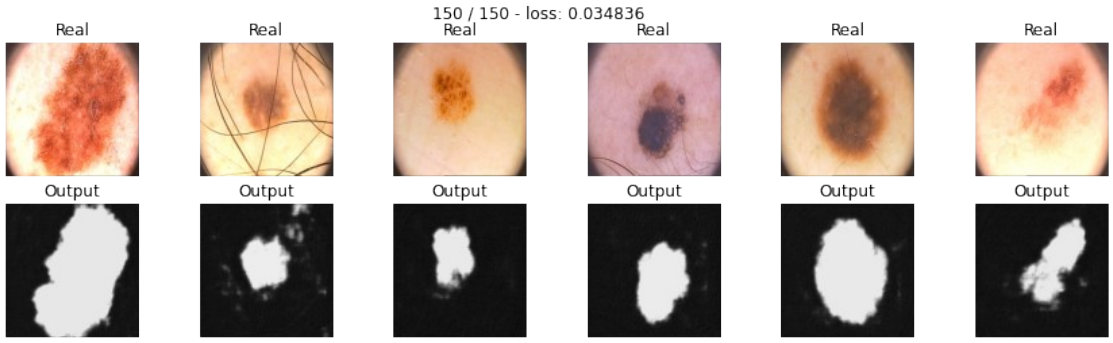
(a) Real-world image (b) Ground truth



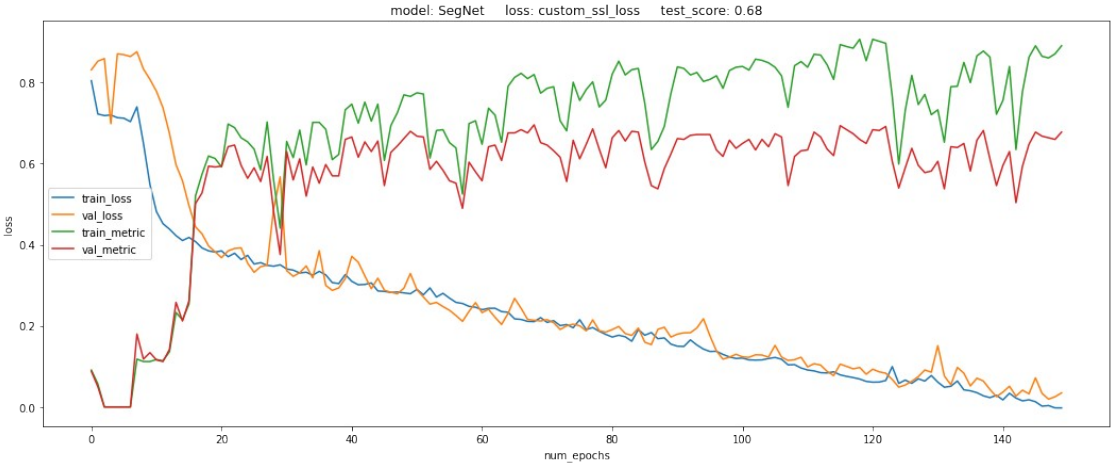
(c) Prediction with cross entropy (d) Prediction with proposed SSL

Авторы предлагают модернизацию бинарной кросс энтропии. В лосс функцию добавляется коэффициент корреляции между изображениями. В статье говорится, что это позволяет модели лучше понимать структуру изображения.

```
def custom_ssl_loss(y_real, y_pred, betta=0.1, lambda_coeff=0.8):

    smooth = 1e-6

    corr = (((y_real - y_real.mean() + smooth) / (y_real.std() +
smooth)) - ((y_pred - y_pred.mean() + smooth) / (y_pred.std() +
smooth))).abs()
    corr =  corr * (corr > betta * corr.max())
    initial_loss =  y_pred - y_real * y_pred + (1 + torch.exp(-
y_pred)).log()
    ssl_loss = corr * initial_loss
    final_loss = (lambda_coeff * initial_loss + (1 - lambda_coeff) *
ssl_loss).mean()

    return final_loss

model, loss_fn = SegNet().to(device), custom_ssl_loss
model_name, loss_name = model._get_name(), loss_fn.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



К сожалению, для данного датасета SegNet с ssl_loss показала наихудшее качество, хоть и разрыв с другими лосами небольшой. После 70-ой эпохи разрыв между кривыми качества на val и train выборке сильно растет, что сигнализирует о переобучении. Скорее всего способность модели запоминать структуру картинки на таком маленьком датасете сыграла в минус, а не в плюс.

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.

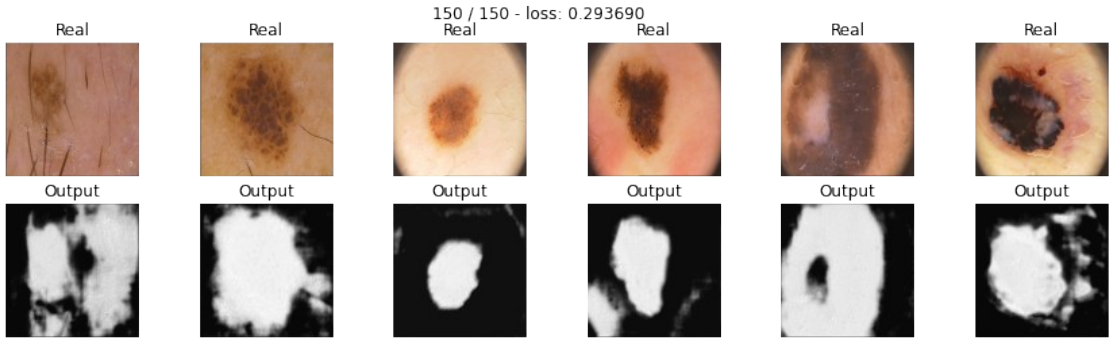
У нас в архитектуре все так же существует энкодер и декодер, как в **SegNet**, но отличительной особеностью данной модели являются *skip-conenctions*, соединяющие части декодера и энкодера. То есть для того чтобы передать на вход декодера тензор, мы конкатенируем симметричный выход с энкодера и выход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

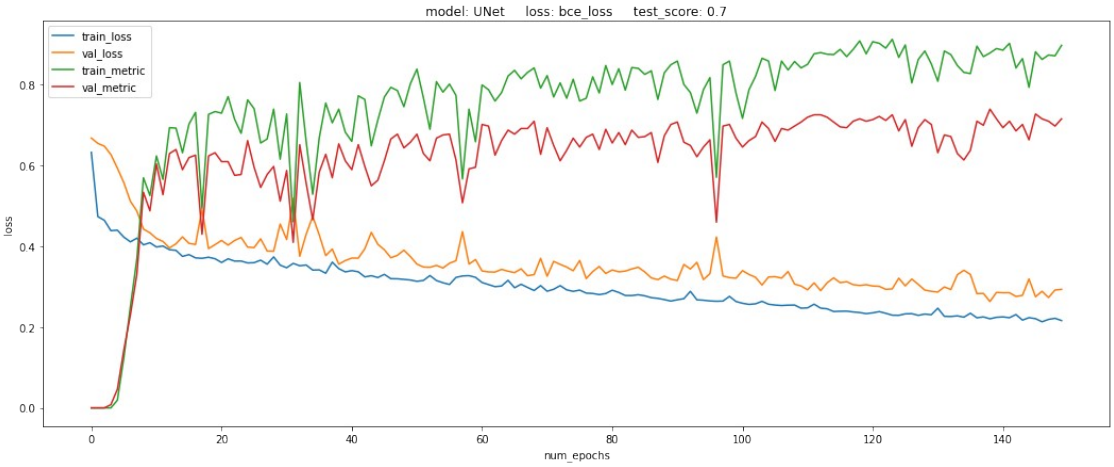
```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

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

        self.enc_conv1 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )
        self.pool1 = nn.MaxPool2d(2, stride=2, return_indices=True) #
128 -> 64

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

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

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

        # decoder (upsampling)
        self.upsample0 = nn.MaxUnpool2d(2, stride=2) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
            nn.Conv2d(1024, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )

        self.upsample1 = nn.MaxUnpool2d(2, stride=2) # 32 -> 64
        self.dec_conv1 = nn.Sequential(
            nn.Conv2d(512, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )

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

```
        )

        self.upsample3 = nn.MaxUnpool2d(2, stride=2) # 128 -> 256
        self.dec_conv3 = nn.Sequential(
            nn.Conv2d(128, 1, kernel_size=3, stride=1, padding=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
        conv_e0 = self.enc_conv0(x)
        e0, i0 = self.pool0(conv_e0)
        conv_e1 = self.enc_conv1(e0)
        e1, i1 = self.pool1(conv_e1)
        conv_e2 = self.enc_conv2(e1)
        e2, i2 = self.pool2(conv_e2)
        conv_e3 = self.enc_conv3(e2)
        e3, i3 = self.pool3(conv_e3)

        # bottleneck
        b = self.bottle_neck(e3)

        # decoder
        d0 = self.dec_conv0(torch.cat([self.upsample3(b, i3),
conv_e3], 1))
        d1 = self.dec_conv1(torch.cat([self.upsample2(d0, i2),
conv_e2], 1))
        d2 = self.dec_conv2(torch.cat([self.upsample1(d1, i1),
conv_e1], 1))
        d3 = self.dec_conv3(torch.cat([self.upsample0(d2, i0),
conv_e0], 1))

        # no activation
        return d3

model, loss_fn = UNet().to(device), bce_loss
model_name, loss_name = model._get_name(), loss_fn.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



Новая модель путем изменения типа пулинга:

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

Down-sampling:

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

Up-Sampling

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

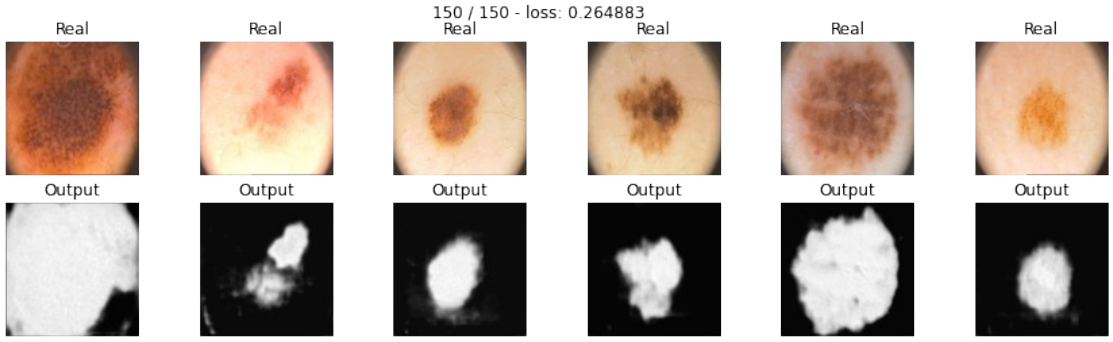
Замените max-pooling на convolutions с stride=2 и upsampling на transpose-convolutions с stride=2.

```
class UNet2(nn.Module):
    def __init__(self):
        super().__init__()
```

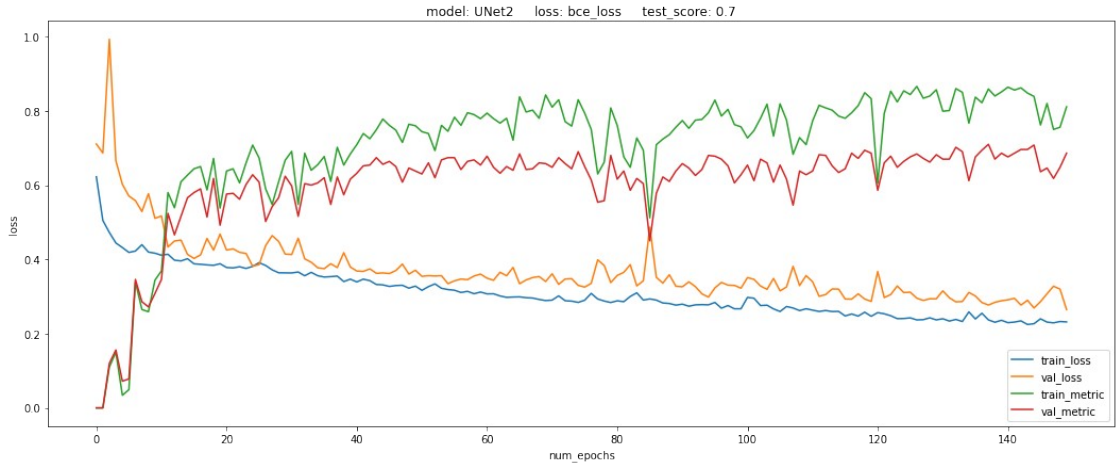
```
        # encoder
        self.enc_conv0 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )
        self.pool0 = nn.Conv2d(64, 64, kernel_size=3, stride=2,
padding=1) # 256 -> 128

        self.enc_conv1 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )
        self.pool1 = nn.Conv2d(128, 128, kernel_size=3, stride=2,
padding=1) # 128 -> 64

        self.enc_conv2 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )
        self.pool2 = nn.Conv2d(256, 256, kernel_size=3, stride=2,
padding=1) # 64 -> 32

        self.enc_conv3 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
        )
        self.pool3 = nn.Conv2d(512, 512, kernel_size=3, stride=2,
padding=1) # 32 -> 16

        # bottleneck
        self.bottle_neck = nn.Sequential(
            nn.Conv2d(512, 1024, kernel_size=1, stride=1, padding=0),
            nn.BatchNorm2d(1024),
```

```
            nn.ReLU(inplace=True),
            nn.Conv2d(1024, 512, kernel_size=1, stride=1, padding=0),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True)
        )

        # decoder (upsampling)

        self.upsample0 = nn.ConvTranspose2d(64, 64, kernel_size=3,
stride=2, padding=1) # 16 -> 32
        self.dec_conv0 = nn.Sequential(
            nn.Conv2d(1024, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )

        self.upsample1 = nn.ConvTranspose2d(128, 128, kernel_size=3,
stride=2, padding=1) # 32 -> 64
        self.dec_conv1 = nn.Sequential(
            nn.Conv2d(512, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True)
        )

        self.upsample2 = nn.ConvTranspose2d(256, 256, kernel_size=3,
stride=2, padding=1) # 64 -> 128
        self.dec_conv2 = nn.Sequential(
            nn.Conv2d(256, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )

        self.upsample3 = nn.ConvTranspose2d(512, 512, kernel_size=3,
stride=2, padding=1) # 128 -> 256
        self.dec_conv3 = nn.Sequential(
            nn.Conv2d(128, 1, kernel_size=3, stride=1, padding=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
        conv_e0 = self.enc_conv0(x)
        e0 = self.pool0(conv_e0)
        conv_e1 = self.enc_conv1(e0)
        e1 = self.pool1(conv_e1)
        conv_e2 = self.enc_conv2(e1)
        e2 = self.pool2(conv_e2)
        conv_e3 = self.enc_conv3(e2)
        e3 = self.pool3(conv_e3)

        # bottleneck
        b = self.bottle_neck(e3)

        # decoder
        d0 = self.dec_conv0(torch.cat([self.upsample3(b,
output_size=conv_e3.size()), conv_e3], 1))
        d1 = self.dec_conv1(torch.cat([self.upsample2(d0,
output_size=conv_e2.size()), conv_e2], 1))
        d2 = self.dec_conv2(torch.cat([self.upsample1(d1,
output_size=conv_e1.size()), conv_e1], 1))
        d3 = self.dec_conv3(torch.cat([self.upsample0(d2,
output_size=conv_e0.size()), conv_e0], 1))

        # no activation
        return d3

model, loss_fn = UNet2().to(device), bce_loss
model_name, loss_name = model._get_name(), loss_fn.__name__
models_data[(model_name, loss_name)] = create_model(model, loss_fn)
```




```
best_model = models_data[(model_name, loss_name)]['model']
models_data[(model_name, loss_name)]['test_score'] =
score_model(best_model, iou_pytorch, data_ts)
loss_plot(models_data[(model_name, loss_name)])
```



Сделайте вывод, какая из моделей лучше

Значительной разницы в качестве UNet и UNet2 не наблюдается - iou 70% vs 70% на тесте.

Отчет (6 баллов):

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

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

Дополнительные комментарии:

Пусть у вас есть N обученных моделей.

- Является ли отчетом N графиков с 1 линей? Да, но очень низкокачественным, потому что проверяющий не сможет сам сравнить их.
- Является ли отчетом 1 график с N линиями? Да, но скорее всего таким образом вы отразили лишь один эффект. Этого мало, чтобы сделать досточно суждений по поводу вашей работы.

- Я проверял метрики на трейне, и привел в результате таблицу с N числами, что не так? ключейвой момент тут, что вы измеряли на трейне ваши метрики, уверены ли вы, что заивисмости останутся такими же на отложенной выборке?
- Я сделал отчет содержащий график лоссов и метрик, и у меня нет ошибок в основной части, но за отчет не стоит максимум, почему? Естестественно максимум баллов за отчет можно получить не за 2 графика (даже при условии их полной правильности). Проверяющий хочет видеть больше сравнений моделей, чем метрики и лоссы (особенно, если они на трейне).

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

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

Каждая модель училась в цикле из 150 эпох. Сохранялась лучшая версия модели - то есть та, у которой метрика качества на val датасете была наибольшей.

```
model_name_col = []
loss_name_col = []
val_score_col = []
test_score_col = []

for key in models_data.keys():
    model_name_col.append(models_data[key]['model_name'])
    loss_name_col.append(models_data[key]['loss_name'])
    val_score_col.append(score_model(models_data[key]['model'],
iou_pytorch, data_val))
    test_score_col.append(models_data[key]['test_score'])

df_models_qual = pd.DataFrame({'model_name' : model_name_col,
                               'loss_name' : loss_name_col,
                               'val_score' : val_score_col,
                               'test_score' : test_score_col})
```

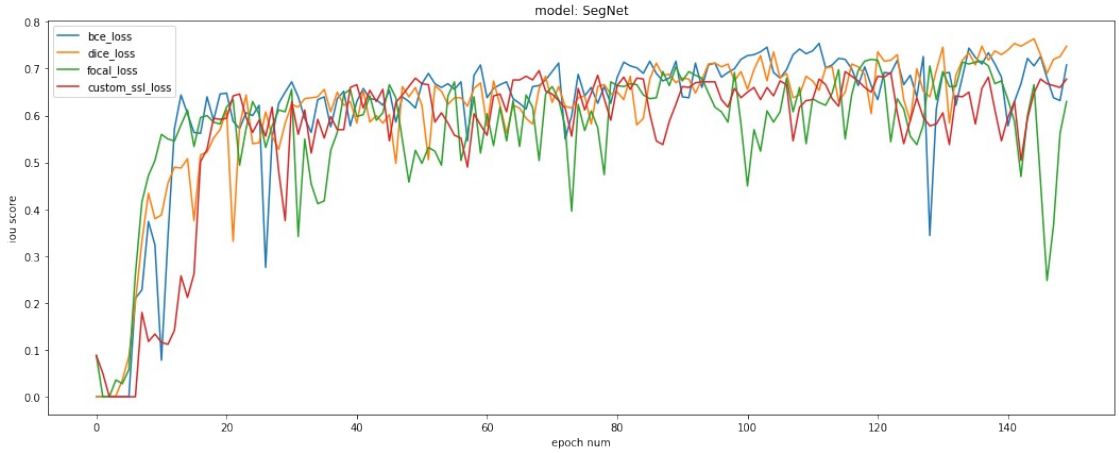
```
df_models_qual.sort_values(by=['model_name', 'loss_name'],
inplace=True)
display(df_models_qual);
```

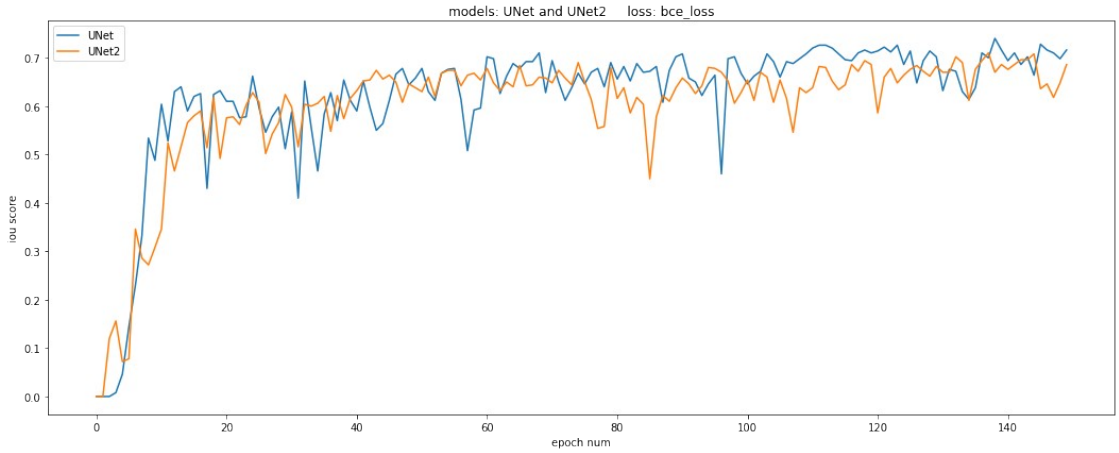
```
/opt/conda/lib/python3.7/site-packages/torch/nn/functional.py:749:
UserWarning: Note that order of the arguments: ceil_mode and
return_indices will changeto match the args list in nn.MaxPool2d in a
future release.
  warnings.warn("Note that order of the arguments: ceil_mode and
return_indices will change"
```

	model_name	loss_name	val_score	test_score
0	SegNet	bce_loss	0.754	0.728
3	SegNet	custom_ssl_loss	0.696	0.676
1	SegNet	dice_loss	0.764	0.706
2	SegNet	focal_loss	0.720	0.724
4	UNet	bce_loss	0.740	0.698
5	UNet2	bce_loss	0.710	0.698

Наилушей моделью оказался SegNet с bce_loss в качестве функции ошибки. Наихудшей - SegNet с custom_ssl_loss.

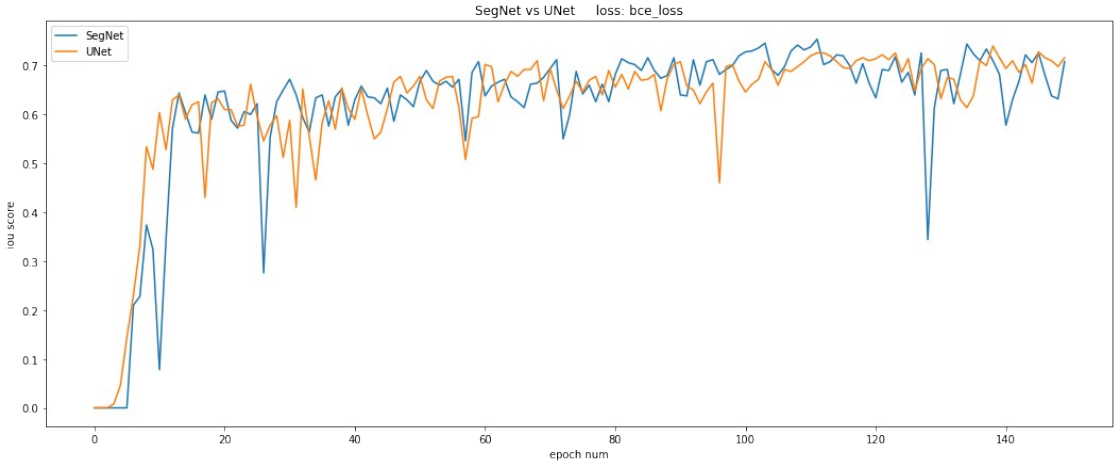
```
x_values = list(range(150)) # epochs
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(18, 15))

for key in models_data.keys():
    if models_data[key]['model_name'] == 'SegNet':
        axs[0].plot(x_values, models_data[key]['val_metric_values'],
label=models_data[key]['loss_name'])
        axs[0].set_title('model: SegNet')
        axs[0].set_xlabel('epoch num')
        axs[0].set_ylabel('iou score')
        axs[0].legend()
    else:
        axs[1].plot(x_values, models_data[key]['val_metric_values'],
label=models_data[key]['model_name'])
        axs[1].set_title('models: UNet vs UNet2     loss: bce_loss')
        axs[1].set_xlabel('epoch num')
        axs[1].set_ylabel('iou score')
        axs[1].legend()

plt.show()
```





Наилучшая SegNet модель против наилучшей UNet модели.

```
x_values = list(range(150)) # epochs
plt.figure(figsize=(18, 7))

for key in models_data.keys():
    if models_data[key]['loss_name'] == 'bce_loss' and
models_data[key]['model_name'] == 'SegNet':
        plt.plot(x_values, models_data[key]['val_metric_values'],
label=models_data[key]['model_name'])
        plt.title('SegNet vs UNet     loss: bce_loss')
        plt.xlabel('epoch num')
        plt.ylabel('iou score')
    elif models_data[key]['loss_name'] == 'bce_loss' and
models_data[key]['model_name'] == 'UNet':
        plt.plot(x_values, models_data[key]['val_metric_values'],
label=models_data[key]['model_name'])

plt.legend()
plt.show()
```


SegNet vs UNet loss: bce_loss
SegNet
UNet
iou score
epoch num
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100
120
140