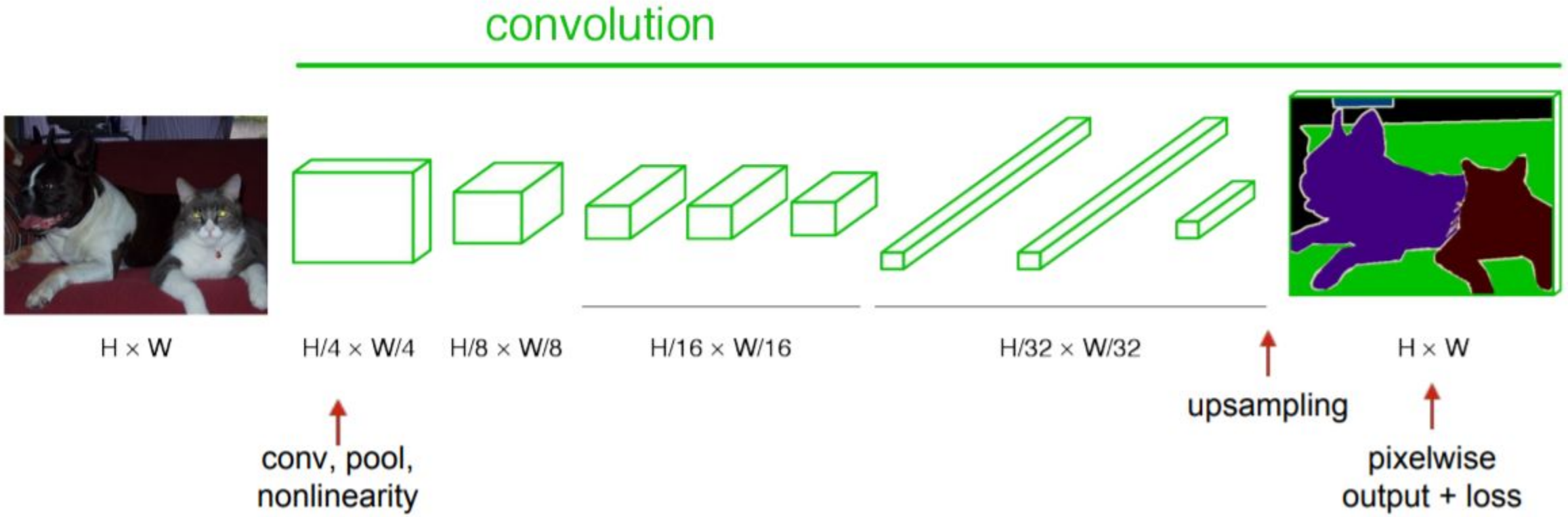
Научно учебная группа
«Цифровые методы в неврологии»

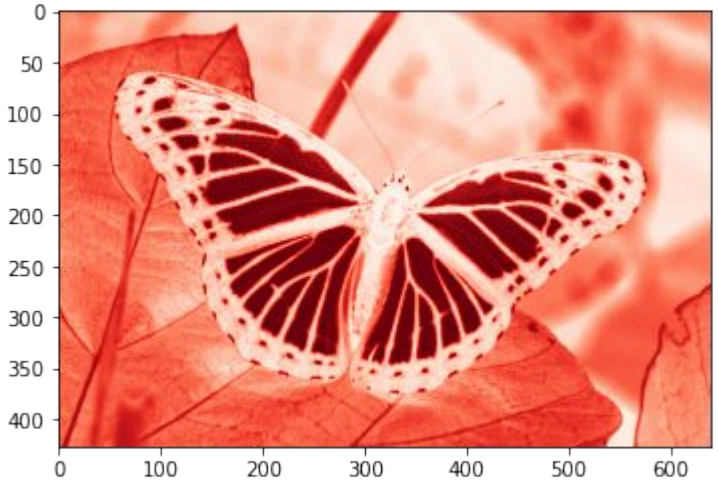
Пермь,
2023

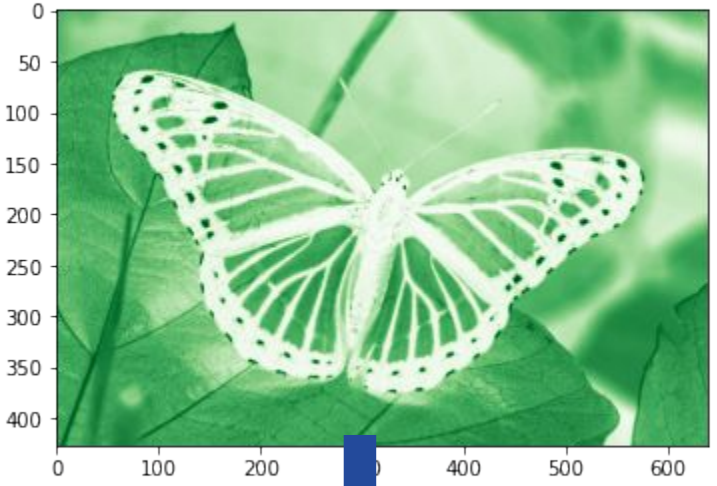
Реализация модели глубинного обучения для модели семантической сегментации

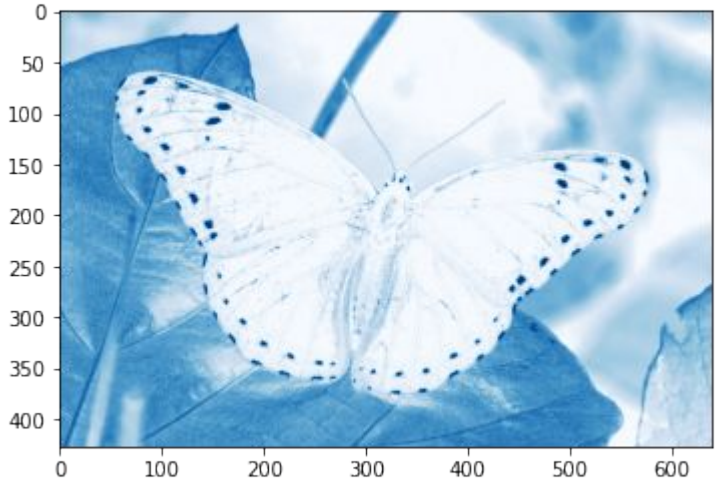
Лукин Семён Олегович
студент 3 курса ОП Программная инженерия

Работа модели



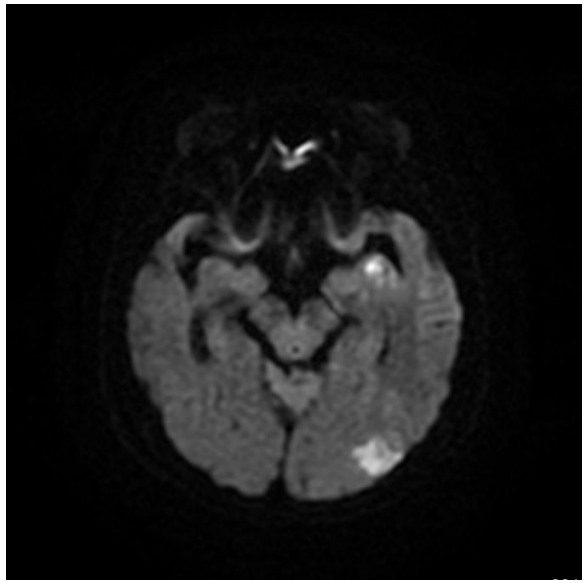

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600

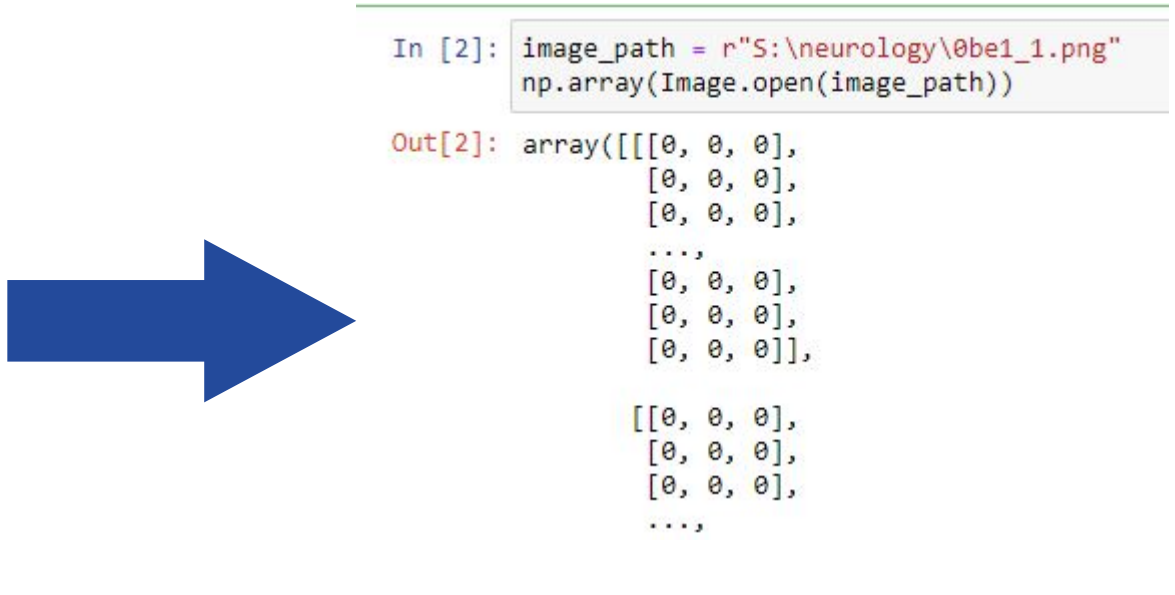

0
50
100
150
200
250
300
350
400
0
100
200
400
500
600


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600

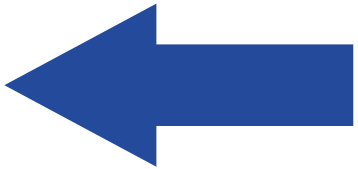

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600

Как хранятся картинки?

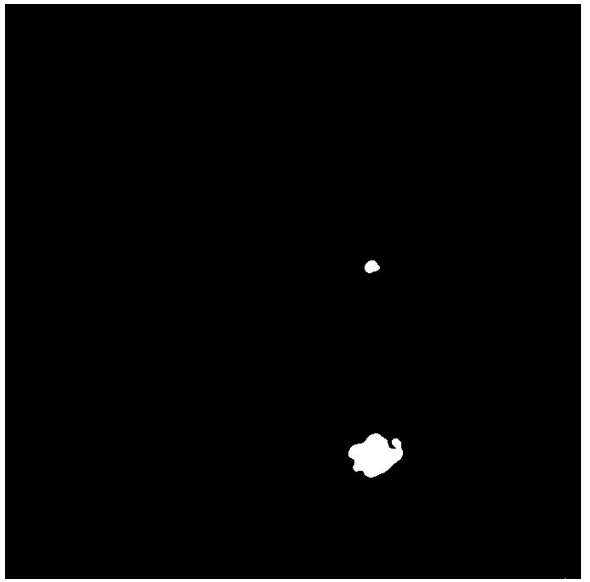




```
In [2]: image_path = r"S:\neurology\0be1_1.png"
        np.array(Image.open(image_path))

Out[2]: array([[[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0],
                ...,
                [0, 0, 0],
                [0, 0, 0],
                [0, 0, 0]],

               [[0, 0, 0],
                [0, 0, 0],
                [0, 0, 0],
                ...,
```



```
In [3]: mask_path = r"S:\neurology\0be1_1_mask.png"
        np.array(Image.open(mask_path))

Out[3]: array([[0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               ...,
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

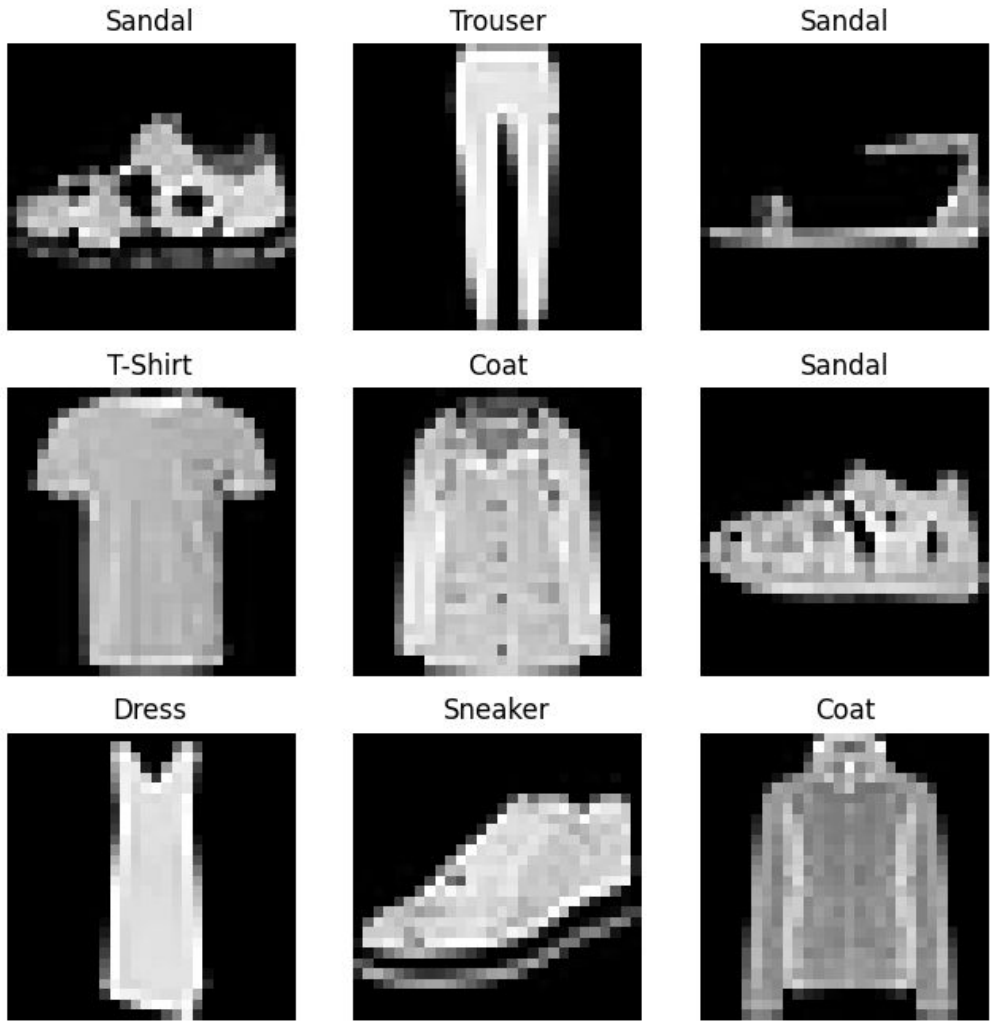


Как собрать все данные в одном месте?

Создать датасет!

```python
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

70,000
28x28 grayscal
e images of
fashion
products from
10 categories,
2017



Как собрать все данные в одном месте?

Создать датасет!

```python
class SegmentationDataset(Dataset):
    def __init__(self,
                 slices_dir,
                 masks_dir,
                 transforms):

        self.slices_dir = sorted(glob.glob(os.path.join(slices_dir, "*.png")))
        self.masks_dir = sorted(glob.glob(os.path.join(masks_dir, "*.png")))
        self.transforms = transforms

        assert len(self.slices_dir) == len(self.masks_dir)
```

dataset.ipynb

dataset.py

```python
def __getitem__(self, idx):
    image = np.array(Image.open(self.slices_dir[idx]))
    mask = np.array(Image.open(self.masks_dir[idx]))

    mask[mask == 255.0] = 1.0

    if self.transforms is not None:
        augmentations = self.transforms(image=image, mask=mask)
        image = augmentations["image"]
        mask = augmentations["mask"]

    mask = torch.unsqueeze(mask, 0)
    mask = mask.type(torch.float32)
    return image, mask


def __len__(self):
    return len(self.slices_dir)
```

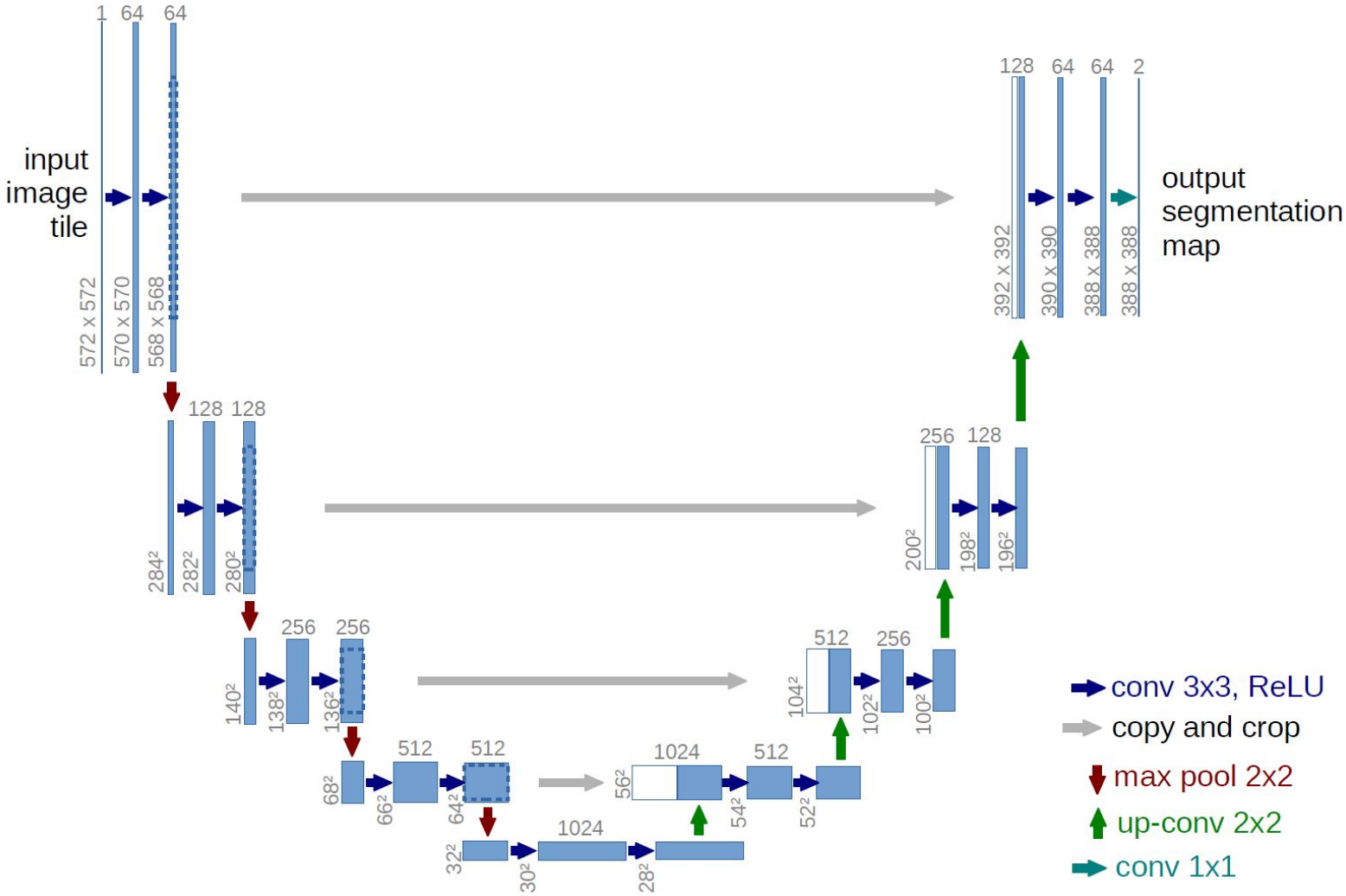
Как сообщить модели о способе работы с датасетом?

Создать даталоадер!

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

```
def create_dataloaders(train_dataset, val_dataset, test_dataset,
                       batch_size=8, pin_memory=True, num_workers=4):

    train_loader = torch.utils.data.DataLoader(
        dataset=train_dataset, batch_size=batch_size,
        shuffle=True, pin_memory=pin_memory, num_workers=num_workers)

    val_loader = torch.utils.data.DataLoader(
        dataset=val_dataset, batch_size=batch_size,
        shuffle=True, pin_memory=pin_memory, num_workers=num_workers
    )

    test_loader = torch.utils.data.DataLoader(
        dataset=test_dataset, batch_size=batch_size,
        shuffle=False, pin_memory=pin_memory, num_workers=num_workers
    )
    return train_loader, val_loader, test_loader
```

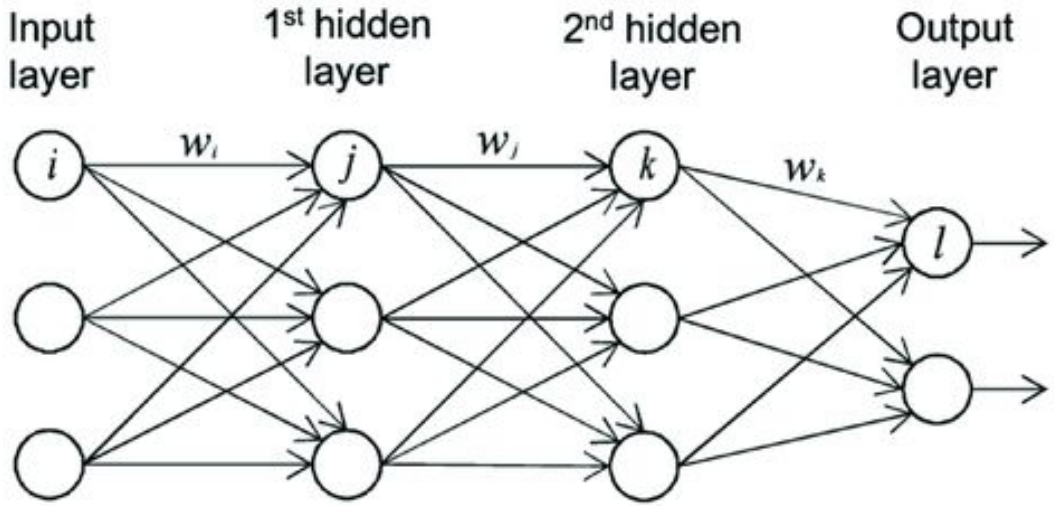
Построение архитектуры модели с помощью PyTorch



Построение архитектуры модели с помощью

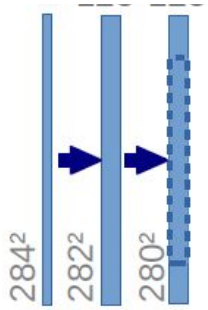
```
model = nn.Sequential()                 # создаем пустую модель, в которую будем добавлять слои
model.add_module('l1', nn.Linear(3, 3)) # добавили слой с 3-мя нейронами на вход и 3-мя на выход
model.add_module('r1', nn.ReLU())       # добавили функцию активации
model.add_module('l2', nn.Linear(3, 3)) # добавили слой с 3-мя нейронами на вход и 3-ю на выход
model.add_module('r2', nn.ReLU())       # добавили функцию активации
model.add_module('l3', nn.Linear(3, 3)) # добавили слой с 3-мя нейронами на вход и 3-ю на выход
model.add_module('r3', nn.ReLU())       # добавили функцию активации
model.add_module('l2', nn.Linear(3, 2)) # добавили слой с 3-мя нейронами на вход и 2-ю на выход
```

```
y_pred = model(batch_x) # получили предсказания модели
```



Построение архитектуры модели с помощью PyTorch

```
class DoubleConv(pl.LightningModule):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(

            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False), # bias=False for B
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),

            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)
```



conv 3x3, ReLU

Построение архитектуры модели с помощью

model.py

```
class UNET(nn.Module):
    def __init__(
            self, in_channels=3, out_channels=1, features=[64, 128, 256, 512],
    ):
        super(UNET, self).__init__()

        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Down part of UNET - only left side
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET - only right side
        for feature in reversed(features):
            # up
            self.ups.append(
                nn.ConvTranspose2d(
                    # with skip-conn
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            # two times right
            self.ups.append(DoubleConv(feature*2, feature))

        self.bottleneck = DoubleConv(features[-1], features[-1]*2)
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)
```

model.ipynb

```
def forward(self, x):
    skip_connections = []

    for down in self.downs:
        x = down(x)
        skip_connections.append(x)
        x = self.pool(x)

    x = self.bottleneck(x)

    # reverse skip_connections
    skip_connections = skip_connections[::-1]

                    # пропускаем DoubleConv
    for idx in range(0, len(self.ups), 2):
        x = self.ups[idx](x)
        skip_connection = skip_connections[idx//2]

        # resize if x does not match skip_conn
        if x.shape != skip_connection.shape:
            x = TF.resize(x, size=skip_connection.shape[2:], antialias=True)

        concat_skip = torch.cat((skip_connection, x), dim=1)
        x = self.ups[idx+1](concat_skip)

    return self.final_conv(x)
```

Выбираем функцию потерь

PyTorch

Loss Functions

`nn.L1Loss`	Creates a criterion that measures the mean absolute error (MAE) between each element in the input x and target y.
`nn.MSELoss`	Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.
`nn.CrossEntropyLoss`	This criterion computes the cross entropy loss between input logits and target.
`nn.CTCLoss`	The Connectionist Temporal Classification loss

Segmentation Models

- Losses
  - Constants
  - JaccardLoss
  - DiceLoss
  - TverskyLoss
  - FocalLoss
  - LovaszLoss
  - SoftBCEWithLogitsLoss
  - SoftCrossEntropyLoss
  - MCCLoss

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

$$Dice\ Loss = 1 - Dice\ coef$$

Выбираем метод оптимизации – TORCH.OPTIM

```
[ ]  1 [elem for elem in dir(torch.optim) if not elem.startswith("_")]
```

```
['ASGD',
 'Adadelta',
 'Adagrad',
 'Adam',
 'AdamW',
 'Adamax',
 'LBFGS',
 'NAdam',
 'Optimizer',
 'RAdam',
 'RMSprop',
 'Rprop',
 'SGD',
 'SparseAdam',
 'lr_scheduler',
 'swa_utils']
```

Algorithms

Adadelta	Implements Adadelta algorithm.
Adagrad	Implements Adagrad algorithm.
Adam	Implements Adam algorithm.
AdamW	Implements AdamW algorithm.
SparseAdam	Implements lazy version of Adam algorithm suitable for sparse tensors.

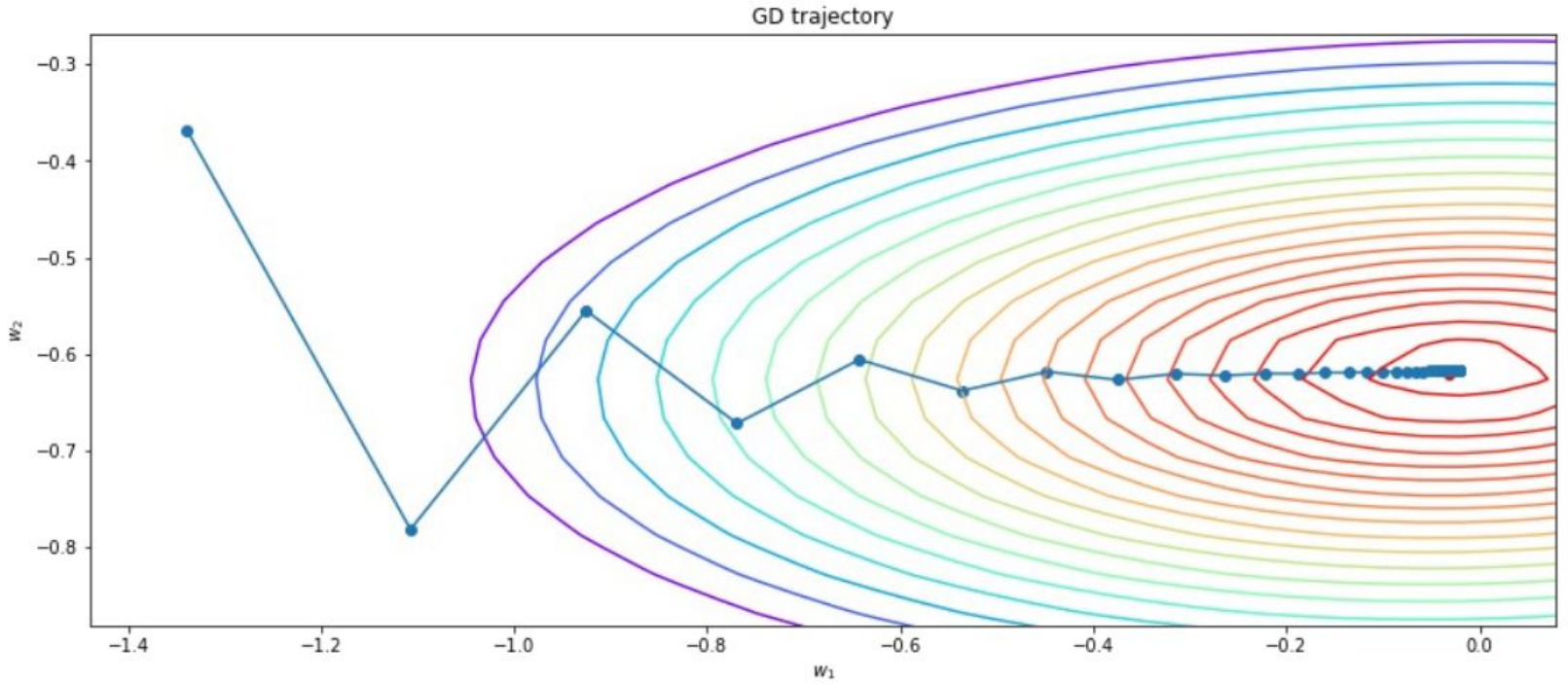
Немного формул

Gradient descent

$$w^t = w^{t-1} - \eta \nabla Q(w^{t-1})$$

+ Точное значение градиента

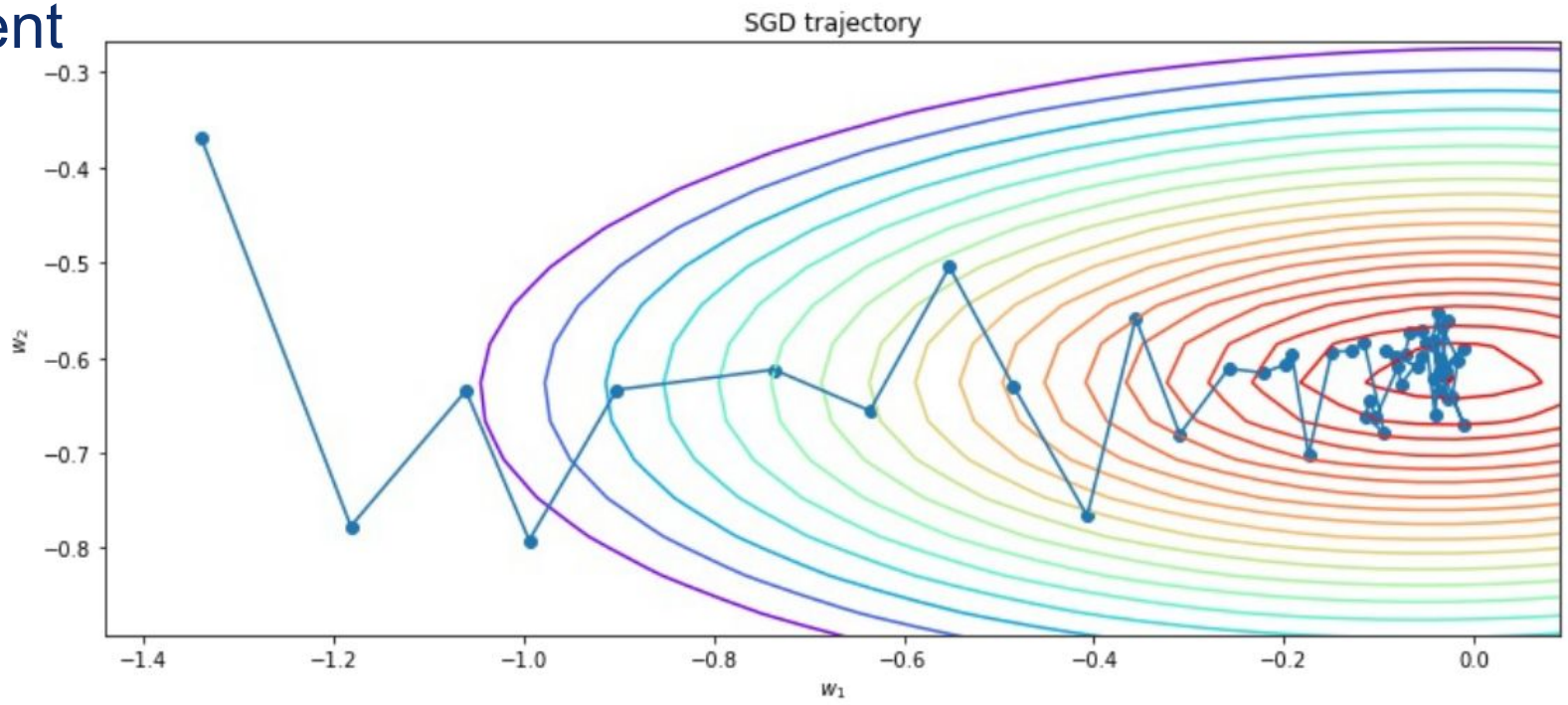
- Долго и трудозатратно



Немного формул

Stochastic Gradient Descent

$$w^t = w^{t-1} - \eta_t \nabla L\left(y_{i_t}, a(x_{i_t})\right)$$



+ Быстро

- Большой разброс

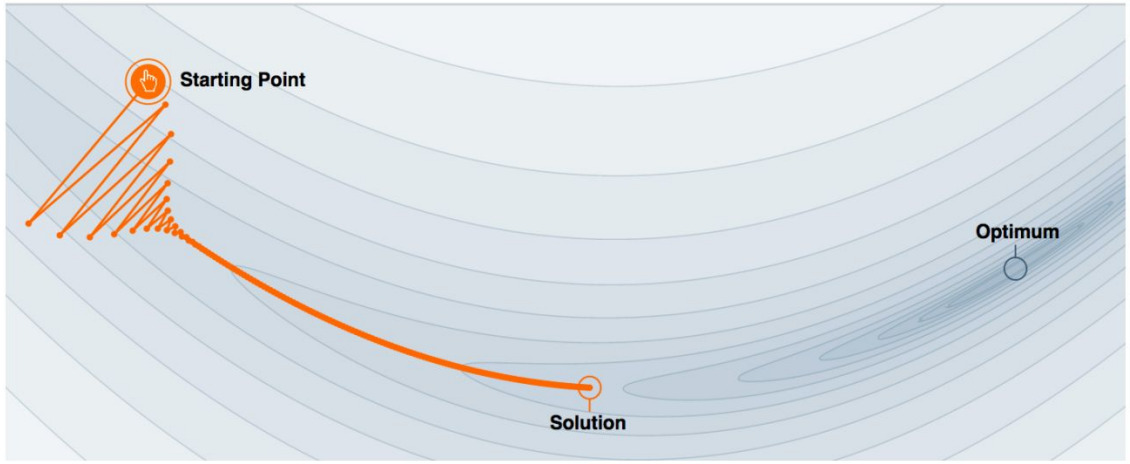
Немного формул

Stochastic Gradient Descent with Momentum

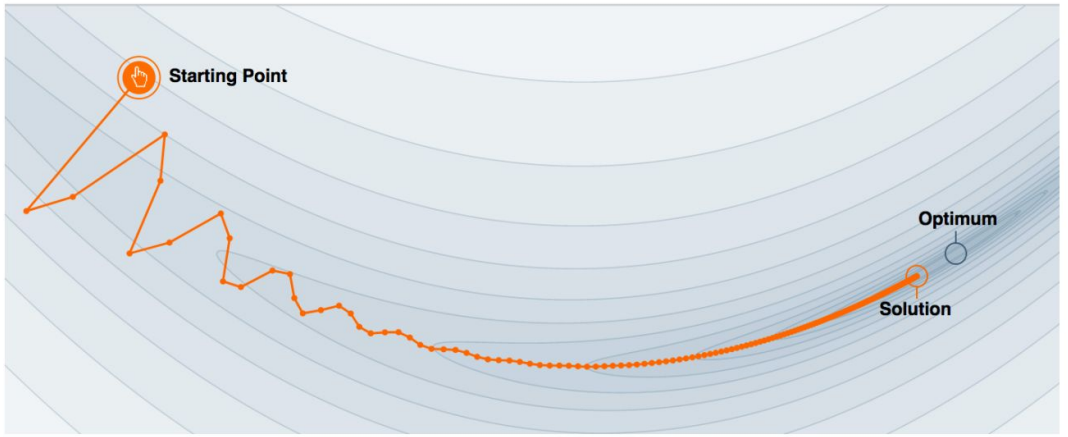
$$h_t = \alpha h_{t-1} + \eta_t \nabla Q(w^{t-1})$$

$$w^t = w^{t-1} - h_t$$

Без инерции



С инерцией



Немного формул

Адаптивные методы градиентного спуска

AdaGrad

$$G_j^t = G_j^{t-1} + \left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}}\left(\nabla Q(w^{t-1})\right)_j$$

RMSProp

$$G_j^t = \alpha G_j^{t-1} + (1-\alpha)\left(\nabla Q(w^{t-1})\right)_j^2$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}}\, g_{tj}$$

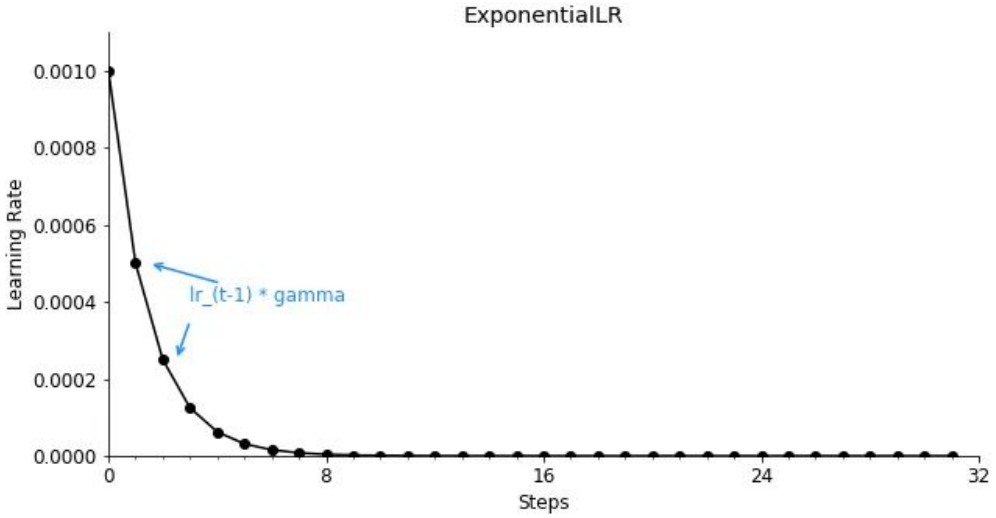
Adam

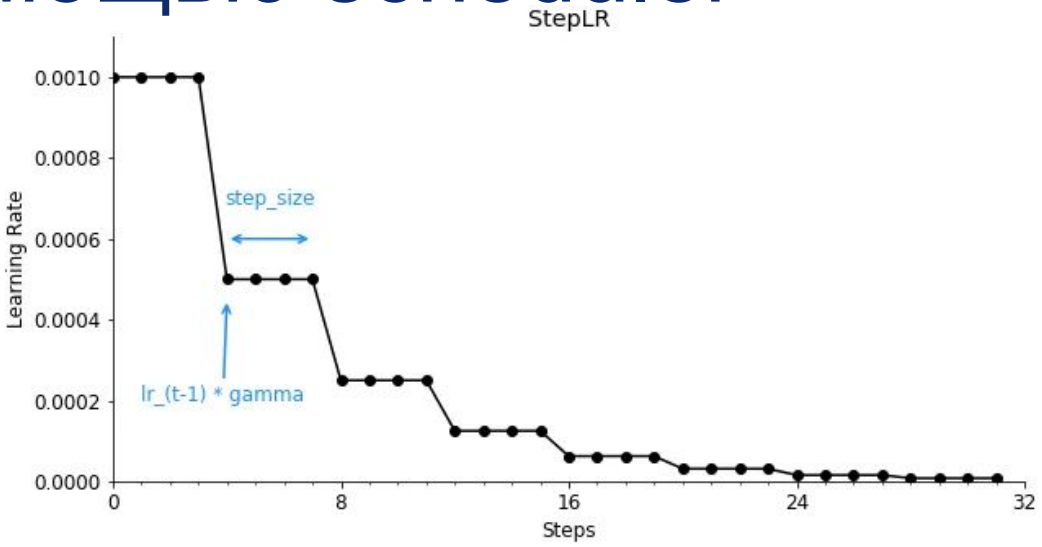
$$m_j^t = \frac{\beta_1 m_j^{t-1} + (1-\beta_1)\left(\nabla Q(w^{t-1})\right)_j}{1-\beta_1^t}$$

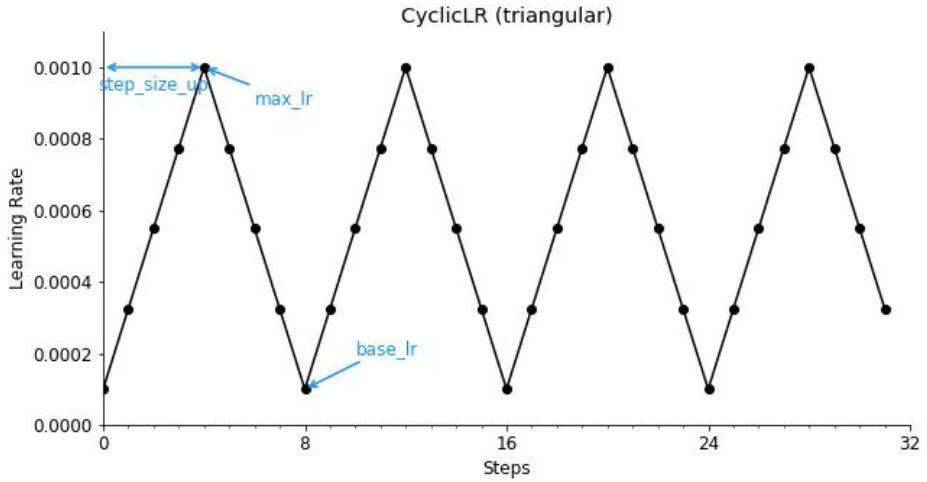
$$v_j^t = \frac{\beta_2 v_j^{t-1} + (1-\beta_2)\left(\nabla Q(w^{t-1})\right)_j^2}{1-\beta_2^t}$$

$$w_j^t = w_j^{t-1} - \frac{\eta_t}{\sqrt{v_j^t + \epsilon}}\, m_j^t$$

Изменяем learning rate с помощью scheduler







Классический цикл обучения модели

```
[ ]  1 total_step = len(train_loader)
     2 for epoch in range(NUM_EPOCHS): # цикл по эпохам обучения
     3     for i, (images, labels) in enumerate(train_loader): # цикл по мини-батчам
     4         # Перенос обоих тензоров на GPU
     5         images = images.to(DEVICE)
     6         labels = labels.to(DEVICE)
     7
     8         # Проход вперед
     9         outputs = net(images)
    10         loss = criterion(outputs, labels)
    11
    12         # Проход назад
    13         optimizer.zero_grad()
    14         loss.backward()
    15
    16         # Обновление параметров нейросети
    17         optimizer.step()
    18
    19         if (i+1) % 200 == 0:
    20             print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
    21                    .format(epoch+1, NUM_EPOCHS, i+1, total_step, loss.item()))
```

Обучение в PyTorch Lightning

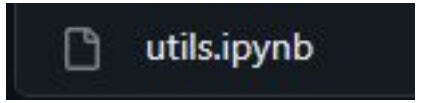
```
[ ]  1 model = UNET()
     2
     3 trainer = pl.Trainer(
     4     max_epochs=2,
     5     accelerator="gpu",
     6     devices=1,
     7     auto_lr_find=True,
     8 )
     9
    10 trainer.fit(model, train_loader, val_loader)
```

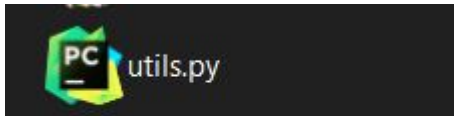
Модель в PyTorch Lightning

`__init__` and `setup()`	Define initialization here
`forward()`	To run data through your model only (separate from `training_step`)
`training_step()`	the complete training step
`validation_step()`	the complete validation step
`test_step()`	the complete test step
`predict_step()`	the complete prediction step
`configure_optimizers()`	define optimizers and LR schedulers

```python
import lightning.pytorch as pl
import torch.nn as nn
import torch.nn.functional as F


class LitModel(pl.LightningModule):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(28 * 28, 10)

    def forward(self, x):
        return torch.relu(self.l1(x.view(x.size(0), -1)))

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = F.cross_entropy(y_hat, y)
        return loss

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=0.02)
```

Вспомогательные функции





- Сохранить и загрузить модель
- Создание датасета и даталоадера
- Подсчитать метрики с помощью готовой модели (mean Dice, accuracy)
- Вывести результат работы

Поле экспериментов

0. Импорты

```
import torch
import torch.nn as nn
from tqdm import tqdm
import torch.nn as nn
import torch.optim as optim
import pytorch_lightning as pl

from model import UNET
from utils import (
    load_checkpoint,
    save_checkpoint,
    create_dataloaders,
    create_datasets,
    check_accuracy,
    get_mean_dice,
    show_three,
)

import albumentations as A
from albumentations.pytorch import ToTensorV2
```

Поле экспериментов

1. Параметры и гиперпараметры

```
# Hyperparameters etc.
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

LEARNING_RATE = 1e-4
BATCH_SIZE = 16
NUM_EPOCHS = 20
NUM_WORKERS = 2
IMAGE_HEIGHT = 256  # 1280 originally
IMAGE_WIDTH = 256  # 1918 originally

PIN_MEMORY = True
LOAD_MODEL = False
#
MASKS = "/kaggle/input/stroke-aid-dataset-splitted/masks"
SLICES = "/kaggle/input/stroke-aid-dataset-splitted/slices"
```

Поле экспериментов

2. Аугментации данных

```
train_transform = A.Compose(
    [
        A.Resize(height=IMAGE_HEIGHT, width=IMAGE_WIDTH),
        # A.Rotate(limit=35, p=1.0),
        # A.HorizontalFlip(p=0.5),
        # A.VerticalFlip(p=0.1),
        A.Normalize(
            mean=[0.0, 0.0, 0.0],
            std=[1.0, 1.0, 1.0],
            max_pixel_value=255.0,
        ),
        ToTensorV2(),
    ],
)

val_transforms = A.Compose(
    [
        A.Resize(height=IMAGE_HEIGHT, width=IMAGE_WIDTH),
        A.Normalize(
            mean=[0.0, 0.0, 0.0],
            std=[1.0, 1.0, 1.0],
            max_pixel_value=255.0,
        ),
        ToTensorV2(),
    ],
)
```

Поле экспериментов

3. Определение функции потерь, оптимизатора и модели

```
loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.Adam
optimizer_params = [{"lr": LEARNING_RATE}]

model = UNET(in_channels=3,
             out_channels=1,
             loss_fn=loss_fn,
             optimizer=optimizer,
             optimizer_params=optimizer_params
            ).to(DEVICE)
```

Поле экспериментов

4. Создаём лоадеры

```
train_dataset, val_dataset, test_dataset = create_datasets(MASKS, SLICES, train_transform)

train_loader, val_loader, test_loader = create_dataloaders(train_dataset, val_dataset, test_dataset, batch_size
```

```
Train size: 450
Valid size: 148
Test size:  149
```

Поле экспериментов

5. Обучаем

```
seed_everything(123456)
torch.cuda.empty_cache()

trainer = pl.Trainer(
    max_epochs=NUM_EPOCHS,
    accelerator=DEVICE,
    devices=1,
)

trainer.fit(model, train_loader, val_loader)
```

Результат

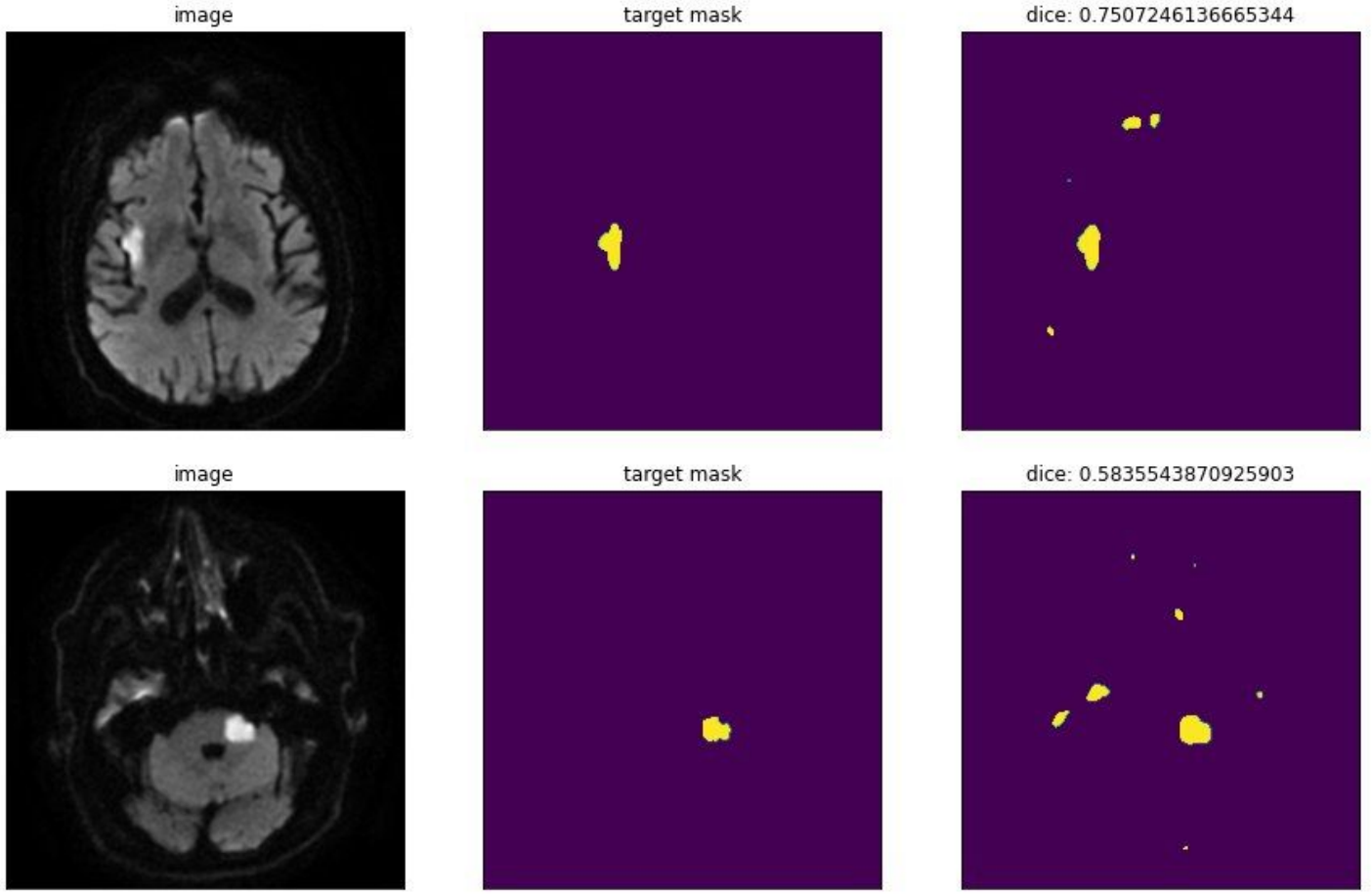


image
target mask
dice: 0.7507246136665344
image
target mask
dice: 0.5835543870925903

Источники

- Лекции и семинары дисциплины «Основы глубинного обучения» https://github.com/hse-ds/iad-deep-learning
- Лекции и семинары дисциплины «Прикладные задачи анализа данных» https://github.com/hse-ds/iad-applied-ds


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •


ВЫСШАЯ • ШКОЛА ЭКОНОМИКИ •

Формулы

$$\epsilon_\theta(x_t, I, t) = D((E_t^I + E_t^x, t), t),$$

где:

θ – параметры обратного процесса,

$I_{n\times n}$ – входное изображение,

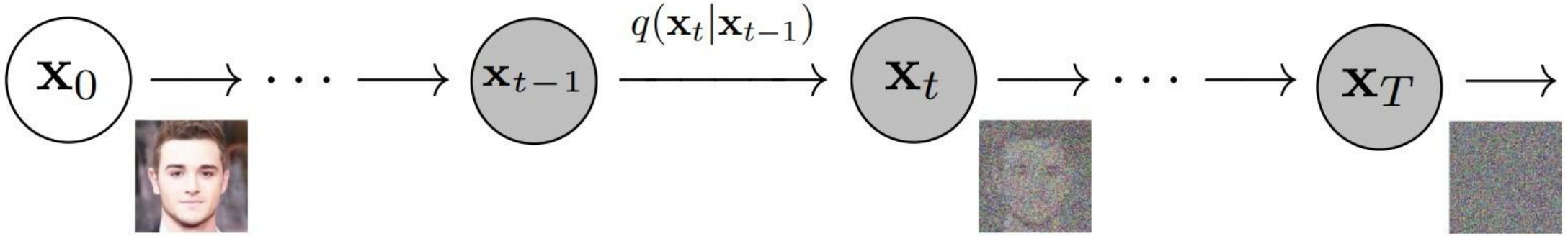
E_t^I – условный эмбендинг (необработанное изображение),

E_t^x – сегментационная разметка,

D – декодер Unet,

Прямой процесс диффузии

Преобразование из x_{t-1} в x_{t-1} задаём сами



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \boldsymbol{\Sigma}_t = \beta_t\mathbf{I})$$

где:

β_t – константа,

I – диагональная матрица ковариаций, на диагоналях стоят σ, остальные – нули

This X Does Not Exist

dd

A