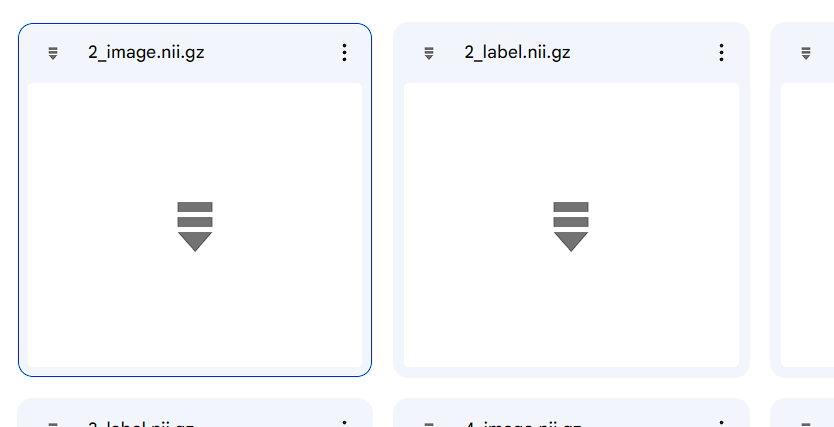
Работа по обучению нейронной сети состояла из следующих этапов:

1. Поиск данных для обучения и тестирования нейронной сети;
2. Реализация загрузки данных с систему;
3. Реализация модели обучения нейронной сети;
4. Реализация расчета метрики IoU;
5. Реализация обучения и тестирования нейронной сети;
6. Анализ получившихся метрик и погрешности.

**Поиск данных**

Для поиска данных “Type B Aortic Dissection”(TBAD) содержащего компьютерную томографию грудной клетки и маски с разграниченной на них аортой. Мной были выбраны данные размещенные автором [XiaoweiXu](https://github.com/XiaoweiXu/Dataset_Type-B-Aortic-Dissection/commits?author=XiaoweiXu), в которых содержится 100 3D изображений и масок. В файлах маски аорта была сегментирована по 3 подструктурам: TL, FL, и FLT.

Структура датасета представляет из себя папку, в которой находятся все файлы изображений и масок. Данные представляют из себя NII файлы. Данное разрешение файлов распространено для медицинских изображений.



**Загрузка данных**

Данные были размещены на Google Disk. Для доступа к ним была использована встроенная библиотека в Google Collab, а именно «google.colab».

from google.colab import drive

drive.mount('/content/drive')

Если код запускался с аккаунта, на котором нет данных для обработки и обучения, тогда выполняется функция, в котором передается ссылка на данные обучения, после чего в Google Collab они загружаются во временную среду.

def folder\_download(folder\_id):

  # authenticate

  from google.colab import auth

  auth.authenticate\_user()

  # get folder\_name

  from googleapiclient.discovery import build

  service = build('drive', 'v3')

  folder\_name = service.files().get(fileId=folder\_id).execute()['name']

  # import library and download

  !wget -qnc https://github.com/segnolin/google-drive-folder-downloader/raw/master/download.py

  from download import download\_folder

  download\_folder(service, folder\_id, './', folder\_name)

  return folder\_name

Загруженные данные необходимо разделить на 3 части: тренировочную, валидационную и тестовую. Для этого 100 файлов были разделены в соотношении 60/20/20 соответственно.

content\_path = "/content/dataset\_ad\_publish"

content\_directory = Path(content\_path)

ids = content\_directory.glob('\*\_image\*')

ids = [id.stem for id in ids]

ids = [id.split('\_')[0] for id in ids]

ids = sorted(map(int, ids))

train\_count = len(ids)\*6//10

train\_ids = random.sample(ids, train\_count)

val\_ids=list(set(ids)-set(train\_ids))

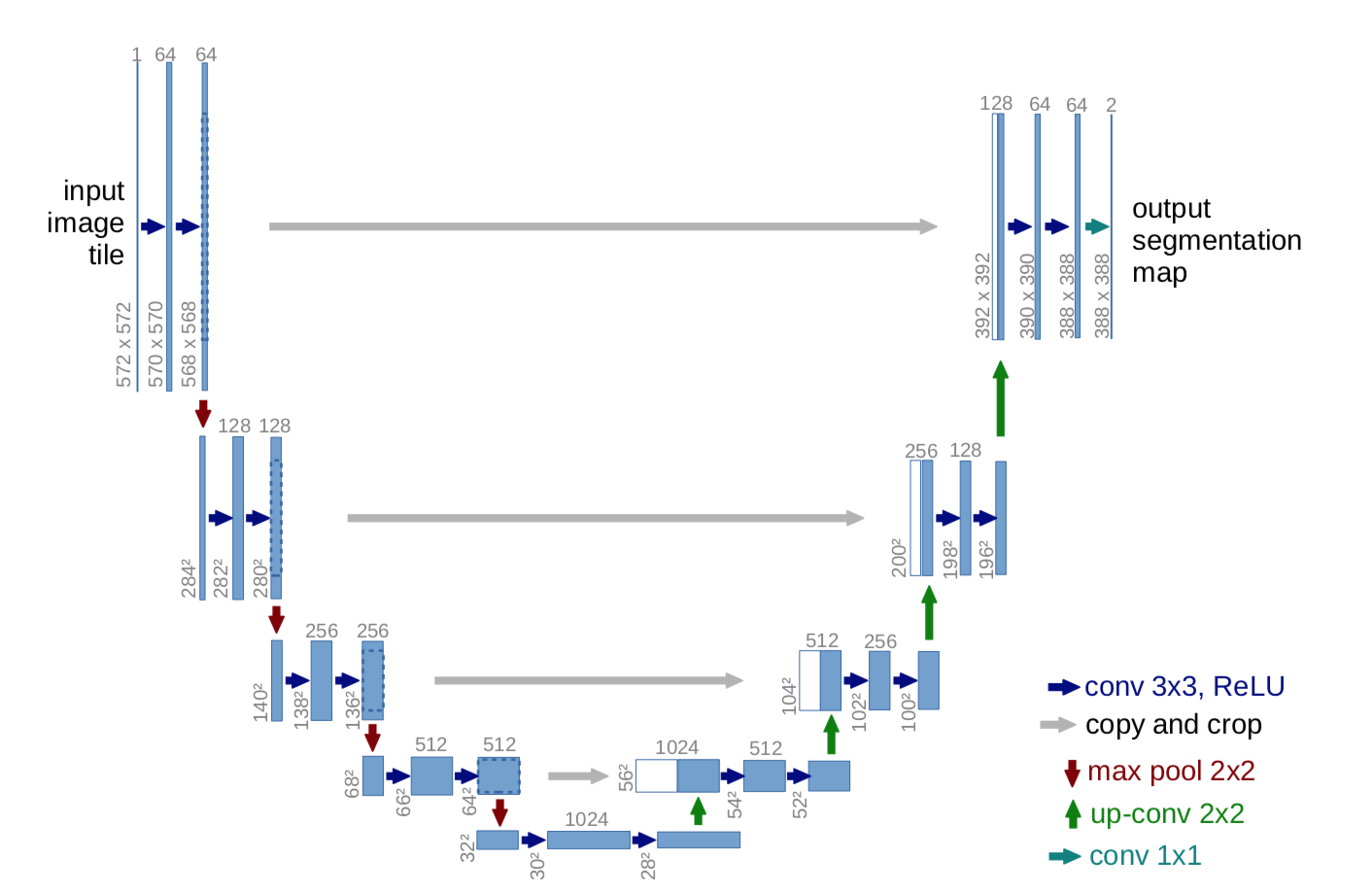
val\_count = len(val\_ids)//2

val\_ids = random.sample(val\_ids, val\_count)

test\_ids = list(set(ids)-set(train\_ids)-set(val\_ids))

**Модель UNet**

Для обучения модели была выбрана модель UNet, которая по форме представляет из себя букву «U».



Для реализации архитектуры были использована библиотека Torch. Первым этапом была реализована функция двойной свертки, на вход которой подается размеры входного и выходного каналов.

def double\_convolution(in\_channels, out\_channels):

    conv\_op = nn.Sequential(

        nn.Conv2d(in\_channels, out\_channels, kernel\_size=3, padding=1),

        nn.ReLU(inplace=True),

        nn.Conv2d(out\_channels, out\_channels, kernel\_size=3, padding=1),

        nn.ReLU(inplace=True)

        )

    return conv\_op

После этого был реализован класс с одноименным названием модели. В первой части реализуется уменьшение изображения, с помощью функции двойной свертки. Во второй происходит расширение, реализованное с помощью функции ConvTranspose2d.

class UNet(nn.Module):

    def \_\_init\_\_(self, num\_classes=1):

        super(UNet, self).\_\_init\_\_()

        self.max\_pool2d = nn.MaxPool2d(kernel\_size=2, stride=2)

        self.down\_convolution\_1 = double\_convolution(1, 64)

……………

        self.down\_convolution\_5 = double\_convolution(512, 1024)

        self.up\_transpose\_1 = nn.ConvTranspose2d( in\_channels=1024,

                                                  out\_channels=512,

                                                  kernel\_size=2,

                                                  stride=2)

        self.up\_convolution\_1 = double\_convolution(1024, 512)

        self.up\_transpose\_2 = nn.ConvTranspose2d( in\_channels=512,

                                                  out\_channels=256,

                                                  kernel\_size=2,

                                                  stride=2)

……………

        self.up\_convolution\_4 = double\_convolution(128, 64)

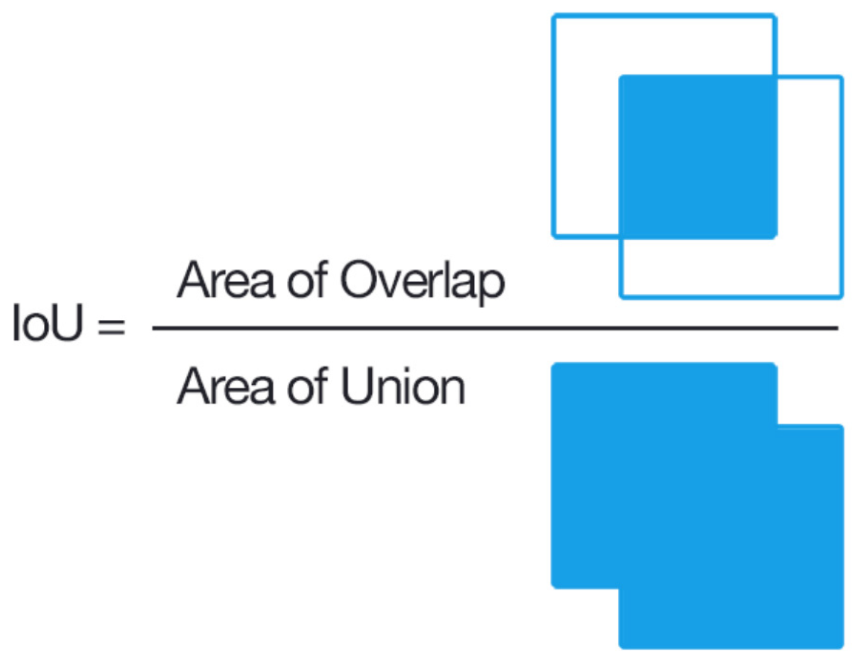
        self.out = nn.Conv2d(in\_channels=64,

                             out\_channels=num\_classes,

                             kernel\_size=1)

**Метрика IoU**

Метрика IoU представляет собой отношение между пересечением полученной из метрики маски и истинного ее значения и объединением этих же параметров.



Для реализации была разработана функция, которая на вход получает две маски, после чего происходит их побитовое умножение и сложение. После этого происходит расчет отношения между ними.

def iou\_pytorch(outputs: torch.Tensor, labels: torch.Tensor):

    outputs = outputs.squeeze(1).byte()

    labels = labels.squeeze(1).byte()

    intersection = (outputs & labels).float().sum((1, 2))

    union = (outputs | labels).float().sum((1, 2))

    iou = ((intersection) / (union)).float()

    return iou

**Обучение нейронной сети**

Для обучения нейронной сети был создан цикл, в котором рассматриваются все идентификаторы для обучения.

for num\_file in tqdm(train\_ids):

        itk\_img = itk.ReadImage(f'{path}/{num\_file}\_image.nii.gz')

        img = itk.GetArrayFromImage(itk\_img)

        itk\_img = itk.ReadImage(f'{path}/{num\_file}\_label.nii.gz')

        msk = itk.GetArrayFromImage(itk\_img)

После получения значений файлов необходимо запустить еще один цикл. Это необходимо так как структура NII состоит из множества срезов, которые размещаются друг за другом, объединяя которые можно получить 3D модель. Так как в маске присутствует несколько частей аорты, следовательно необходимо их объединить ее в одну. После этого необходимо передать в модель изображение для проверки и выявления аорты. После этого рассчитываются погрешность и метрика IoU, обновляется оптимизор и лоссы.

for num\_file in tqdm(train\_ids):

        itk\_img = itk.ReadImage(f'{path}/{num\_file}\_image.nii.gz')

        img = itk.GetArrayFromImage(itk\_img)

        itk\_img = itk.ReadImage(f'{path}/{num\_file}\_label.nii.gz')

        msk = itk.GetArrayFromImage(itk\_img)

        for num\_image in range(0, img.shape[0], 1):

            image = np.array(img[num\_image].copy())

            image\_tensor = transform(image).float()

            image\_tensor = image\_tensor.to(device)

            mask = np.array(msk[num\_image], dtype=np.float32)

            mask[mask >= 2] = 1

            mask\_tensor = transform(mask).float()

            mask\_tensor=mask\_tensor.to(device)

            outputs = model(image\_tensor.unsqueeze(0))

            loss1 = criteron(outputs, mask\_tensor.unsqueeze(0))

            metric1=iou\_pytorch(outputs,mask\_tensor)

            optimizer.zero\_grad()

            loss1.backward()

            optimizer.step()

            \_, preds = torch.max(outputs, 1)

            loss1=loss1.to(device)

            loss += loss1.item()

            metric += metric1.item()

            correct += torch.sum(preds == mask\_tensor.data)

После завершения работы с тренировочными данными необходимо проверить работоспособность на валидационных данных. Для это необходимо использовать похожий цикл, однако при этом не будут обновляться значения параметров модели.

with torch.no\_grad():

            for val\_num\_file in tqdm(val\_ids):

              itk\_img=itk.ReadImage(f'{path}/{val\_num\_file}\_image.nii.gz')

              img = itk.GetArrayFromImage(itk\_img)

              itk\_img=itk.ReadImage(f'{path}/{val\_num\_file}\_label.nii.gz')

              msk = itk.GetArrayFromImage(itk\_img)

              for num\_image in range(0, img.shape[0], 1):

                  image = np.array(img[num\_image].copy())

                  image\_tensor = transform(image).float()

                  image\_tensor = image\_tensor.to(device)

                  mask = np.array(msk[num\_image], dtype=np.float32)

                  mask[mask >=2] = 1

                  mask\_tensor = transform(mask).float()

                  mask\_tensor=mask\_tensor.to(device)

                  val\_outputs = model(image\_tensor.unsqueeze(0))

                  val\_loss1=criteron(val\_outputs,mask\_tensor.unsqueeze(0))

                  val\_metric1=iou\_pytorch(val\_outputs, mask\_tensor)

                  optimizer.zero\_grad()

                  \_, val\_preds = torch.max(val\_outputs, 1)

                  val\_loss += val\_loss1.item()

                  val\_metric += val\_metric1.item()

                  val\_correct += torch.sum(val\_preds == mask\_tensor.data)

После того как будет валидация будет успешна завершена необходимо подсчитать и сохранить данные о лоссах и метриках на обучающей и валидационной выборках для дальнейшего анализа, а также сохраним на диск модель полученной.

epoch\_loss=loss/len(train\_ids)

epoch\_acc=correct.float()/len(train\_ids)

epoch\_metric=metric/len(train\_ids)

loss\_history.append(epoch\_loss)

metric\_history.append(epoch\_metric)

correct\_history.append(epoch\_acc)

val\_epoch\_loss=val\_loss/len(val\_ids)

val\_epoch\_acc=val\_correct.float()/len(val\_ids)

val\_epoch\_metric=val\_metric/len(val\_ids)

val\_loss\_history.append(val\_epoch\_loss)

val\_metric\_history.append(epoch\_metric)

val\_correct\_history.append(val\_epoch\_acc)

model\_history.append(model)

torch.save(model, f'/content/drive/MyDrive/new\_model{e+4}.pth')

После этого обучающую и валидационную выборку необходимо запустить на несколько эпох для ее обучения.

Реализация тестирование ничем не отличается от реализации валидации, только за исключением того, что данные используются для тестирования работоспособности нейронной сети.

test\_loss\_history=[]

test\_metric\_history=[]

test\_correct\_history=[]

for num\_file in tqdm(test\_ids):

………

    else:

        test\_epoch\_loss=loss/len(test\_ids)

        test\_epoch\_acc=correct.float()/len(test\_ids)

        test\_epoch\_metric=metric/len(test\_ids)

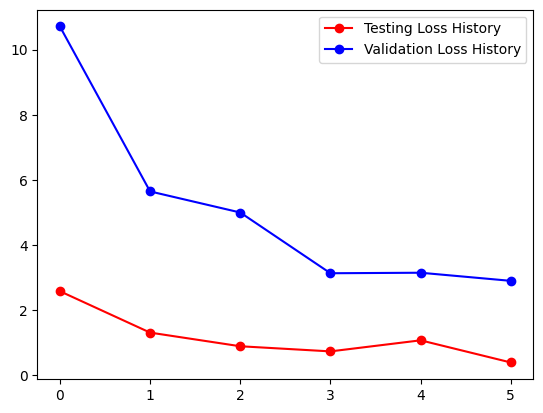
        test\_loss\_history.append(test\_epoch\_loss)

        test\_metric\_history.append(test\_epoch\_metric)

        test\_correct\_history.append(test\_epoch\_acc)

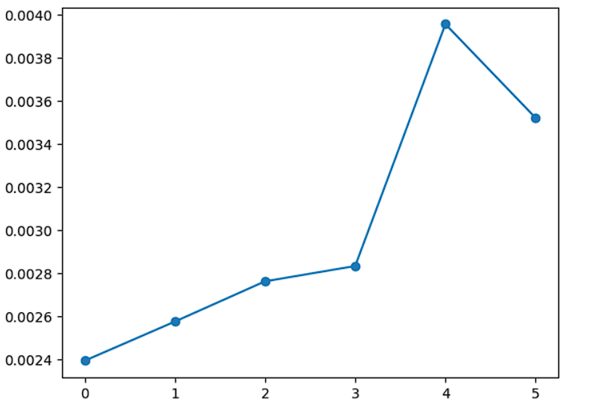
**Анализ получившихся данных**

При обучении нейронной сети было применено 6 эпох, в которых рассчитывались лоссы и метрики.



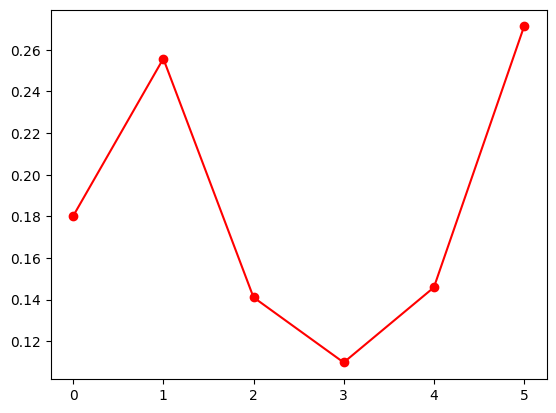
По результатам можно увидеть, что на первой эпохе валидационной значение имеет сильную погрешность в отличии от обучающей. В последующих эпохах валидация, как и лоссы уменьшается, что говорит о правильной работе нейронной сети.

В следующем графике рассматривается как меняется метрика IoU в зависимости от эпохи.

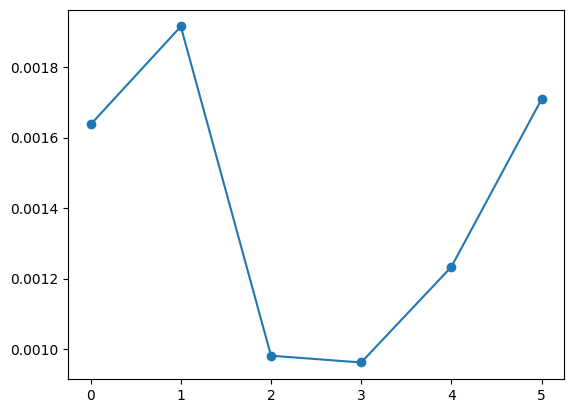


По данному графику видно, что максимальная метрика за все 6 эпох была на 5 эпохе, а следовательно ее и будет применять при тестировании нейронной сети.

После тестирования нейронной сети на 6 файлах погрешность для каждого файла указывает на то, что нейронной сети необходимо больше эпох для обучения.



Метрики нейронной сети также указывают на это.



**Вывод**

Во время прохождения учебной практики мною была разработана модель обучения нейронной сети UNet, а также реализованы загрузка данных, обучение нейронной сети, ее тестирование и графическое отображение метрик и погрешностей обучения.

Из проделанного обучения можно сделать вывод о том, что нейронная сеть работает правильно, однако для более точного результата вычисления необходимо применять большее количество эпох в обучении.