Image Segmentation

Group members:

Hager Abouroumia 5943

Khlod Mohamed 5695

Yehia ElDallal 5607

Fall Semester 2022

Contents

1. INTRODUCTION

Image segmentation is the task of assigning each pixel a label which is a class that it belongs to. In binary segmentation, you only have two classes: foreground and background. In this assignment we will work on a binary segmentation problem and you will implement and test two famous architectures Fully Convolution Network and U-Net.

2. PREPARATIONS

Data processing is an essential and extremely important step because it has a significant impact on the final output.

The data processing in this project is divided into several sub-parts, namely:

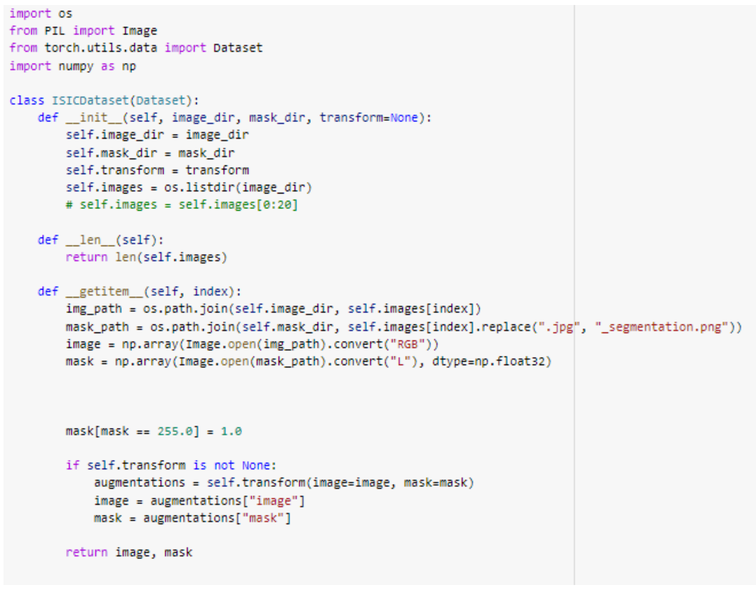
2.1. Upload the Data sets

Preparing the data by uploading it to ”colab” using ”!wget” and then decompressing the files using ”!unzip”. After that we remove unused files.

2.2. Data Loader and ISCDataset

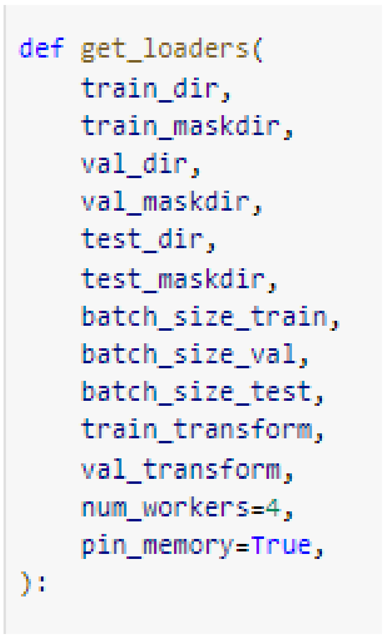
Store the Data sets to arrays after adding augmentation and convert 255 to 1.

ISICDataset: used to store the data in arrays and convert 255 to 1.

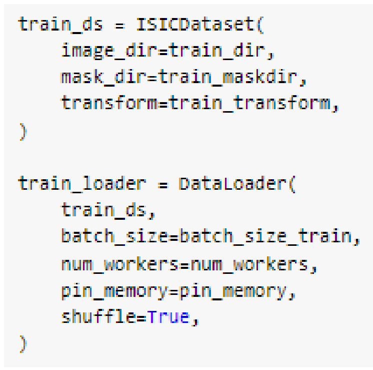


```python
import os
from PIL import Image
from torch.utils.data import Dataset
import numpy as np

class ISICDataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.images = os.listdir(image_dir)
        # self.images = self.images[0:20]

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        img_path = os.path.join(self.image_dir, self.images[index])
        mask_path = os.path.join(self.mask_dir, self.images[index].replace(".jpg", "_segmentation.png"))
        image = np.array(Image.open(img_path).convert("RGB"))
        mask = np.array(Image.open(mask_path).convert("L"), dtype=np.float32)



        mask[mask == 255.0] = 1.0

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

        return image, mask
```

getloaders: it takes the directions of the data sets and some information like batch size and returns arrays that contain the data which is ready to train.



```python
def get_loaders(
    train_dir,
    train_maskdir,
    val_dir,
    val_maskdir,
    test_dir,
    test_maskdir,
    batch_size_train,
    batch_size_val,
    batch_size_test,
    train_transform,
    val_transform,
    num_workers=4,
    pin_memory=True,
):
```

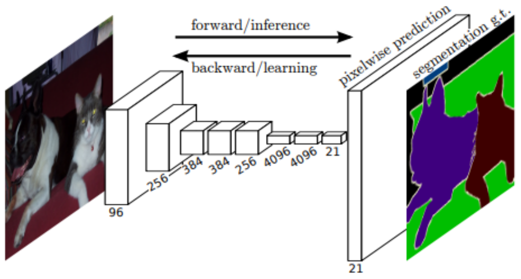


```python
train_ds = ISICDataset(
    image_dir=train_dir,
    mask_dir=train_maskdir,
    transform=train_transform,
)

train_loader = DataLoader(
    train_ds,
    batch_size=batch_size_train,
    num_workers=num_workers,
    pin_memory=pin_memory,
    shuffle=True,
)
```

3. FCN MODEL

3.1. Architecture

It is implemented using a VGG-16 backbone but instead of the 3 fully connected classification layers, the first two are replaced with a 1x1 convolution with the same number of filters. The last layer is replaced with 1x1 convolution layer with the number of filters equals the number of classes. Finally a deconvolutional layer that upsamples the output to the original image size is added to produce the final prediction.



3.2. Code

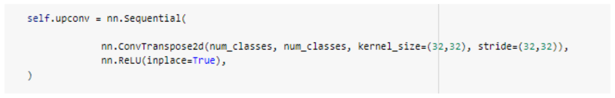
to Create **VGG16** model with output (7*7*512)

```python
def create_conv_layers(self, architecture):
    layers = []
    in_channels = self.in_channels

    for x in architecture:
        if type(x) == int:
            out_channels = x

            layers += [
                nn.Conv2d(
                    in_channels=in_channels,
                    out_channels=out_channels,
                    kernel_size=(3, 3),
                    stride=(1, 1),
                    padding=(1, 1),
                ),
                nn.BatchNorm2d(x),
                nn.ReLU(),
            ]
            in_channels = x
        elif x == "M":
            layers += [nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))]

    return nn.Sequential(*layers)
```

Rather than fully connected neural network, the first two are replaced with a **1x1 convolution** with the same number of filters. The last layer is replaced with 1x1 convolution layer with the number of filters equals the number of classes which is one.

```python
self.fcs = nn.Sequential(
    nn.Conv2d(
        in_channels=512,
        out_channels=512,
        kernel_size=(1, 1),
        stride=(1, 1),
    ),
    # nn.BatchNorm2d(x),
    nn.ReLU(),
    nn.Conv2d(
        in_channels=512,
        out_channels=512,
        kernel_size=(1, 1),
        stride=(1, 1),
    ),
    # nn.BatchNorm2d(x),
    nn.ReLU(),
    nn.Conv2d(
        in_channels=512,
        out_channels=num_classes,
        kernel_size=(1, 1),
        stride=(1, 1),
    ),
      )
```

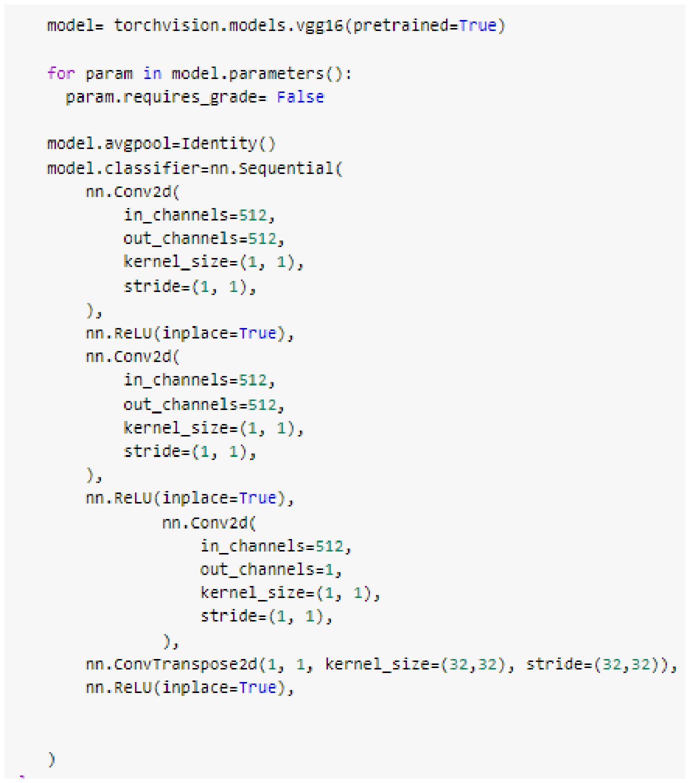
Finally, adding the last layer to return the size to (224*224*number of classes)



```
self.upconv = nn.Sequential(

            nn.ConvTranspose2d(num_classes, num_classes, kernel_size=(32,32), stride=(32,32)),
            nn.ReLU(inplace=True),
)
```

Pretrained Model

pre-trained model with ImageNet weights.



```
model= torchvision.models.vgg16(pretrained=True)

for param in model.parameters():
  param.requires_grade= False

model.avgpool=Identity()
model.classifier=nn.Sequential(
    nn.Conv2d(
        in_channels=512,
        out_channels=512,
        kernel_size=(1, 1),
        stride=(1, 1),
    ),
    nn.ReLU(inplace=True),
    nn.Conv2d(
        in_channels=512,
        out_channels=512,
        kernel_size=(1, 1),
        stride=(1, 1),
    ),
    nn.ReLU(inplace=True),
            nn.Conv2d(
                in_channels=512,
                out_channels=1,
                kernel_size=(1, 1),
                stride=(1, 1),
            ),
    nn.ConvTranspose2d(1, 1, kernel_size=(32,32), stride=(32,32)),
    nn.ReLU(inplace=True),


)
```

4. U-NET MODEL

UNet is a convolutional neural network architecture that expanded with few changes in the **CNN architecture**. It was invented to deal with biomedical images where the target is not only to classify whether there is an infection or not but also to identify the area of infection.

4.1. Architecture

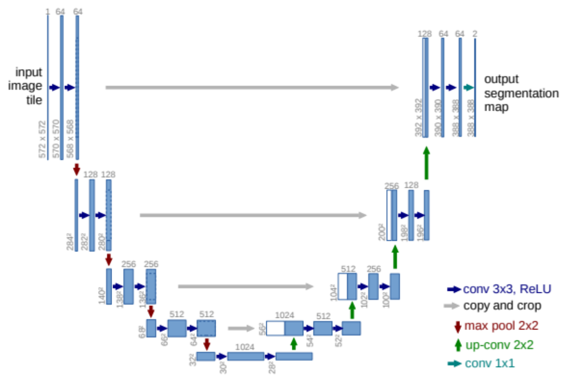
- 2@Conv layers means that two consecutive Convolution Layers are applied
- c1, c2, ... c9 are the output tensors of Convolutional Layers
- p1, p2, p3 and p4 are the output tensors of Max Pooling Layers
- u6, u7, u8 and u9 are the output tensors of up-sampling (transposed convolutional) layers
- The left hand side is the contraction path (Encoder) where we apply regular convolutions and max pooling layers.
- In the Encoder, the size of the image gradually reduces while the depth gradually increases. Starting from 128x128x3 to 8x8x256
- This basically means the network learns the “WHAT” information in the image, however it has lost the “WHERE” information
- The right hand side is the expansion path (Decoder) where we apply transposed convolutions along with regular convolutions
- In the decoder, the size of the image gradually increases and the depth gradually decreases. Starting from 8x8x256 to 128x128x1

- Intuitively, the Decoder recovers the “WHERE” information (precise localization) by gradually applying upsampling

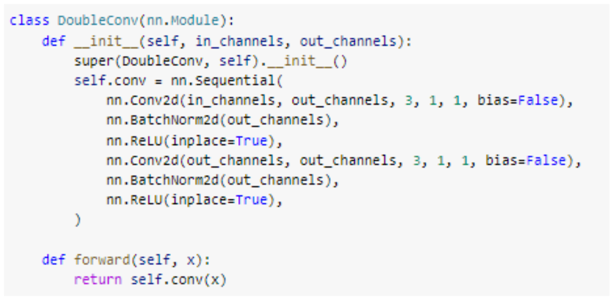
- To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level: u6 = u6 + c4 u7 = u7 + c3 u8 = u8 + c2 u9 = u9 + c1 After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output

- This is what gives the architecture a symmetric U-shape, hence the name UNET

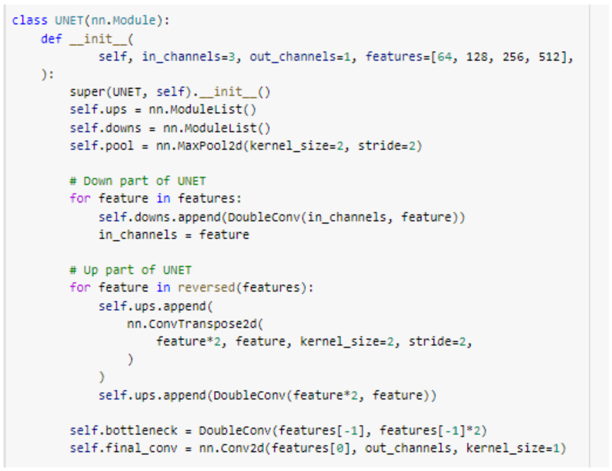
- On a high level, we have the following relationship: Input (128x128x1) : Encoder : (8x8x256) : Decoder : Ouput (128x128x1)



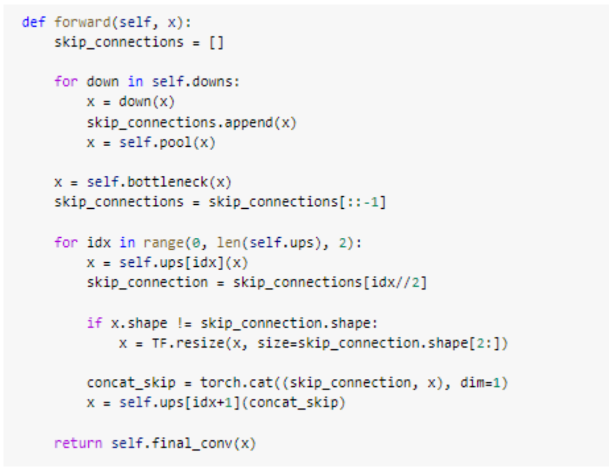
4.2. Code



```python
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            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)
```



```python
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
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET
        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            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)
```



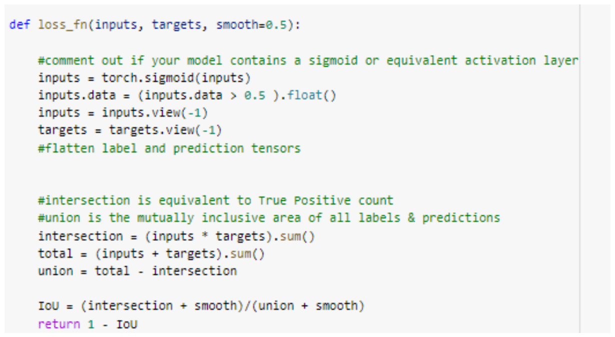
```python
def forward(self, x):
    skip_connections = []

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

    x = self.bottleneck(x)
    skip_connections = skip_connections[::-1]

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

        if x.shape != skip_connection.shape:
            x = TF.resize(x, size=skip_connection.shape[2:])

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

    return self.final_conv(x)
```

5. IOU LOSS

IOU=

$$\frac{T \cap P}{T \cup P}$$

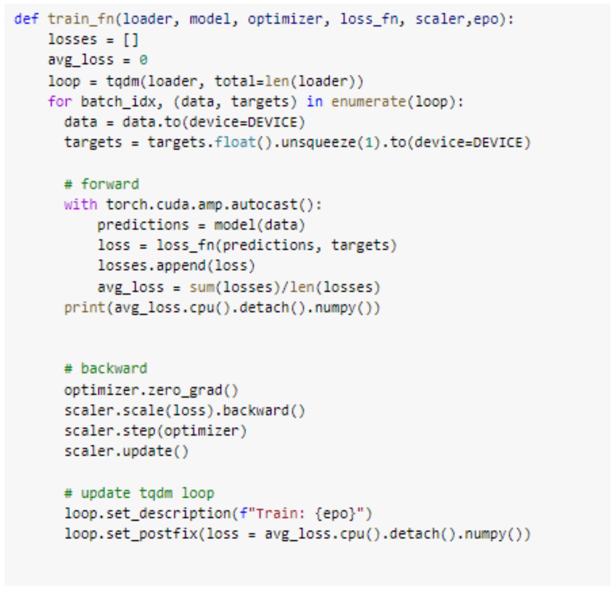
T stands for the true label image, **P** for the prediction of the output image and the symbols are taken from set theory. This IoU is then taken as the average over the entire set to be considered producing an IoU value between 0 and 1.



```python
def loss_fn(inputs, targets, smooth=0.5):

    #comment out if your model contains a sigmoid or equivalent activation layer
    inputs = torch.sigmoid(inputs)
    inputs.data = (inputs.data > 0.5 ).float()
    inputs = inputs.view(-1)
    targets = targets.view(-1)
    #flatten label and prediction tensors


    #intersection is equivalent to True Positive count
    #union is the mutually inclusive area of all labels & predictions
    intersection = (inputs * targets).sum()
    total = (inputs + targets).sum()
    union = total - intersection

    IoU = (intersection + smooth)/(union + smooth)
    return 1 - IoU
```

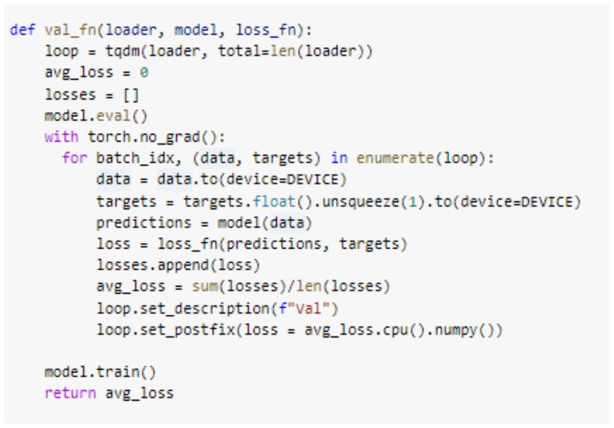
6. TRAIN

Model is compiled with **Adam** optimizer and we use **IOU** loss function. and use checkpoint file.



```python
def train_fn(loader, model, optimizer, loss_fn, scaler,epo):
    losses = []
    avg_loss = 0
    loop = tqdm(loader, total=len(loader))
    for batch_idx, (data, targets) in enumerate(loop):
        data = data.to(device=DEVICE)
        targets = targets.float().unsqueeze(1).to(device=DEVICE)

        # forward
        with torch.cuda.amp.autocast():
            predictions = model(data)
            loss = loss_fn(predictions, targets)
            losses.append(loss)
            avg_loss = sum(losses)/len(losses)
        print(avg_loss.cpu().detach().numpy())


        # backward
        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        # update tqdm loop
        loop.set_description(f"Train: {epo}")
        loop.set_postfix(loss = avg_loss.cpu().detach().numpy())
```

7. VALIDATION



```python
def val_fn(loader, model, loss_fn):
    loop = tqdm(loader, total=len(loader))
    avg_loss = 0
    losses = []
    model.eval()
    with torch.no_grad():
      for batch_idx, (data, targets) in enumerate(loop):
          data = data.to(device=DEVICE)
          targets = targets.float().unsqueeze(1).to(device=DEVICE)
          predictions = model(data)
          loss = loss_fn(predictions, targets)
          losses.append(loss)
          avg_loss = sum(losses)/len(losses)
          loop.set_description(f"Val")
          loop.set_postfix(loss = avg_loss.cpu().numpy())

    model.train()
    return avg_loss
```

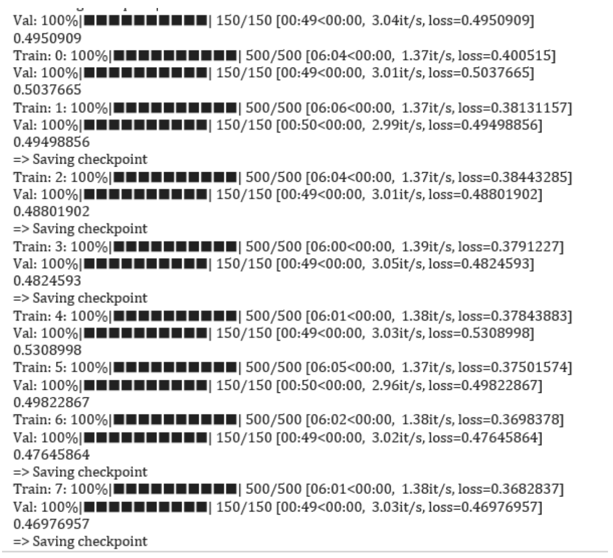
8. RESULTS

8.1. FCN

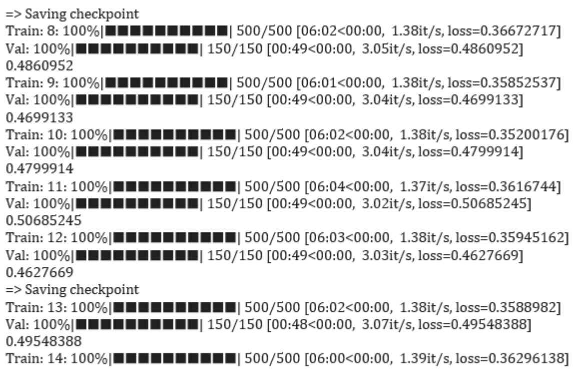
- number of epochs= 50
- Learning rate used=

$$1 * e^-3, 1 * e^-4, 1 * e^-5$$

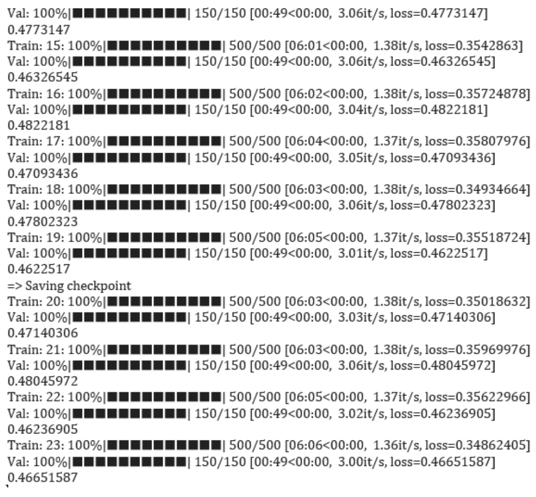
- Bach size= 4



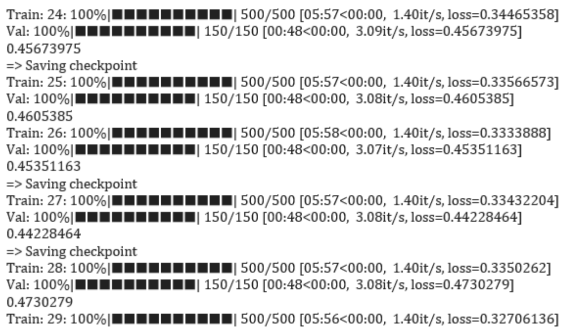
```
Val: 100%|███████████| 150/150 [00:49<00:00,  3.04it/s, loss=0.4950909]
0.4950909
Train: 0: 100%|███████████| 500/500 [06:04<00:00,  1.37it/s, loss=0.400515]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.5037665]
0.5037665
Train: 1: 100%|███████████| 500/500 [06:06<00:00,  1.37it/s, loss=0.38131157]
Val: 100%|███████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.49498856]
0.49498856
=> Saving checkpoint
Train: 2: 100%|███████████| 500/500 [06:04<00:00,  1.37it/s, loss=0.38443285]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.48801902]
0.48801902
=> Saving checkpoint
Train: 3: 100%|███████████| 500/500 [06:00<00:00,  1.39it/s, loss=0.3791227]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.05it/s, loss=0.4824593]
0.4824593
=> Saving checkpoint
Train: 4: 100%|███████████| 500/500 [06:01<00:00,  1.38it/s, loss=0.37843883]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.03it/s, loss=0.5308998]
0.5308998
Train: 5: 100%|███████████| 500/500 [06:05<00:00,  1.37it/s, loss=0.37501574]
Val: 100%|███████████| 150/150 [00:50<00:00,  2.96it/s, loss=0.49822867]
0.49822867
Train: 6: 100%|███████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.3698378]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.02it/s, loss=0.47645864]
0.47645864
=> Saving checkpoint
Train: 7: 100%|███████████| 500/500 [06:01<00:00,  1.38it/s, loss=0.3682837]
Val: 100%|███████████| 150/150 [00:49<00:00,  3.03it/s, loss=0.46976957]
0.46976957
=> Saving checkpoint
```



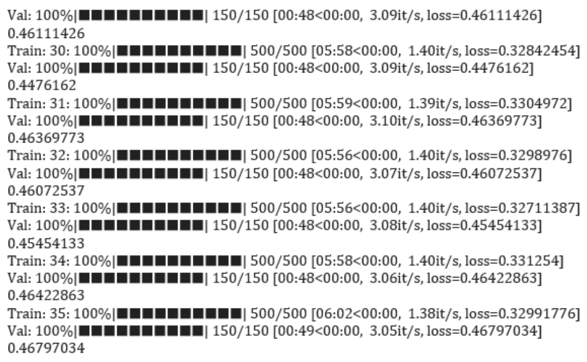
```
=> Saving checkpoint
Train: 8: 100%|██████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.36672717]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.05it/s, loss=0.4860952]
0.4860952
Train: 9: 100%|██████████| 500/500 [06:01<00:00,  1.38it/s, loss=0.35852537]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.04it/s, loss=0.4699133]
0.4699133
Train: 10: 100%|██████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.35200176]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.04it/s, loss=0.4799914]
0.4799914
Train: 11: 100%|██████████| 500/500 [06:04<00:00,  1.37it/s, loss=0.3616744]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.02it/s, loss=0.50685245]
0.50685245
Train: 12: 100%|██████████| 500/500 [06:03<00:00,  1.38it/s, loss=0.35945162]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.03it/s, loss=0.4627669]
0.4627669
=> Saving checkpoint
Train: 13: 100%|██████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.3588982]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.07it/s, loss=0.49548388]
0.49548388
Train: 14: 100%|██████████| 500/500 [06:00<00:00,  1.39it/s, loss=0.36296138]
```



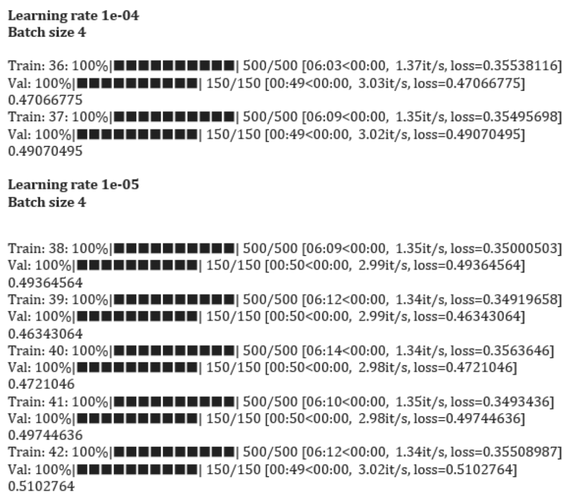
```
Val: 100%|██████████| 150/150 [00:49<00:00,  3.06it/s, loss=0.4773147]
0.4773147
Train: 15: 100%|██████████| 500/500 [06:01<00:00,  1.38it/s, loss=0.3542863]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.06it/s, loss=0.46326545]
0.46326545
Train: 16: 100%|██████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.35724878]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.04it/s, loss=0.4822181]
0.4822181
Train: 17: 100%|██████████| 500/500 [06:04<00:00,  1.37it/s, loss=0.35807976]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.05it/s, loss=0.47093436]
0.47093436
Train: 18: 100%|██████████| 500/500 [06:03<00:00,  1.38it/s, loss=0.34934664]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.06it/s, loss=0.47802323]
0.47802323
Train: 19: 100%|██████████| 500/500 [06:05<00:00,  1.37it/s, loss=0.35518724]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.4622517]
0.4622517
=> Saving checkpoint
Train: 20: 100%|██████████| 500/500 [06:03<00:00,  1.38it/s, loss=0.35018632]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.03it/s, loss=0.47140306]
0.47140306
Train: 21: 100%|██████████| 500/500 [06:03<00:00,  1.38it/s, loss=0.35969976]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.06it/s, loss=0.48045972]
0.48045972
Train: 22: 100%|██████████| 500/500 [06:05<00:00,  1.37it/s, loss=0.35622966]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.02it/s, loss=0.46236905]
0.46236905
Train: 23: 100%|██████████| 500/500 [06:06<00:00,  1.36it/s, loss=0.34862405]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.00it/s, loss=0.46651587]
0.46651587
```



```
Train: 24: 100%|██████████| 500/500 [05:57<00:00,  1.40it/s, loss=0.34465358]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.09it/s, loss=0.45673975]
0.45673975
=> Saving checkpoint
Train: 25: 100%|██████████| 500/500 [05:57<00:00,  1.40it/s, loss=0.33566573]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.08it/s, loss=0.4605385]
0.4605385
Train: 26: 100%|██████████| 500/500 [05:58<00:00,  1.40it/s, loss=0.3333888]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.07it/s, loss=0.45351163]
0.45351163
=> Saving checkpoint
Train: 27: 100%|██████████| 500/500 [05:57<00:00,  1.40it/s, loss=0.33432204]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.08it/s, loss=0.44228464]
0.44228464
=> Saving checkpoint
Train: 28: 100%|██████████| 500/500 [05:57<00:00,  1.40it/s, loss=0.3350262]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.08it/s, loss=0.4730279]
0.4730279
Train: 29: 100%|██████████| 500/500 [05:56<00:00,  1.40it/s, loss=0.32706136]
```



```
Val: 100%|██████████| 150/150 [00:48<00:00,  3.09it/s, loss=0.46111426]
0.46111426
Train: 30: 100%|██████████| 500/500 [05:58<00:00,  1.40it/s, loss=0.32842454]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.09it/s, loss=0.4476162]
0.4476162
Train: 31: 100%|██████████| 500/500 [05:59<00:00,  1.39it/s, loss=0.3304972]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.10it/s, loss=0.46369773]
0.46369773
Train: 32: 100%|██████████| 500/500 [05:56<00:00,  1.40it/s, loss=0.3298976]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.07it/s, loss=0.46072537]
0.46072537
Train: 33: 100%|██████████| 500/500 [05:56<00:00,  1.40it/s, loss=0.32711387]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.08it/s, loss=0.45454133]
0.45454133
Train: 34: 100%|██████████| 500/500 [05:58<00:00,  1.40it/s, loss=0.331254]
Val: 100%|██████████| 150/150 [00:48<00:00,  3.06it/s, loss=0.46422863]
0.46422863
Train: 35: 100%|██████████| 500/500 [06:02<00:00,  1.38it/s, loss=0.32991776]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.05it/s, loss=0.46797034]
0.46797034
```

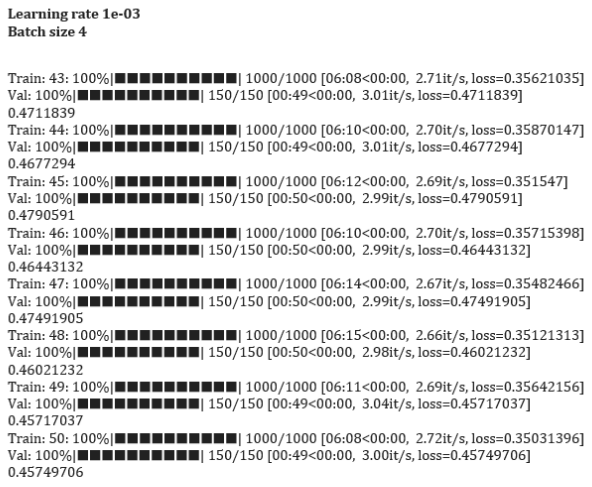


Learning rate 1e-04
Batch size 4

```
Train: 36: 100%|██████████| 500/500 [06:03<00:00,  1.37it/s, loss=0.35538116]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.03it/s, loss=0.47066775]
0.47066775
Train: 37: 100%|██████████| 500/500 [06:09<00:00,  1.35it/s, loss=0.35495698]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.02it/s, loss=0.49070495]
0.49070495
```

Learning rate 1e-05
Batch size 4

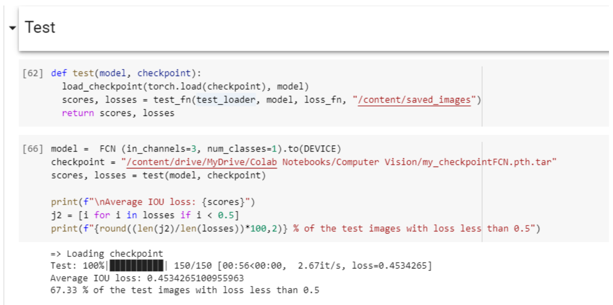
```
Train: 38: 100%|██████████| 500/500 [06:09<00:00,  1.35it/s, loss=0.35000503]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.49364564]
0.49364564
Train: 39: 100%|██████████| 500/500 [06:12<00:00,  1.34it/s, loss=0.34919658]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.46343064]
0.46343064
Train: 40: 100%|██████████| 500/500 [06:14<00:00,  1.34it/s, loss=0.3563646]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.98it/s, loss=0.4721046]
0.4721046
Train: 41: 100%|██████████| 500/500 [06:10<00:00,  1.35it/s, loss=0.3493436]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.98it/s, loss=0.49744636]
0.49744636
Train: 42: 100%|██████████| 500/500 [06:12<00:00,  1.34it/s, loss=0.35508987]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.02it/s, loss=0.5102764]
0.5102764
```



Learning rate 1e-03
Batch size 4

```
Train: 43: 100%|██████████| 1000/1000 [06:08<00:00,  2.71it/s, loss=0.35621035]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.4711839]
0.4711839
Train: 44: 100%|██████████| 1000/1000 [06:10<00:00,  2.70it/s, loss=0.35870147]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.4677294]
0.4677294
Train: 45: 100%|██████████| 1000/1000 [06:12<00:00,  2.69it/s, loss=0.351547]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.4790591]
0.4790591
Train: 46: 100%|██████████| 1000/1000 [06:10<00:00,  2.70it/s, loss=0.35715398]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.46443132]
0.46443132
Train: 47: 100%|██████████| 1000/1000 [06:14<00:00,  2.67it/s, loss=0.35482466]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.99it/s, loss=0.47491905]
0.47491905
Train: 48: 100%|██████████| 1000/1000 [06:15<00:00,  2.66it/s, loss=0.35121313]
Val: 100%|██████████| 150/150 [00:50<00:00,  2.98it/s, loss=0.46021232]
0.46021232
Train: 49: 100%|██████████| 1000/1000 [06:11<00:00,  2.69it/s, loss=0.35642156]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.04it/s, loss=0.45717037]
0.45717037
Train: 50: 100%|██████████| 1000/1000 [06:08<00:00,  2.72it/s, loss=0.35031396]
Val: 100%|██████████| 150/150 [00:49<00:00,  3.00it/s, loss=0.45749706]
0.45749706
```

Loss in Test Data set: 0.453



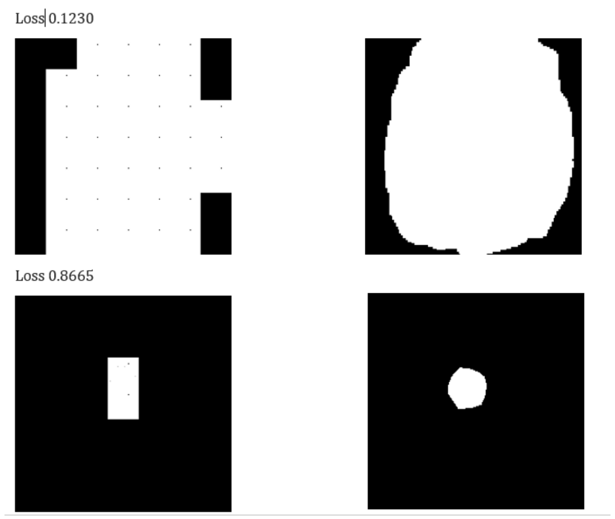
Test

```
[62] def test(model, checkpoint):
       load_checkpoint(torch.load(checkpoint), model)
       scores, losses = test_fn(test_loader, model, loss_fn, "/content/saved_images")
       return scores, losses
```

```
[66] model =  FCN (in_channels=3, num_classes=1).to(DEVICE)
     checkpoint = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/my_checkpointFCN.pth.tar"
     scores, losses = test(model, checkpoint)

     print(f"\nAverage IOU loss: {scores}")
     j2 = [i for i in losses if i < 0.5]
     print(f"{round((len(j2)/len(losses))*100,2)} % of the test images with loss less than 0.5")
```

```
=> Loading checkpoint
Test: 100%|██████████| 150/150 [00:56<00:00,  2.67it/s, loss=0.4534265]
Average IOU loss: 0.4534265100955963
67.33 % of the test images with loss less than 0.5
```

print some samples:

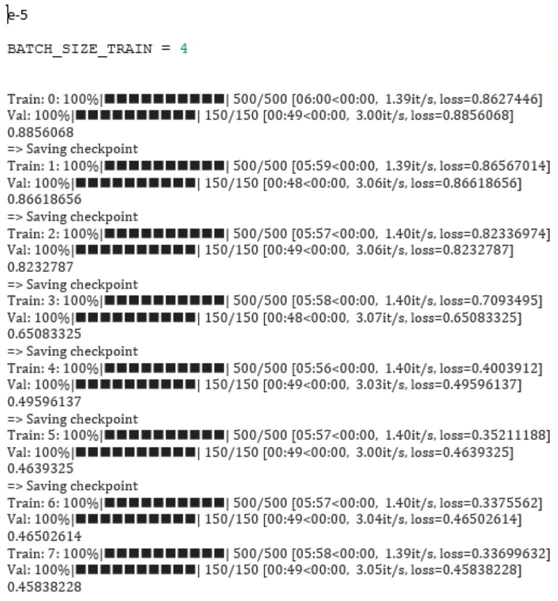


8.2. Pretrained FCN

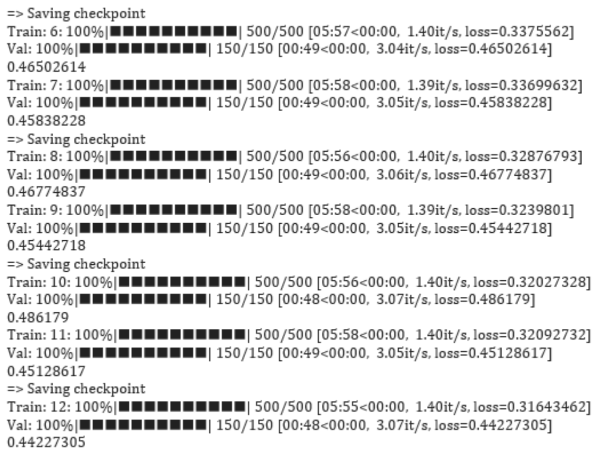
- number of epochs= 50
- Learning rate used=

$$1 * e^{-}6, 1 * e^{-}5$$

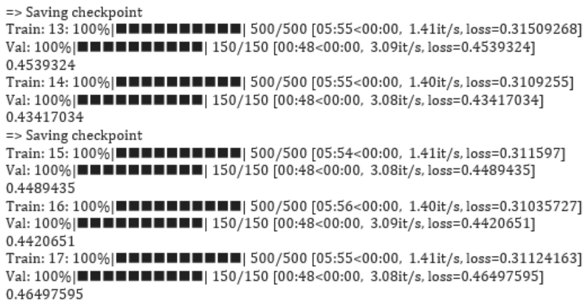
- Bach size= 4



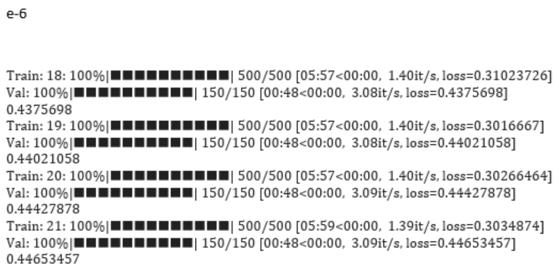
```
e-5

BATCH_SIZE_TRAIN = 4

Train: 0: 100%|■■■■■■■■■■| 500/500 [06:00<00:00,  1.39it/s, loss=0.8627446]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.00it/s, loss=0.8856068]
0.8856068
=> Saving checkpoint
Train: 1: 100%|■■■■■■■■■■| 500/500 [05:59<00:00,  1.39it/s, loss=0.86567014]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.06it/s, loss=0.86618656]
0.86618656
=> Saving checkpoint
Train: 2: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.82336974]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.06it/s, loss=0.8232787]
0.8232787
=> Saving checkpoint
Train: 3: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.40it/s, loss=0.7093495]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.07it/s, loss=0.65083325]
0.65083325
=> Saving checkpoint
Train: 4: 100%|■■■■■■■■■■| 500/500 [05:56<00:00,  1.40it/s, loss=0.4003912]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.03it/s, loss=0.49596137]
0.49596137
=> Saving checkpoint
Train: 5: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.35211188]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.00it/s, loss=0.4639325]
0.4639325
=> Saving checkpoint
Train: 6: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.3375562]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.04it/s, loss=0.46502614]
0.46502614
Train: 7: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.39it/s, loss=0.33699632]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.05it/s, loss=0.45838228]
0.45838228
```



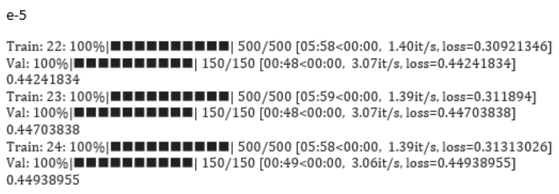
```
=> Saving checkpoint
Train: 6: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.3375562]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.04it/s, loss=0.46502614]
0.46502614
Train: 7: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.39it/s, loss=0.33699632]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.05it/s, loss=0.45838228]
0.45838228
=> Saving checkpoint
Train: 8: 100%|■■■■■■■■■■| 500/500 [05:56<00:00,  1.40it/s, loss=0.32876793]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.06it/s, loss=0.46774837]
0.46774837
Train: 9: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.39it/s, loss=0.3239801]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.05it/s, loss=0.45442718]
0.45442718
=> Saving checkpoint
Train: 10: 100%|■■■■■■■■■■| 500/500 [05:56<00:00,  1.40it/s, loss=0.32027328]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.07it/s, loss=0.486179]
0.486179
Train: 11: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.40it/s, loss=0.32092732]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.05it/s, loss=0.45128617]
0.45128617
=> Saving checkpoint
Train: 12: 100%|■■■■■■■■■■| 500/500 [05:55<00:00,  1.40it/s, loss=0.31643462]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.07it/s, loss=0.44227305]
0.44227305
```



```
=> Saving checkpoint
Train: 13: 100%|■■■■■■■■■■| 500/500 [05:55<00:00,  1.41it/s, loss=0.31509268]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.09it/s, loss=0.4539324]
0.4539324
Train: 14: 100%|■■■■■■■■■■| 500/500 [05:55<00:00,  1.40it/s, loss=0.3109255]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.08it/s, loss=0.43417034]
0.43417034
=> Saving checkpoint
Train: 15: 100%|■■■■■■■■■■| 500/500 [05:54<00:00,  1.41it/s, loss=0.311597]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.08it/s, loss=0.4489435]
0.4489435
Train: 16: 100%|■■■■■■■■■■| 500/500 [05:56<00:00,  1.40it/s, loss=0.31035727]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.09it/s, loss=0.4420651]
0.4420651
Train: 17: 100%|■■■■■■■■■■| 500/500 [05:55<00:00,  1.41it/s, loss=0.31124163]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.08it/s, loss=0.46497595]
0.46497595
```

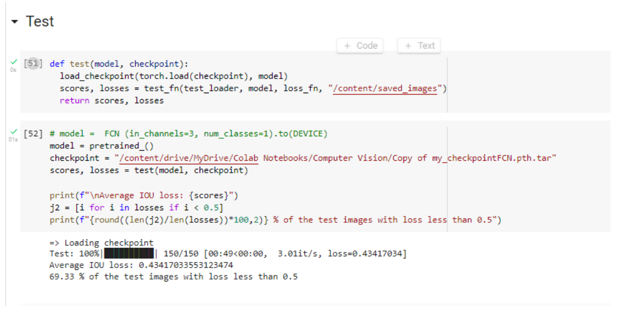


```
e-6

Train: 18: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.31023726]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.08it/s, loss=0.4375698]
0.4375698
Train: 19: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.3016667]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.08it/s, loss=0.44021058]
0.44021058
Train: 20: 100%|■■■■■■■■■■| 500/500 [05:57<00:00,  1.40it/s, loss=0.30266464]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.09it/s, loss=0.44427878]
0.44427878
Train: 21: 100%|■■■■■■■■■■| 500/500 [05:59<00:00,  1.39it/s, loss=0.3034874]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.09it/s, loss=0.44653457]
0.44653457
```



```
e-5

Train: 22: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.40it/s, loss=0.30921346]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.07it/s, loss=0.44241834]
0.44241834
Train: 23: 100%|■■■■■■■■■■| 500/500 [05:59<00:00,  1.39it/s, loss=0.311894]
Val: 100%|■■■■■■■■■■| 150/150 [00:48<00:00,  3.07it/s, loss=0.44703838]
0.44703838
Train: 24: 100%|■■■■■■■■■■| 500/500 [05:58<00:00,  1.39it/s, loss=0.31313026]
Val: 100%|■■■■■■■■■■| 150/150 [00:49<00:00,  3.06it/s, loss=0.44938955]
0.44938955
```

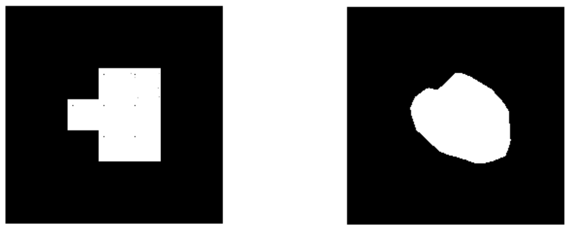
Loss in Test Data set: 0.434



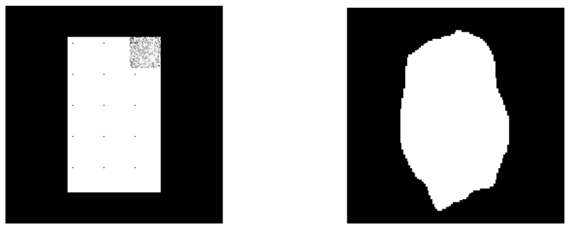
```
▾ Test

[51] def test(model, checkpoint):
       load_checkpoint(torch.load(checkpoint), model)
       scores, losses = test_fn(test_loader, model, loss_fn, "/content/saved_images")
       return scores, losses

[52] # model =  FCN (in_channels=3, num_classes=1).to(DEVICE)
     model = pretrained_()
     checkpoint = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/Copy of my_checkpointFCN.pth.tar"
     scores, losses = test(model, checkpoint)

     print(f"\nAverage IOU loss: {scores}")
     j2 = [i for i in losses if i < 0.5]
     print(f"{round((len(j2)/len(losses))*100,2)} % of the test images with loss less than 0.5")

     => Loading checkpoint
     Test: 100%|██████████| 150/150 [00:49<00:00,  3.01it/s, loss=0.43417034]
     Average IOU loss: 0.43417033553123474
     69.33 % of the test images with loss less than 0.5
```

print some samples:

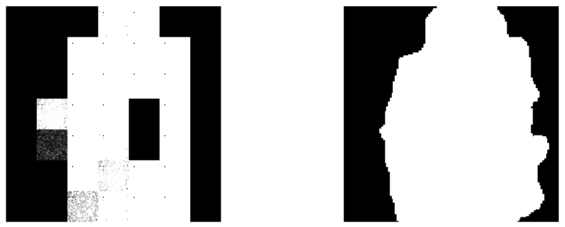
Loss 0.2593



Loss 0.1761



Loss 0.2074



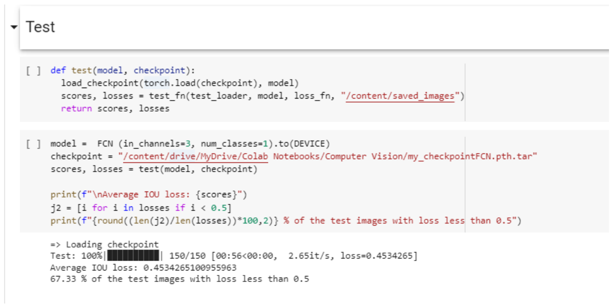
8.3. U-Net

- number of epochs= 50
- Learning rate used=

$$1 * e^{-}3, 1 * e^{-}4, 1 * e^{-}5$$

- Bach size= 4

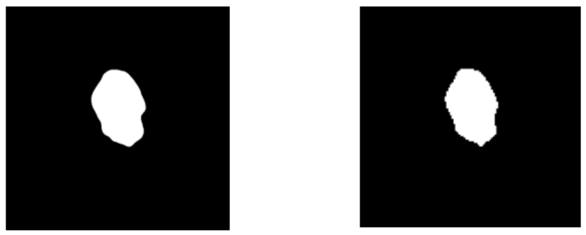
Loss in Test Data set: 0.453



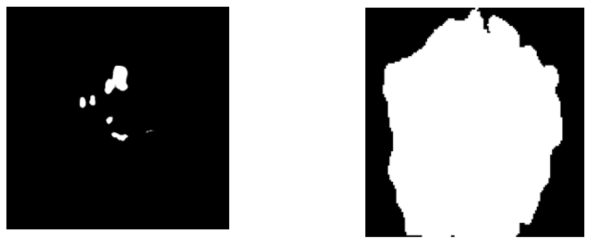
```
▾ Test

[ ] def test(model, checkpoint):
      load_checkpoint(torch.load(checkpoint), model)
      scores, losses = test_fn(test_loader, model, loss_fn, "/content/saved_images")
      return scores, losses

[ ] model =  FCN (in_channels=3, num_classes=1).to(DEVICE)
    checkpoint = "/content/drive/MyDrive/Colab Notebooks/Computer Vision/my_checkpointFCN.pth.tar"
    scores, losses = test(model, checkpoint)

    print(f"\nAverage IOU loss: {scores}")
    j2 = [i for i in losses if i < 0.5]
    print(f"{round((len(j2)/len(losses))*100,2)} % of the test images with loss less than 0.5")

    => Loading checkpoint
    Test: 100%|██████████| 150/150 [00:56<00:00,  2.65it/s, loss=0.4534265]
    Average IOU loss: 0.4534265100955963
    67.33 % of the test images with loss less than 0.5
```

print some samples:

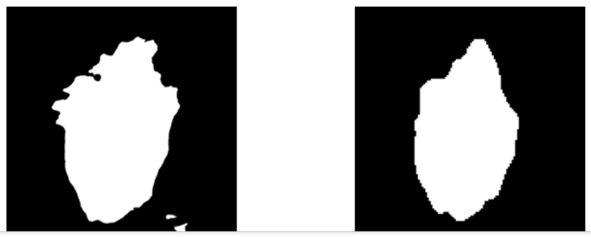
Loos 0.0385



Loss 0.9780



0.1597



0.3598