Computer Vision: Homework Assignment 2 - Describable Texture Dataset

Pascal Sager

December 12, 2020

The code from this project is available under:
`https://github.com/sagerpascal/describable-textures-dataset`.

1 Introduction

I had very limited computational power to train the models. Therefore, I decided to focus on the implementation instead of tuning the hyperparameters or running long-lasting trainings. As a result, I didn't achieve a good accuracy. Nevertheless, I think that I learned a lot about the whole topic of semantic segmentation, especially because I put a lot of effort in the implementation of different components. For example, I have not only tested the cross-entropy loss but also the dice loss and the BCE with logits loss. Additionally, I implemented a simple version of the U-Net and a more complex one with a pretrained Resnet-18 encoder besides the required simple FCN. By implementing these additional components, I had the opportunity to understand their functionality better. Overall, I enjoyed this lab a lot and was able to prepare well for the exam.

2 Used Technologies

I decided to use PyTorch instead of Keras and Tensorflow because I know this framework better and prefer it personally. Especially the concept of data types suits me more. The disadvantage of PyTorch is that the implementation takes much more effort. For example, I couldn't use the given example of the dataloader. I also implemented the calculation of metrics and the training loop by myself instead of using the predefined keras routines. But In return I could learn how this functions work in detail.
I also used wandb instead of Tensorboard. Even if I implemented Tensorboard and tested it, I manly worked with wandb, because it visualizes the results directly in the cloud and also stores the trained models there. Additionally, it allows to implement sweeps, which could be used to determine the hyper-parameters.

3 Data Exploration

I started with downloading the large (coloured) as well as the small (greyscale) data set. Then I examined the distribution of the labels. The smaller dataset is relatively even distributed. However, as figure 1 shows, the colored data is not perfectly balanced. Although it is not absolutely necessary I decided to weight the BCE with logits loss function accordingly. Therefore, classes with less labels have a higher weight than

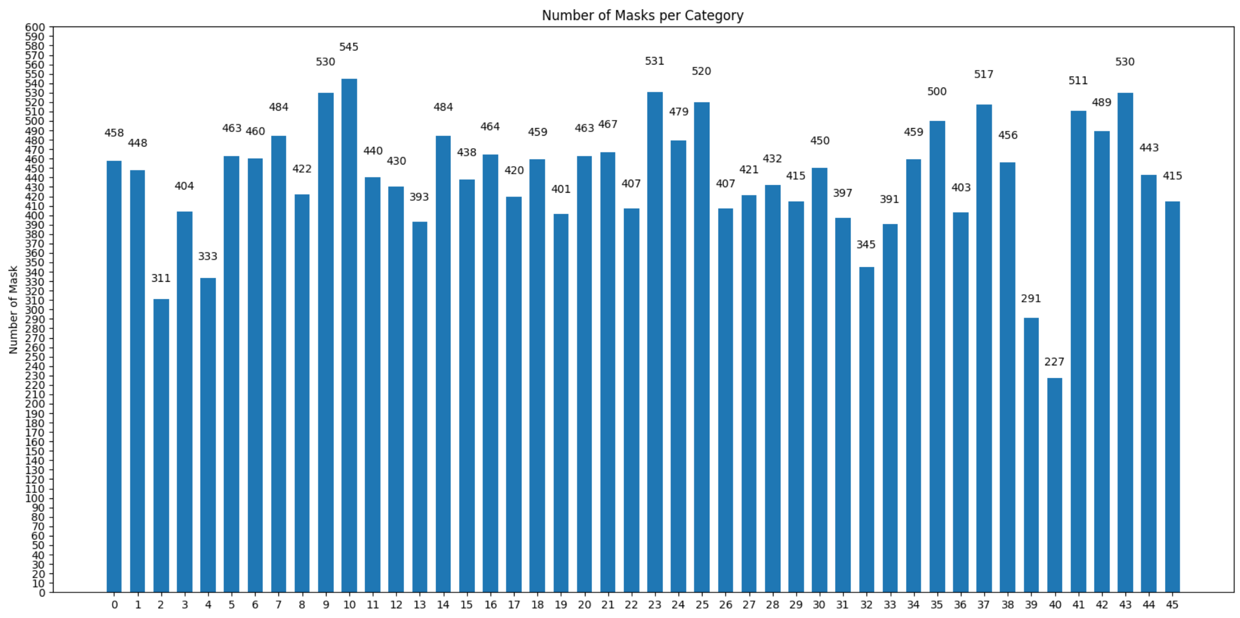

Figure 1: Number of labels per class in the large dataset

classes with more labels. However, when I used the weighted loss function, the result neither became worse nor improved, so I decided not to weight the other two loss functions. But if the code would be used for a "real" application, this should be examined more precisely.

4 Training

Since very limited computing power was available, only a limited training could be done:

- the smaller grayscale dataset was used
- both U-Nets could not really be trained
- only the cross entropy loss was used, the other loss function were only tested for functionality
- wandb sweeps were implemented but not used
- data augmentation was not used

But even if not all components were trained, their functionality was tested. The results of the training look promising and indicate a better accuracy could be achieved with better hardware. For example the loss of the simple FCN was still decreasing, as figure 2 shows. Nevertheless, I stopped the training due to time constraints. The loss curve indicates that the model overfitted. This could be due to the architecture, since only convolution layers were used. The overfitting could also be reduced by using the larger dataset or applying data augmentation.

The loss curve of the simple U-Net (see figure 3) and the U-Net with pretrained encoder (see figure 4) looks pretty bad. The networks learn on the training data, but almost not on the validation data. This is a clear indicator of overfitting, which could be reduced with more data (using the larger dataset) as well as data augmentation. In addition, this is also due to the fact that the networks have (too) many parameters. A test with fewer parameters shows that reducing the parameters would improve the performance on the validation set. However, I decided to develop the network according to [`https://arxiv.org/abs/1505.04597`], because I expected the network to perform well on the large dataset (which I couldn't test so far).

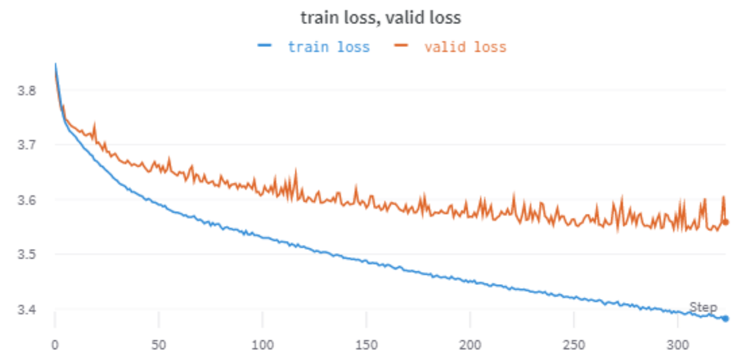

Figure 2: Loss of the simple FCN during training and validation

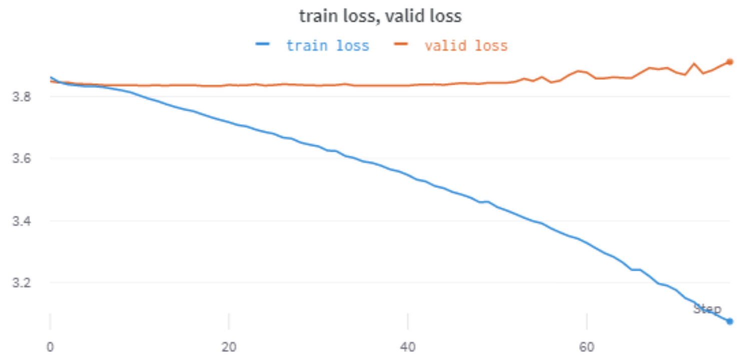

Figure 3: Loss of the simple U-Net during training and validation

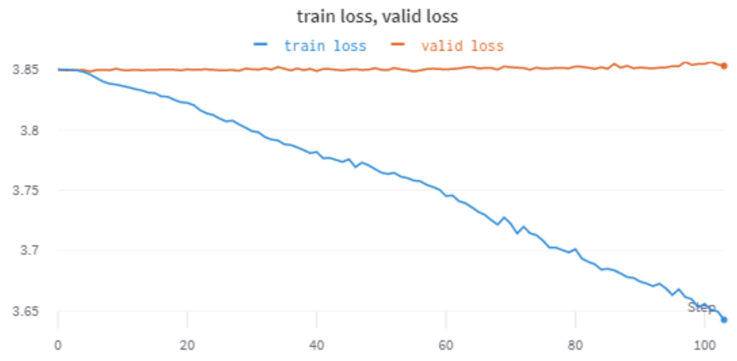

Figure 4: Loss of the U-Net with pretrained ResNet18 encoder during training and validation

5 Results

I created a script to evaluate the models. This script loads the saved model and can then optionally plot the predicted masks or calculate the accuracy on the test set. The top 1 accuracy of the different models is summarized in table 1. The results are not really good. It is expected that the results would be better on the large data set.

Of course, I could have used less parameters fot the U-Nets. However, I wanted to develop a network that was as close as possible to the original implementation. In addition, I expect that the more complex network will provide better performance on the large (colored) data set and on other segmentation tasks.

Model	Dataset	Top-1 Acc. Train	Top-1 Acc. Val	Top-1 Acc. Test
Simple FCN	Small	12.07%	9.67%	9.42%
Simple U-Net	Small	5.16%	3.16%	3.17%
Pretrained U-Net	Small	5.28%	3.31%	3.23%

Table 1: Results of the different models on the training, validation and test dataset

6 Appendix

6.1 Architecture Simple FCN

```
Model has 28427 parameters

SimpleFullyCnn(
  (layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1))
    (3): ReLU()
    (4): Conv2d(4, 8, kernel_size=(3, 3), stride=(1, 1))
    (5): ReLU()
    (6): Conv2d(8, 8, kernel_size=(3, 3), stride=(1, 1))
    (7): ReLU()
    (8): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
    (9): ReLU()
    (10): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
    (11): ReLU()
    (12): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (13): ReLU()
    (14): Conv2d(64, 47, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

6.2 Architecture Simple U-Net

```
Model has 117439663 parameters

SimpleUnet(
  (upfeature): FeatureMapBlock(
    (conv): Conv2d(1, 32, kernel_size=(1, 1), stride=(1, 1))
  )
  (contract1): ContractingBlock(
    (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (activation): LeakyReLU(negative_slope=0.2)
    (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (batchnorm): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (contract2): ContractingBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (activation): LeakyReLU(negative_slope=0.2)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
(contract3): ContractingBlock(
  (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (activation): LeakyReLU(negative_slope=0.2)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
(contract4): ContractingBlock(
  (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (activation): LeakyReLU(negative_slope=0.2)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(contract5): ContractingBlock(
  (conv1): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (activation): LeakyReLU(negative_slope=0.2)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(contract6): ContractingBlock(
  (conv1): Conv2d(1024, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(2048, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (activation): LeakyReLU(negative_slope=0.2)
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batchnorm): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(expand0): ExpandingBlock(
  (upsample): Upsample(scale_factor=2.0, mode=bilinear)
  (conv1): Conv2d(2048, 1024, kernel_size=(2, 2), stride=(1, 1))
  (conv2): Conv2d(2048, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(1024, 1024, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (activation): ReLU()
)
(expand1): ExpandingBlock(
  (upsample): Upsample(scale_factor=2.0, mode=bilinear)
  (conv1): Conv2d(1024, 512, kernel_size=(2, 2), stride=(1, 1))
  (conv2): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(512, 512, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (activation): ReLU()
)
(expand2): ExpandingBlock(
  (upsample): Upsample(scale_factor=2.0, mode=bilinear)
  (conv1): Conv2d(512, 256, kernel_size=(2, 2), stride=(1, 1))
  (conv2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(256, 256, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (activation): ReLU()
  )
  (expand3): ExpandingBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (conv1): Conv2d(256, 128, kernel_size=(2, 2), stride=(1, 1))
    (conv2): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(128, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (batchnorm): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (activation): ReLU()
  )
  (expand4): ExpandingBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (conv1): Conv2d(128, 64, kernel_size=(2, 2), stride=(1, 1))
    (conv2): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(64, 64, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (batchnorm): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (activation): ReLU()
  )
  (expand5): ExpandingBlock(
    (upsample): Upsample(scale_factor=2.0, mode=bilinear)
    (conv1): Conv2d(64, 32, kernel_size=(2, 2), stride=(1, 1))
    (conv2): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(32, 32, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
    (batchnorm): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (activation): ReLU()
  )
  (downfeature): FeatureMapBlock(
    (conv): Conv2d(32, 47, kernel_size=(1, 1), stride=(1, 1))
  )
)
```

6.3 Architecture Pretrained U-Net

```
Model has 14328607 parameters

Unet(
  (encoder): ResNetEncoder(
    (conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
  (decoder): UnetDecoder(
    (center): Identity()
    (blocks): ModuleList(
      (0): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(768, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
      (1): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(384, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
      (2): DecoderBlock(
        (conv1): Conv2dReLU(
        (0): Conv2d(192, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention1): Attention(
        (attention): Identity()
      )
      (conv2): Conv2dReLU(
        (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention2): Attention(
        (attention): Identity()
      )
    )
    (3): DecoderBlock(
      (conv1): Conv2dReLU(
        (0): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention1): Attention(
        (attention): Identity()
      )
      (conv2): Conv2dReLU(
        (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention2): Attention(
        (attention): Identity()
      )
    )
    (4): DecoderBlock(
      (conv1): Conv2dReLU(
        (0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention1): Attention(
        (attention): Identity()
      )
      (conv2): Conv2dReLU(
        (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
      (attention2): Attention(
        (attention): Identity()
      )
    )
  )
)
```

```
  (segmentation_head): SegmentationHead(
    (0): Conv2d(16, 47, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Identity()
    (2): Activation(
      (activation): Softmax(dim=1)
    )
  )
)
```