Hardik Prajapati

USC ID: 2678294168

Semantic Segmentation, ResNet, FCN32, FCN16

HomewoRK 5 rEPORT

CSCI 677

|  |  |  |
| --- | --- | --- |
| **Parameter** | **FCN32** | **FCN16** |
| **Epochs** | **20** | **20** |
| **Initial learning rate** | **0.001** | **0.001** |
| **Batch size** | **1** | **1** |
| **Optimizer** | **Adam()** | **Adam()** |
| **Train+Validation time** | **8822 sec** | **9862 sec** |
| **mIoU** | **0.1821** | **0.2769** |

1. **A brief description of the programs you write, including the source listing.**
2. Architecture: FCN32

Graphical user interface, text

Description automatically generated

Download the pretrained ResNet Network. Take the first 8 layers(till conv5\_x block )from this network building our backbone Network.

Add avgPool layer on top of this and then add FCN (fully convolutional layer). On top of this a transpose convolutional layer is added.

Text

Description automatically generated

This code is referred from: <https://github.com/wkentaro/pytorch-fcn/tree/main/torchfcn/models>. I have removed the first ‘if’ loop from the source in order to use the pre-trained weights of the ResNet as our initial weights.

Graphical user interface, text, application

Description automatically generated

This is the forward function of the architecture. At the end, I crop the output to match the dimension of the Ground truth.

1. Architecture: FCN16

Graphical user interface, text, application, email

Description automatically generated

In addition to the FCN32 architecture, we add another convolutional layer on the output of conv4\_x block. On top of this another transpose convolutional layer is added.

Graphical user interface, text, application

Description automatically generated

This is the forward function of FCN16 network. I have cropped at 2 places to match the dimension of tensor. The 2 upscores are added to finally have the output which is trimmed to match the dimension of gt\_mask.

1. Custom Dataset Class

Text

Description automatically generated

This is the dataset class I have created for Kitti dataset. It reads the image files, apply border of 100 pixels on each side of input image and then apply transforms if any. In getitem class, we return the input image and gt\_mask at particular index.

1. Dataset splitter

Graphical user interface, text, application, email

Description automatically generated

We split the dataset into train/val/test with 70%/15%/15% .We have total 200 images.

1. Experiment function

Graphical user interface, text, application, email

Description automatically generated

Here, we run the dataloader for train and validation dataset. We initialize the model. We define the Adam optimizer and criterion function as Cross Entropy loss. We also initialize the dictionary to include the train and val loss.

Graphical user interface, text, application

Description automatically generated

We Start the training of our network. We load the train and validation loss after each epoch. We do Backward propagation for loss. We make sure that for validation, model is kept in evaluation mode and set torch,no\_grad.

1. Prediction

Chart

Description automatically generated with medium confidence

We forward pass an input through our trained network and display the output. We take argmax on axis contain all classes which gives best label for the respective pixel. We convert this to numpy and display the output through matplotlib.plt.imshow function.

1. Color Mapping

Chart

Description automatically generated

We color map each pixel with the help of defined dictionary earlier. Each pixel represents one class and accordingly color mapping is done. For this I created (w,h,3) array containing zeros. And then iterate over each pixel to color map. At end we change the datatype of this array to uint8 in order to display correctly through plt.imshow

1. One-Hot encoding of ytrue and ypred

Text

Description automatically generated

I create 2 arrays of shape matching to ytrue and ypred. Then for the provided class, I compare each pixel to the given class and if True I’ll change the pixel value of the ypred\_hot and ytrue\_hot to 1 respectively.

1. Evaluation Metric (Intersection over Union)

Text

Description automatically generated

Here, I iterate over each class and keep model in evaluation mode. Then I’ll convert the ytrue and ypred into on-hot encoded labels. Then Intersection is equal to the common 1’s in both the arrays. That is computed by intersection=np.sum(np.abs(ypred\_hot\*ytrue\_hot),axis=(0,1))

Mask\_sum is the total ‘1’ in both array (including common 1’s). Hence union is computed by

union = mask\_sum - intersection (to remove the duplicate of common 1’s).

This is calculated for each test image and finally we have one value of IoU for each class given by

score=(intersection\_class+smooth)/(union\_class+smooth).

Here, I provide smoothing=0.005 in order to avoid nan values when a particular class in not present in any test image.

Code reference: <https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html>

1. Plotting bar chart for IoU values

Chart

Description automatically generated

Here, I plot a bar chart for IoU values for each class. Note: here please consider IoU=1 as ‘N/A’ these are the classes that were not present in any test images. They are ‘1’ due to smoothness parameter. Also, few of the classes have IoU very low and hence cannot be seen in the bar chart. I have presented all those values in the later part of this report.

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Training Loss** | **Validation Loss** |
| **1** | **1.3115** | **1.2198** |
| **2** | **1.0315** | **1.509** |
| **3** | **0.9084** | **1.0056** |
| **4** | **0.8042** | **1.0210** |
| **5** | **0.7479** | **0.8942** |
| **6** | **0.7018** | **1.0156** |
| **7** | **0.6481** | **0.7914** |
| **8** | **0.6181** | **0.7717** |
| **9** | **0.5757** | **0.8414** |
| **10** | **0.5883** | **0.9546** |
| **11** | **0.5483** | **0.8399** |
| **12** | **0.4992** | **0.7460** |
| **13** | **0.4779** | **0.7603** |
| **14** | **0.4746** | **0.7725** |
| **15** | **0.4720** | **0.7558** |
| **16** | **0.4426** | **0.7623** |
| **17** | **0.4319** | **0.7075** |
| **18** | **0.4209** | **0.7409** |
| **19** | **0.4137** | **0.7331** |
| **20** | **0.4045** | **0.8215** |

1. **Evolution of loss function with multiple steps.**

Chart, line chart

Description automatically generated

Figure 1: Train/Validation Loss\_FCN32

Chart, line chart

Description automatically generated

Figure 2: Train\Validation Loss: FCN16

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Training Loss** | **Validation Loss** |
| **1** | **1.1172** | **1.0747** |
| **2** | **0.7545** | **0.8471** |
| **3** | **0.6016** | **0.7293** |
| **4** | **0.5235** | **0.8314** |
| **5** | **0.4876** | **0.8042** |
| **6** | **0.4331** | **0.7126** |
| **7** | **0.3862** | **0.6207** |
| **8** | **0.3594** | **0.6130** |
| **9** | **0.2802** | **0.6541** |
| **10** | **0.2443** | **0.5291** |
| **11** | **0.2257** | **0.6113** |
| **12** | **0.2502** | **0.6484** |
| **13** | **0.2388** | **0.5536** |
| **14** | **0.2207** | **0.5699** |
| **15** | **0.2085** | **0.5940** |
| **16** | **0.1950** | **0.5723** |
| **17** | **0.1684** | **0.5364** |
| **18** | **0.1482** | **0.5846** |
| **19** | **0.1377** | **0.5444** |
| **20** | **0.1308** | **0.5736** |

1. **A summary and discussion of the results, including the effects of parameter choices. Compare the 2 versions of modified FCN (32s and 16s). Include the visualization of results; show some examples of successful and some failure examples.**

**Ouput prediction of both Networks compared with GT\_mask**

|  |  |  |
| --- | --- | --- |
| **GT mask** | **FCN32 Output** | **FCN16 Output** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Color mapping of the output prediction.**

|  |  |  |
| --- | --- | --- |
| **Input image** | **FCN32** | **FCN16** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**FCN16: IoU SCORES**

Chart, bar chart

Description automatically generated

Figure 3: IoU\_FCN16

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **IoU score** | 2.2e-06 | N/A | N/A | N/A | 0.23 | 0.05 | 0.099 | 0.83 | 0.37 | 0.04 | 0.71 | 0.66 | 0.35 | 0.14 | 0.44 | 0.09 | 3.3e-09 | 0.33 | 9.6e-06 | 0.41 | 0.22 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **21** | **22** | **23** | **25** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** |
| **IoU score** | 0.79 | 0.78 | 0.93 | 0.08 | 8.7e-07 | 0.75 | 0.21 | 2.5e-08 | 2.6e-07 | 6.5e-07 | 1.1e-07 | 2.7e-07 | 0.07 |

**Mean IoU score = 0.2769**

**FCN32: IoU SCORES**

Chart, timeline

Description automatically generated

Figure 4: IoU\_FCN32

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **IoU score** | 2.4e-06 | N/A | N/A | N/A | 0.13 | 1.5e-07 | 0.08 | 0.81 | 0.29 | 2.3e-08 | 0.31 | 0.55 | 0.46 | 0.11 | 0.20 | 4.6e-07 | 3.3e-09 | 0.02 | 9.6e-06 | 0.002 | 2.4e-08 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **21** | **22** | **23** | **25** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** |
| **IoU score** | 0.71 | 0.59 | 0.81 | 7.9e-08 | 1.6e-06 | 0.53 | 0.005 | 2.5e-08 | 4.2e-07 | 7.0e-07 | 0.01 | 2.7e-07 | 4.0e-07 |

**Mean IoU score= 0.1821**

**Observations:**

1. **FCN16 performs far better than FCN32 when the visualizations are compared.**
2. **The networks were trained on CPU as I kept batchsize=1. Due to which using GPU was ineffective due to large number of data transfer from CPU to GPU.**
3. **FCN32 could be trained with more number of epochs and might get better results.**
4. **FCN16 segments more coarse details of the image due to upscoring from previous convolutional block.**
5. **Using trained ResNet network, helped in training as we got good weights values for initializing.**
6. **IoU score is best for class 23: sky for both networks. ‘Road’ is second best classified and segmented for both networks.**
7. **Few classes were finely classified with FCN16 which was not classified more often in case of FCN32. Eg. Class 27 (truck)**
8. **Mean Intersection over Union value is more for FCN16 compared to FCN32.**
9. **Cross Entropy loss function calculates loss over multi dimension, in our case (34, number of classes) and hence was good consideration of loss function.**
10. **Since, I used only 20 epochs, I did not lower the learning rate after few epochs. I continued with initial learning rate of 0.001.**