CS 498: Assignment 4: Segmentation

Priyam Mazumdar, priyamm2

March 25, 2022

Submission

In this assignment, you will implement semantic segmentation using neural networks. The starter code consists of an iPython notebook "mp4.ipynb" which can be opened and run on Google colab. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: JBXJVZ). We recommend you add your answers to the latex template files we provided. More details on what to report are in the provided notebook.

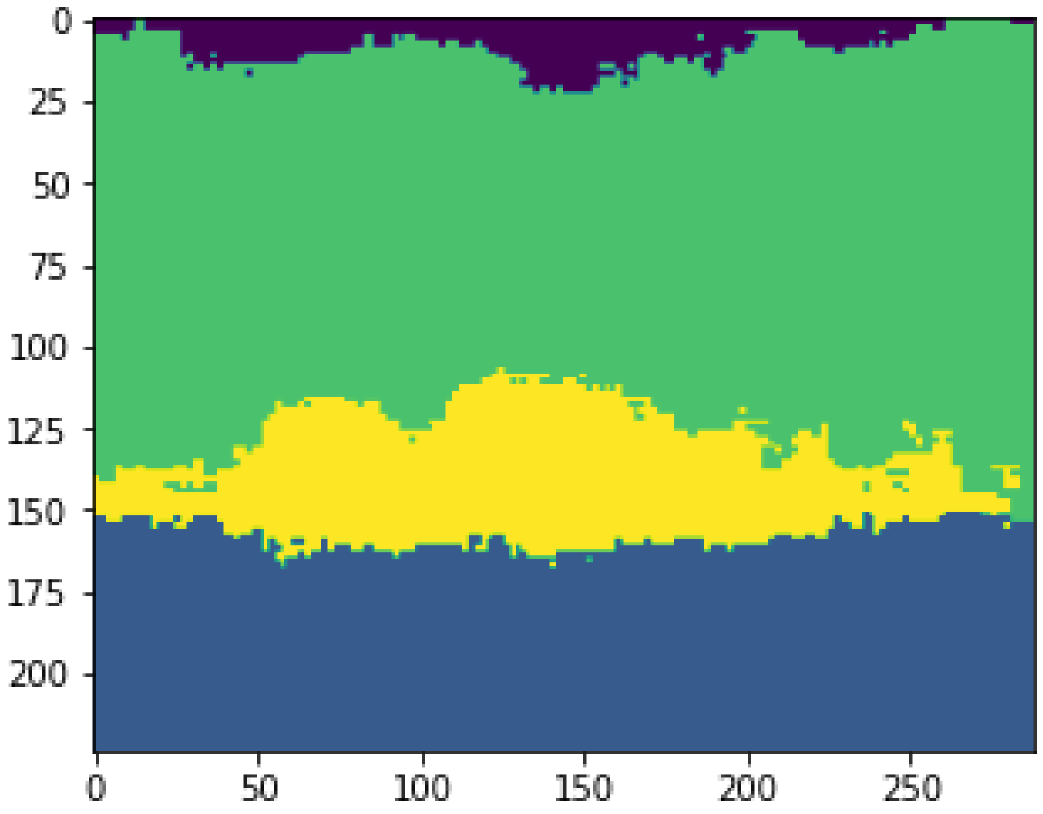
Reminder: please put your name and netid in your pdf. Your submission should include your pdf and filled out mp4.ipynb.

Semantic Segmentation

Question 1 (Data loading and augmentation)[1 pt]: We provide code that loads the segmentation data. In this part you will need to perform data augmentation on the loaded data within the "SegmentationDataset" class. In particular you should take a random crop of the image and with some probability you should flip the image horizontally. You should experiment with different probabilities and crop sizes and report the results in your pdf. Make sure to use pytorch built in transforms methods.

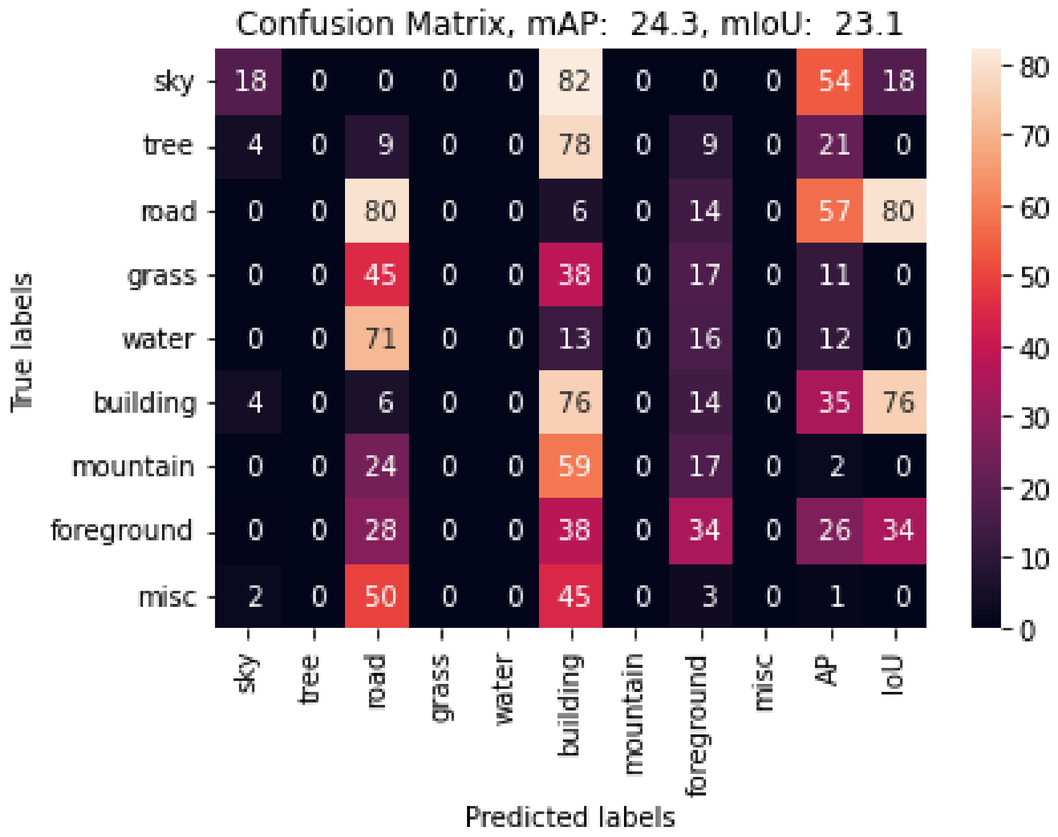
Question 2 (Simple Baseline) [2 pts]: In this part you will be modifying "simple_train" and "simple_predict". For each pixel you should compute the distribution of class labels at that pixel from the training dataset. When predicting classes for a new image, simply output these class frequencies at each pixel. You can run the evaluation code (from the next question) with your simple baseline and see its mean average precision which we provide - it should be around 24.

Question 2 Answer: After building our simple train (number of occurrences of each class label at each pixel) and then simple predict (the probability of each class label at each pixel) we are able to plot both a representation of the predicted output.



Question 3 (Evaluation Metrics) [1 pts]: We must evaluate the quality of our predictions. In this part you will fill in "compute_confusion_matrix". You should write code to compute the confusion matrix as well as IoU for the predicted segmentation when compared to ground truth. We provide code for visualizing the computed values as well as computing mean average precision.

Question 3 Answer: We see that our Mean Average Precision is around 24.3 as expected. We see pretty good classification ability of roads and buildings, but this method poorly evaluates other classes.



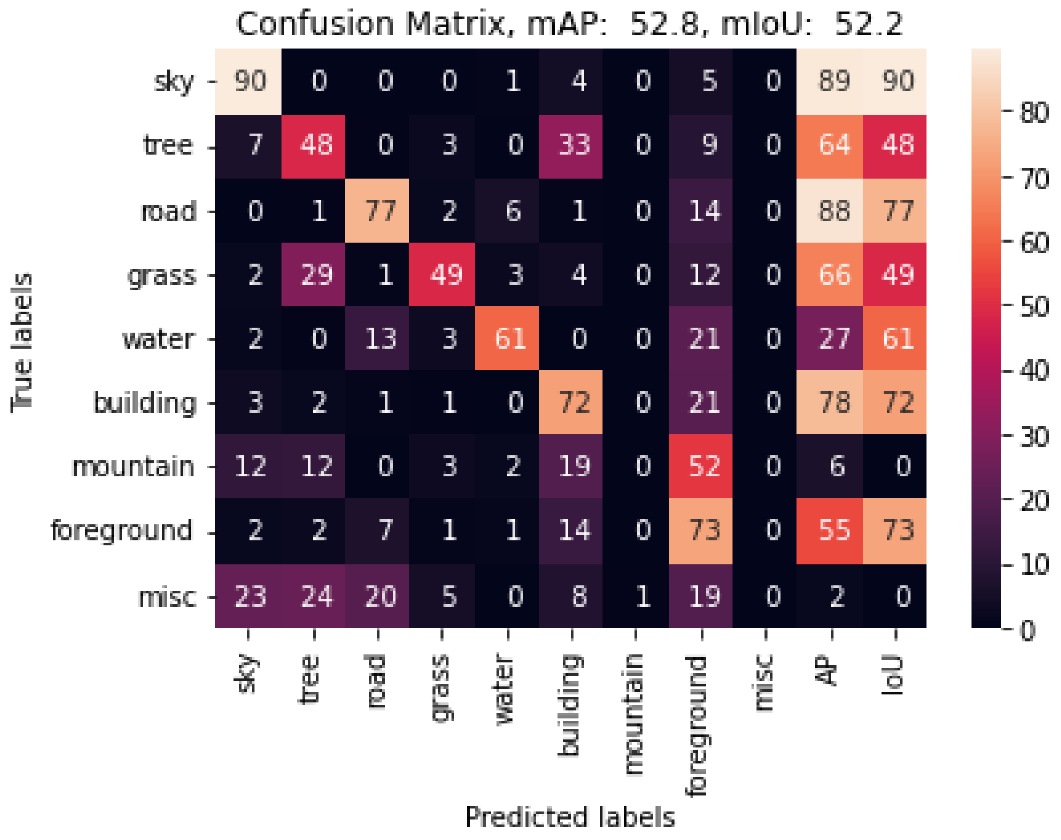
	sky	tree	road	grass	water	building	mountain	foreground	misc	AP	IoU
sky	18	0	0	0	0	82	0	0	0	54	18
tree	4	0	9	0	0	78	0	9	0	21	0
road	0	0	80	0	0	6	0	14	0	57	80
grass	0	0	45	0	0	38	0	17	0	11	0
water	0	0	71	0	0	13	0	16	0	12	0
building	4	0	6	0	0	76	0	14	0	35	76
mountain	0	0	24	0	0	59	0	17	0	2	0
foreground	0	0	28	0	0	38	0	34	0	26	34
misc	2	0	50	0	0	45	0	3	0	1	0

Question 4 (Loss function) [2 pt]: To train a model we need a loss function. In this part you will fill in "cross_entropy_criterion" with your implementation of the weighted cross entropy between predicted class probabilities and ground truth class labels.

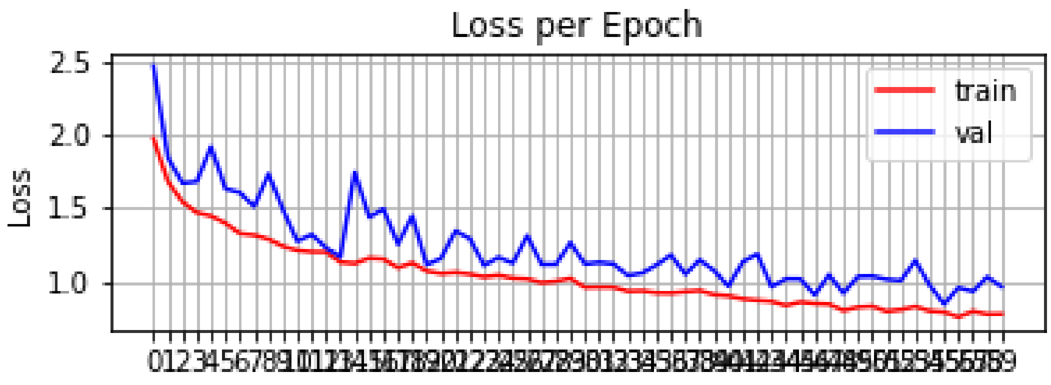
Question 5 (Train loop) [2 pt]: In this part you will implement the stochastic gradient descent training loop in pytorch, modifying "train". We provide code to validate a trained model and a skeleton for training one.

Question 6 (Model definition and training) [4 pt]: Implement a basic convolutional neural network, as well as the U-Net architecture for semantic segmentation. Train your models with the code you wrote for Question 5.

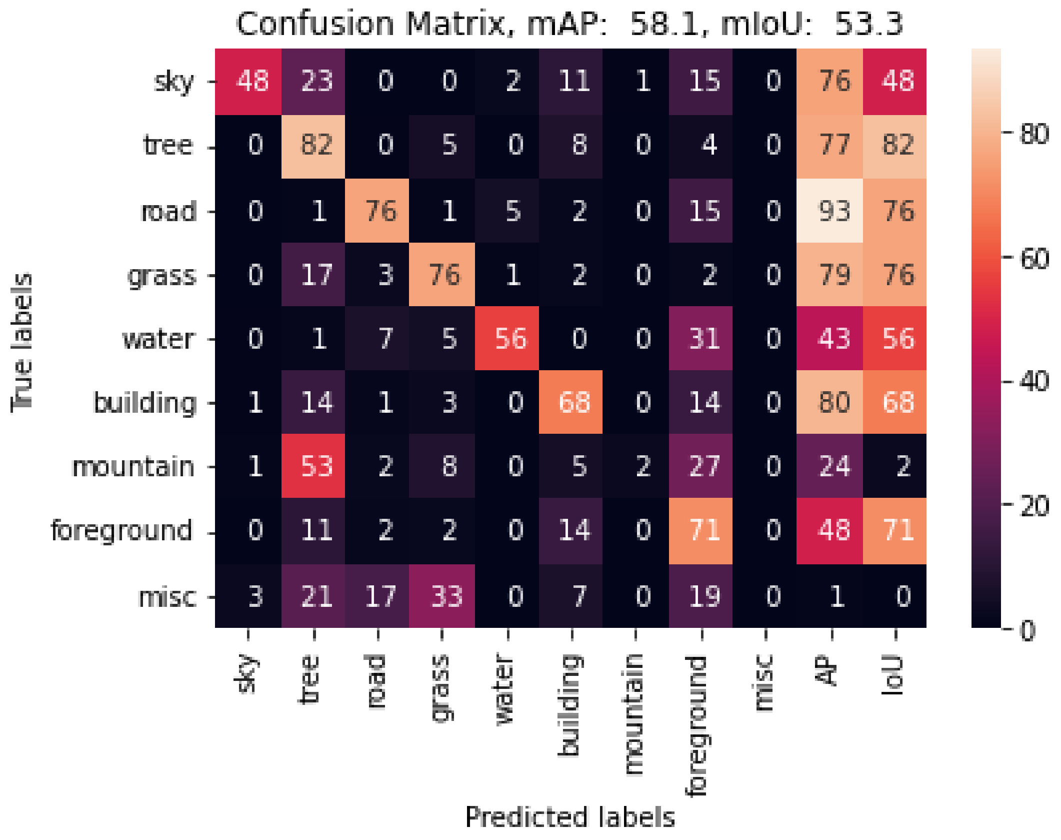
Question 6 Answer: My implementation of the base convolutional network was a UNet without any skip connections. I was curious about the amount of added predictive ability that they would add. My results gave me a validation error of around 0.8, trained with the AdamW optimizer (0.01 weight regularization) and a mAP of around 52.8 and mIoU of 52.2 which is already much higher than our simple predict. Here are the outputs of our confusion matrix and training loss:



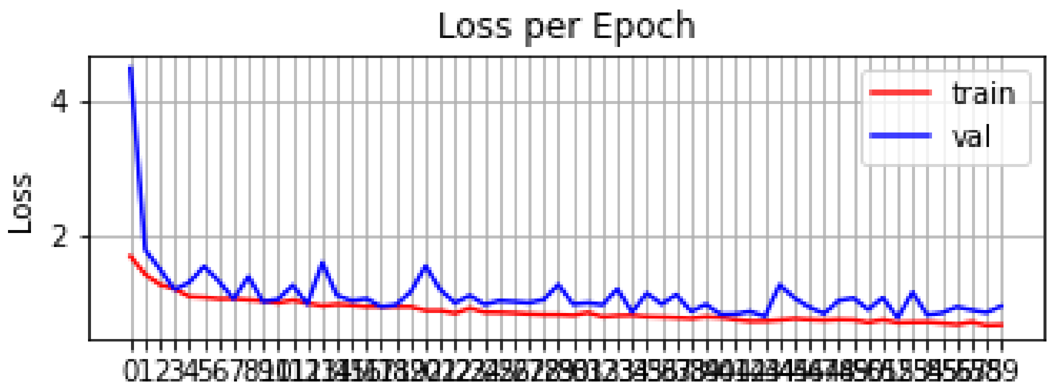
	sky	tree	road	grass	water	building	mountain	foreground	misc	AP	IoU
sky	90	0	0	0	1	4	0	5	0	89	90
tree	7	48	0	3	0	33	0	9	0	64	48
road	0	1	77	2	6	1	0	14	0	88	77
grass	2	29	1	49	3	4	0	12	0	66	49
water	2	0	13	3	61	0	0	21	0	27	61
building	3	2	1	1	0	72	0	21	0	78	72
mountain	12	12	0	3	2	19	0	52	0	6	0
foreground	2	2	7	1	1	14	0	73	0	55	73
misc	23	24	20	5	0	8	1	19	0	2	0



Afterwards, I added in the skip connections from the original UNet implementation that rendered an even higher mAP of 58.1 , mIoU of 53.3, and a validation loss around 0.7. Here are the outputs from UNet:



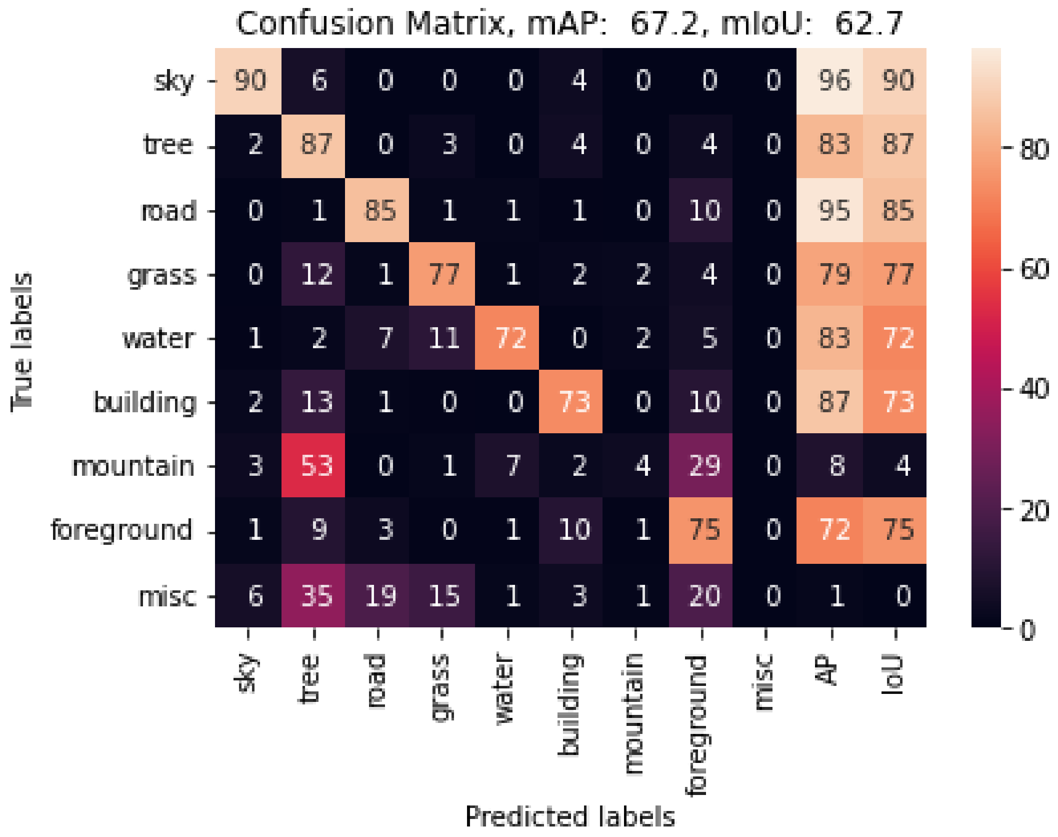
	sky	tree	road	grass	water	building	mountain	foreground	misc	AP	IoU
sky	48	23	0	0	2	11	1	15	0	76	48
tree	0	82	0	5	0	8	0	4	0	77	82
road	0	1	76	1	5	2	0	15	0	93	76
grass	0	17	3	76	1	2	0	2	0	79	76
water	0	1	7	5	56	0	0	31	0	43	56
building	1	14	1	3	0	68	0	14	0	80	68
mountain	1	53	2	8	0	5	2	27	0	24	2
foreground	0	11	2	2	0	14	0	71	0	48	71
misc	3	21	17	33	0	7	0	19	0	1	0



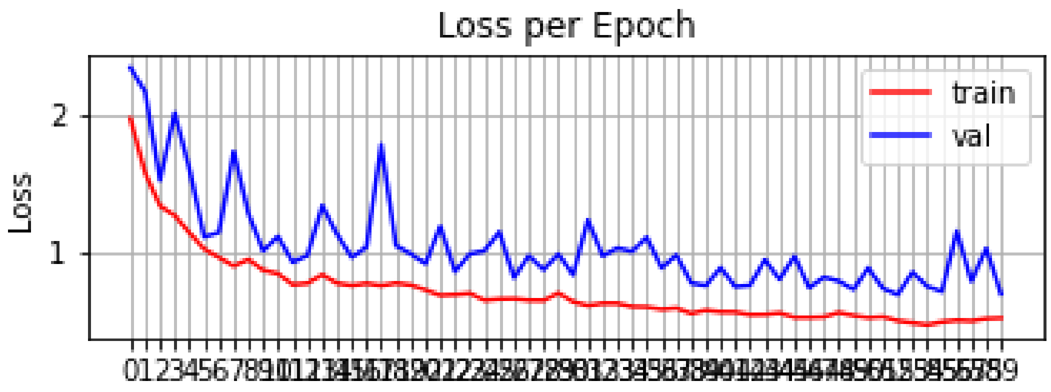
This makes sense as the purpose of the skip connection is similar to the reason why we have them in ResNets: Vanishing gradients. In our UNet implementation there are a lot of convolutional and transpose convolutional layers, and during back propagation we would lose our gradient values at the start of our network. The addition of skip connections allow for another path of gradient updates to take place so we can effectively use our loss to toggle weights of our entire model.

Question 7 (Use Pretrained Model) [3 pt]: In this part you will build on resnet-18 (note there are multiple ways to do this). Report your results, they should be better than the best you got using UNet training from scratch.

Question 7 Answer: It is almost always better to start with a pretrained model that has seen data similar to what we have. The reason for this is we can start at an already good predictive ability and continue to finetune the model to best predict our own data. In my implementation, I removed the last two layers of the pretrained ResNet18 (this removed the linear layer to map to the 1000 classes of imagenet, as well as the averagepool2d that precedes it. The remaining part of the model was treated as the encoder of our UNet, and I built a decoder to take the resnet output and map it back to the target image size via transpose convolutions and our convolution blocks. After training this model, which was trained end to end as I believed locking some layers out from gradient updates wouldnt allow me to fully leverage resnet18 for my usecase, I got the highest mAP of 67.2 and mIoU of 62.7 wtih a validation loss around 0.6. This shows that starting from a pretrained model can more easily and immediately return some of the best performance. Here is the output of my confusion matrix and training loss:



True labels	sky	tree	road	grass	water	building	mountain	foreground	misc	AP	IoU
sky	90	6	0	0	0	4	0	0	0	96	90
tree	2	87	0	3	0	4	0	4	0	83	87
road	0	1	85	1	1	1	0	10	0	95	85
grass	0	12	1	77	1	2	2	4	0	79	77
water	1	2	7	11	72	0	2	5	0	83	72
building	2	13	1	0	0	73	0	10	0	87	73
mountain	3	53	0	1	7	2	4	29	0	8	4
foreground	1	9	3	0	1	10	1	75	0	72	75
misc	6	35	19	15	1	3	1	20	0	1	0



Because of the best performance of the Resnet-UNet model, I wanted to also share some of the outputs of my model to compare the predicted segmentation and ground truth: