CS 498: Assignment 4: Segmentation

Due on April 8, 2022

March 25, 2022

Submission

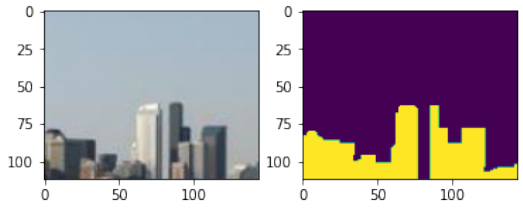
In this assignment, you will implement semantic segmentation using neural networks. The starter code consists of an iPython notebook "mp4.ipynb" which can be opened and run on Google colab. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course 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.

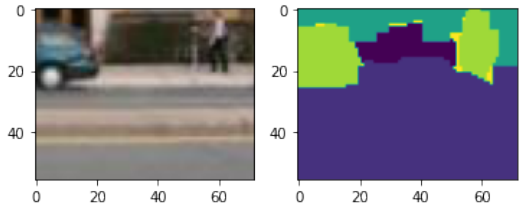
Semantic Segmentation

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.

Q1 Answer Results:



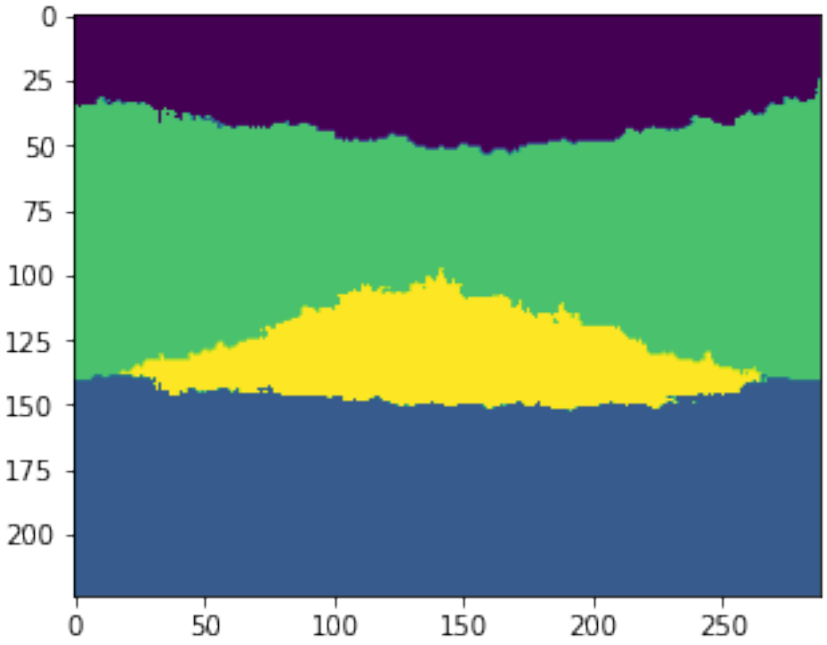
data augmentation with $\frac{1}{4}$ crop size and 0.5 flip rate



data augmentation with $\frac{1}{16}$ crop size and 0.25 flip rate

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.

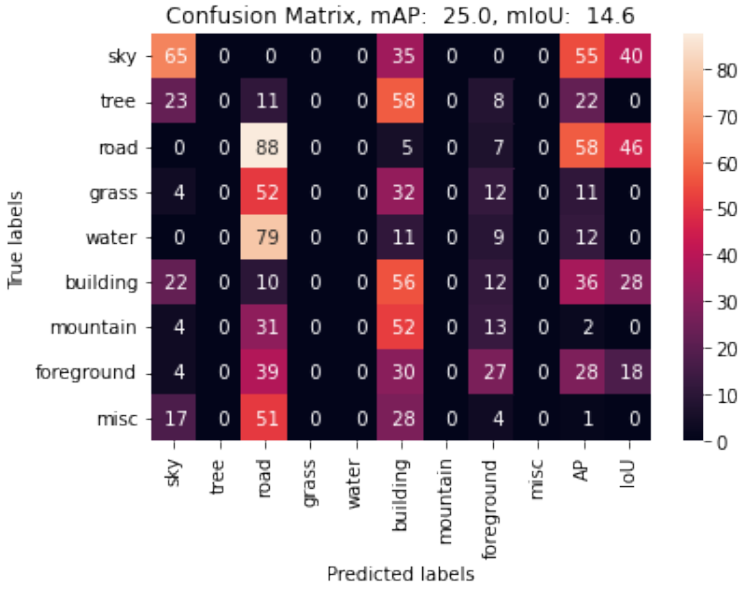
Q2 Answer Results:



Simple Predict segmentation result

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.

Q3 Answer Results:



Simple Predict result

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.

Q4 Answer Multi-class cross entropy loss from CS 446 Machine Learning:

Cross-entropy loss (multi-class logistic loss) via MLE

Conditional model: given predictor $f : \mathbb{R}^d \to \mathbb{R}^k$, model $p_f(\hat{y} = \cdot|\boldsymbol{x}) \propto \exp(f(\boldsymbol{x}))$, which means $p_f(\hat{y} = j|\boldsymbol{x}) = \frac{\exp(f(\boldsymbol{x})_j)}{\sum_{i=1}^k \exp(f(\boldsymbol{x})_i)}$.

Corresponding **cross-entropy loss** ℓ_{ce}: given true label $y \in \{1, \ldots, k\}$,

$$\begin{aligned}
\ln \frac{1}{p_f(\widehat{y} = y|\boldsymbol{x})} &= \ln \frac{\sum_{j=1}^k \exp(f(\boldsymbol{x})_j)}{\exp(f(\boldsymbol{x}))_y} \\
&= -f(\boldsymbol{x})_y + \ln \sum_{j=1}^k \exp\left(f(\boldsymbol{x})_j\right) \\
&=: \ell_{\mathsf{ce}}(f(\boldsymbol{x}), y).
\end{aligned}$$

Notes.
In `pytorch`, this is `torch.nn.CrossEntropyLoss()(f(x), y)`.
Loss is minimized when $p_f(\widehat{y} = y|\boldsymbol{x}) \approx 1$.
Name comes from the cross-entropy expression $\sum_{j=1}^k (\boldsymbol{e}_y)_j \ln \frac{1}{p_f(\hat{y}=j|\boldsymbol{x})}$.

CS 446 Lecture 4 Slide

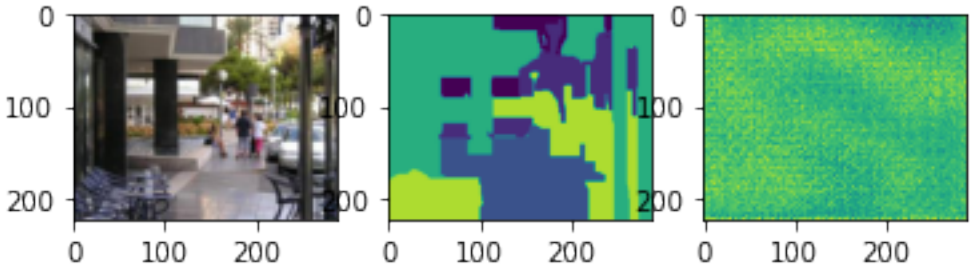
Question 5 (Train loop) [2 pt]: In this part you will implement the stochastic gradient descent training loop in pytorch, modifying "train". We provide code to validate a trained model and a skeleton for training one.

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.

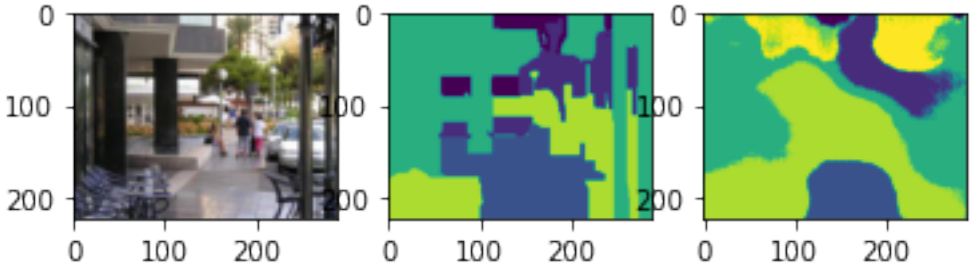
Question 7 (Use Pretrained Model) [3 pt]: In this part you will build on resnet-18 (note there are multiple ways to do this). Report your results, they should be better than the best you got using UNet training from scratch.

Q5-7 Answer The following neural networks are all trained with data augmentation $\frac{1}{4}$ crop size and 0.5 flip rate. Training Process:

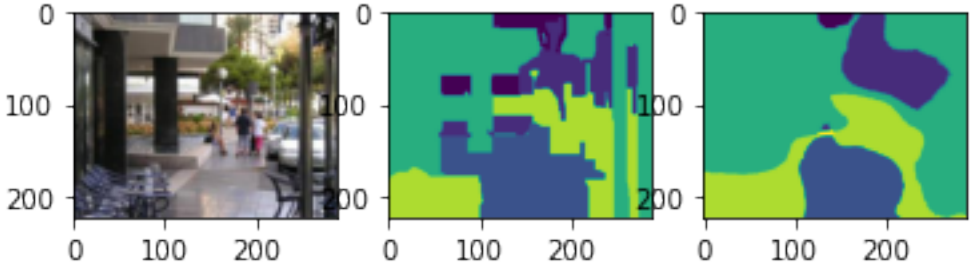
1. Basic Convolutional Neural Network: (from left to right: original image from validation dataset, ground truth label, prediction result)



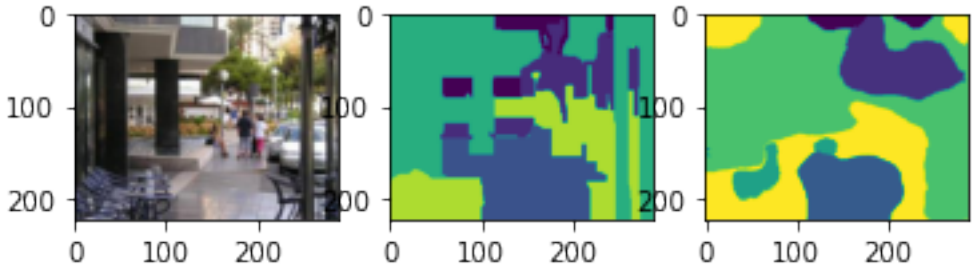
Epoch 1, training loss 20.260, validation loss 21.841



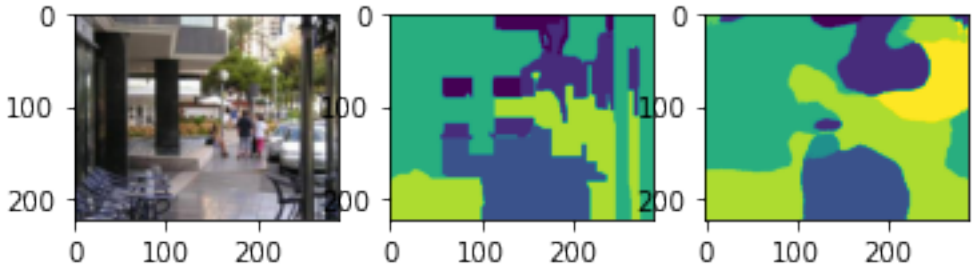
Epoch 16, training loss 14.008, validation loss 17.215



Epoch 32, training loss 11.302, validation loss 16.087

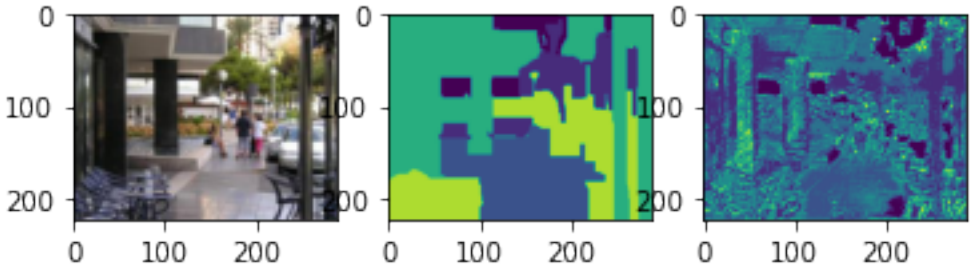


Epoch 64, training loss 8.864, validation loss 14.817

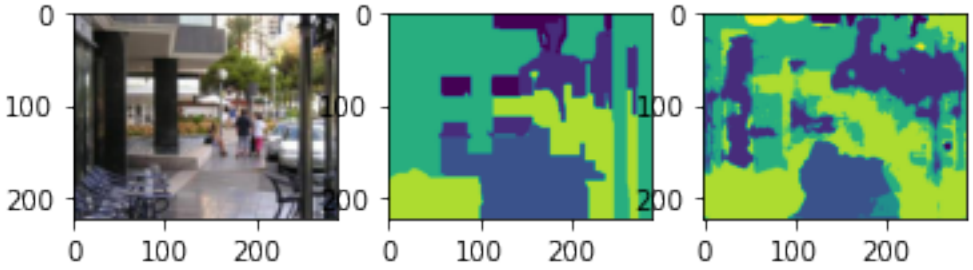


Epoch 128, training loss 7.700, validation loss 15.685

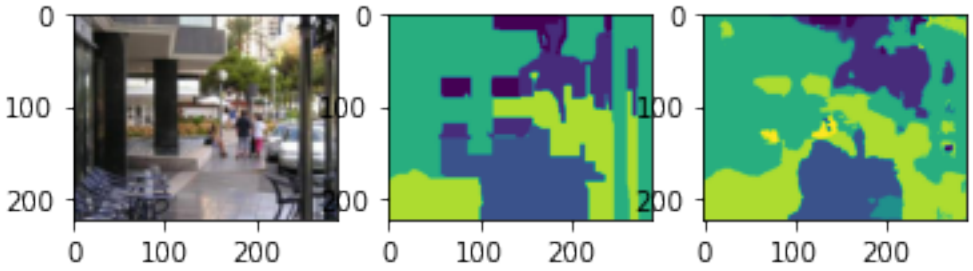
2. UNet: (from left to right: original image from validation dataset, ground truth label, prediction result)



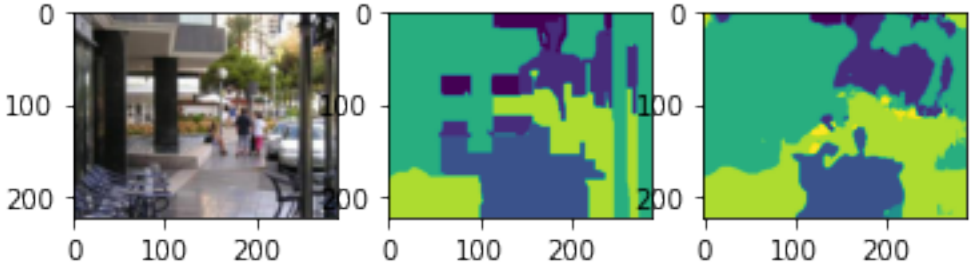
Epoch 1, training loss 18.419, validation loss 20.415



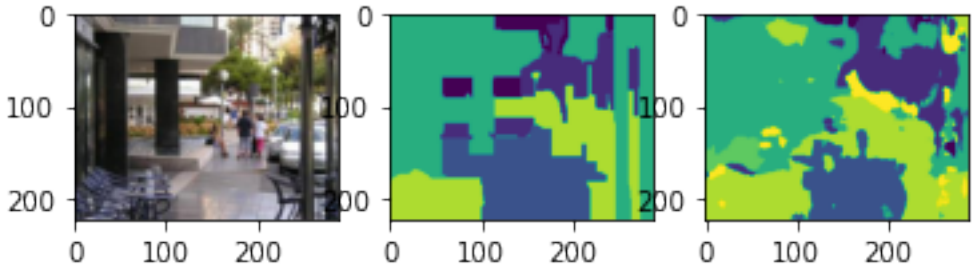
Epoch 16, training loss 11.648, validation loss 16.831



Epoch 32, training loss 10.526, validation loss 16.415



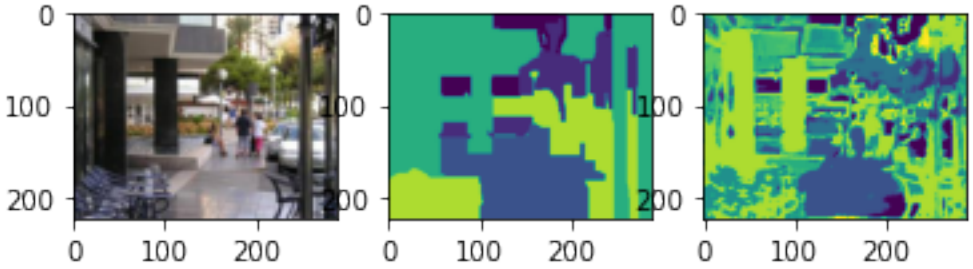
Epoch 64, training loss 8.042, validation loss 15.625



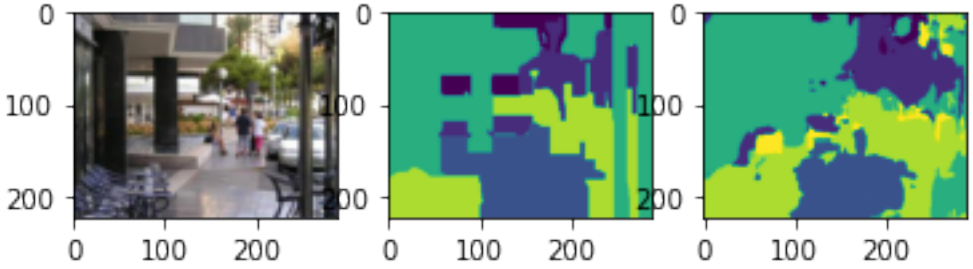
Epoch 128, training loss 6.353, validation loss 17.381

3. ResUNet: (from left to right: original image from validation dataset, ground truth label, prediction result)

   (a) Freeze ResNet parameters:

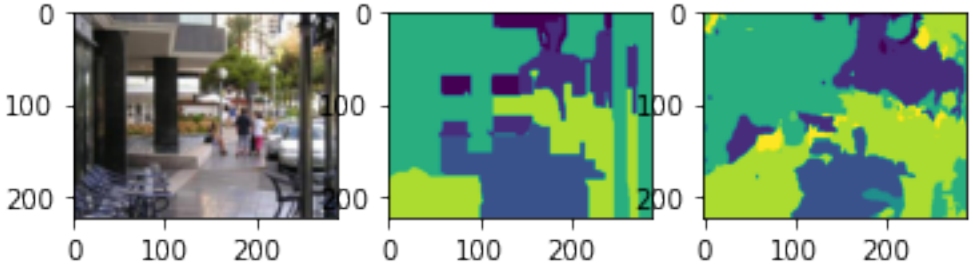


Epoch 1, training loss 17.826, validation loss 19.511

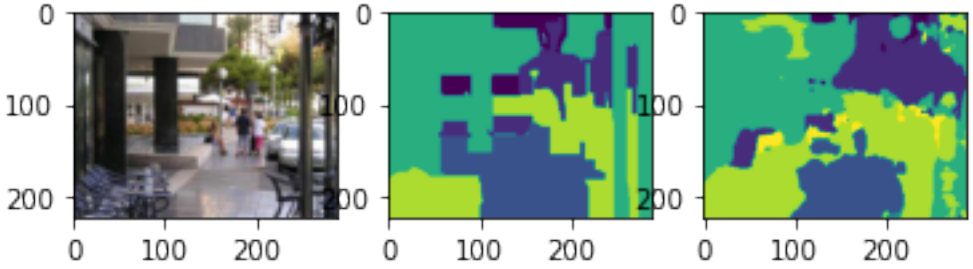


Epoch 64, training loss 7.980, validation loss 15.390

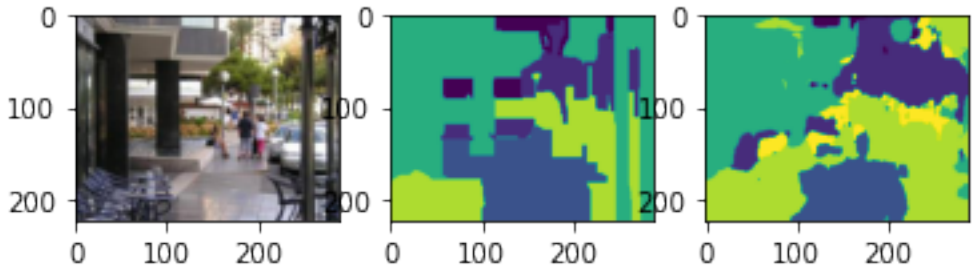
   (b) Then unfreeze ResNet parameter:



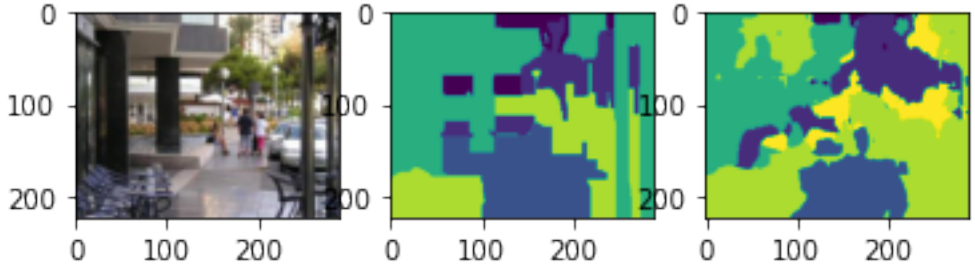
Epoch 1, training loss 7.177, validation loss 14.920



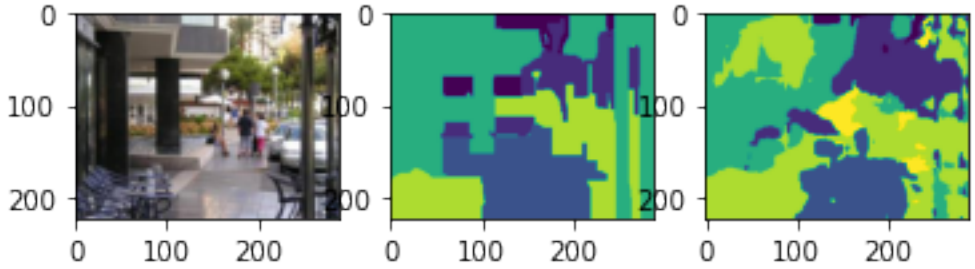
Epoch 16, training loss 5.691, validation loss 13.209



Epoch 32, training loss 5.381, validation loss 12.803

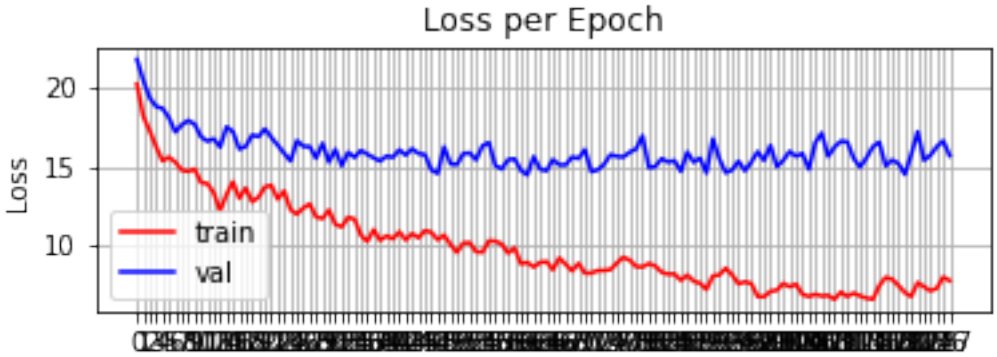


Epoch 64, training loss 4.287, validation loss 13.304

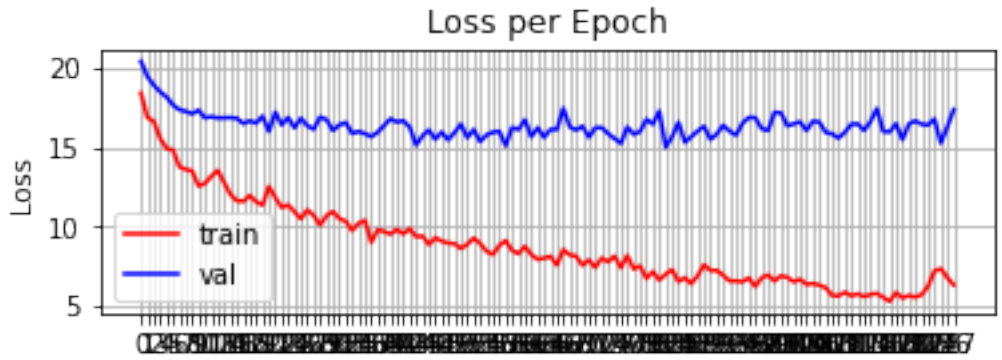


Epoch 128, training loss 3.423, validation loss 15.460

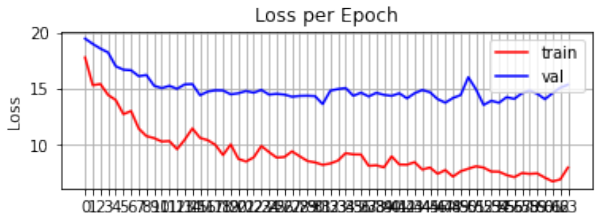
Loss plot cross compare:



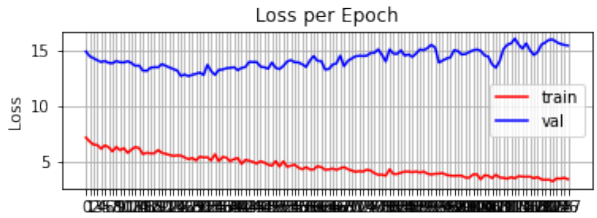
Basic Convolutional Neural Network, 128 epochs, Adams, lr = 0.0001



UNet, 128 epochs, Adams, lr = 0.0001

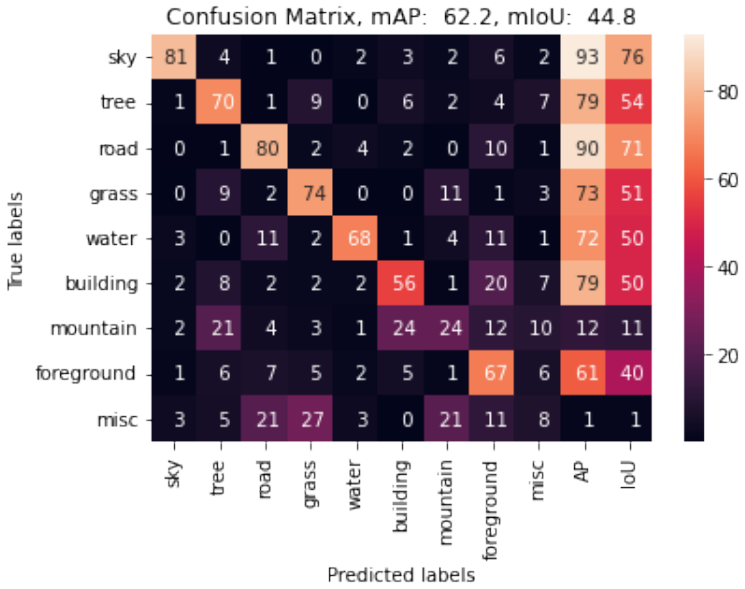


ResUNet freeze ResNet parameters
64 epochs, Adams, lr = 0.001

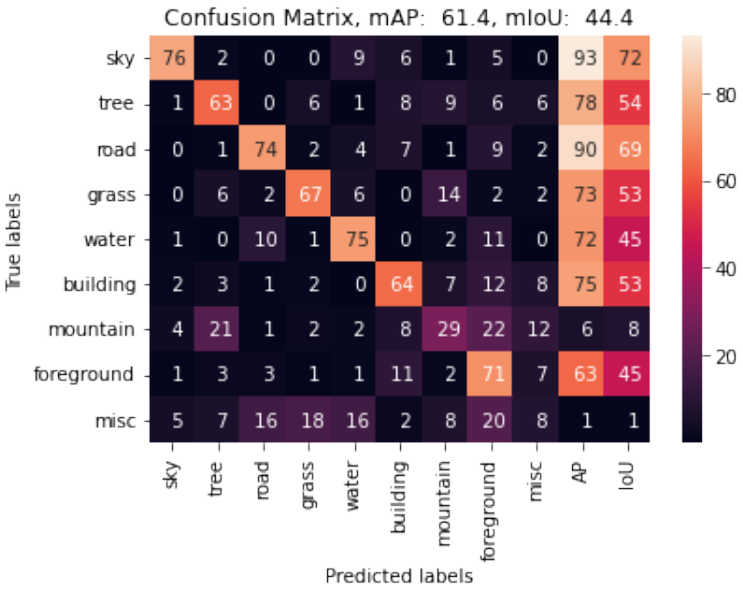


ResUNet unfreeze ResNet parameters
128 epochs, Adams, lr = 0.0001

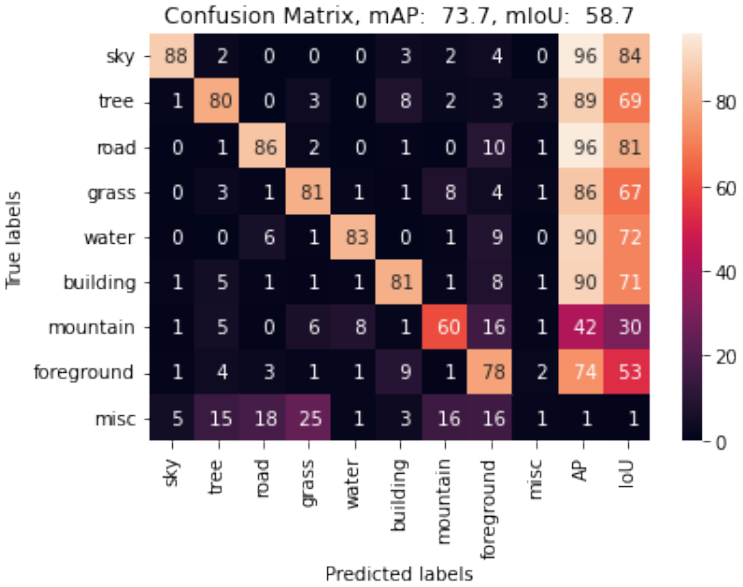
Validation result cross compare:



Basic Convolutional Neural Network, 128 epochs, Adams, lr = 0.0001

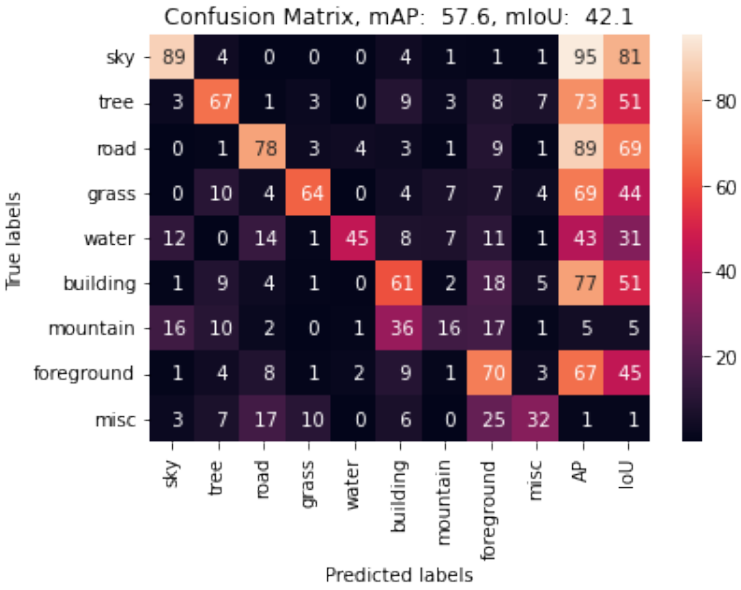


UNet, 128 epochs, Adams, lr = 0.0001

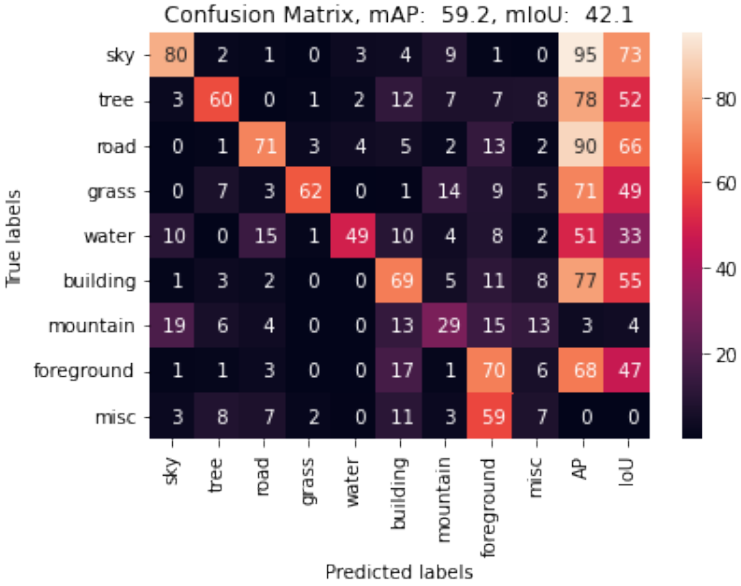


ResUNet, 64 + 128 epochs, Adams, lr = 0.001/0.0001

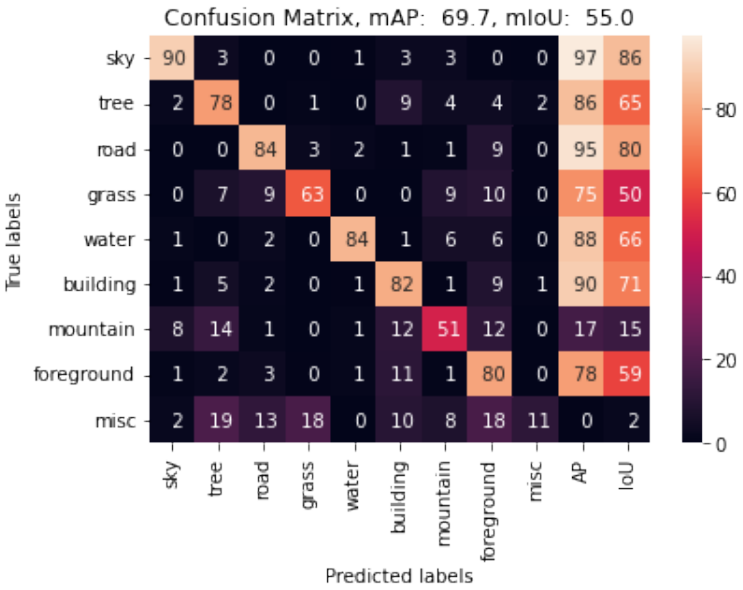
Test result cross compare:



Basic Convolutional Neural Network, 128 epochs, Adams, lr = 0.0001

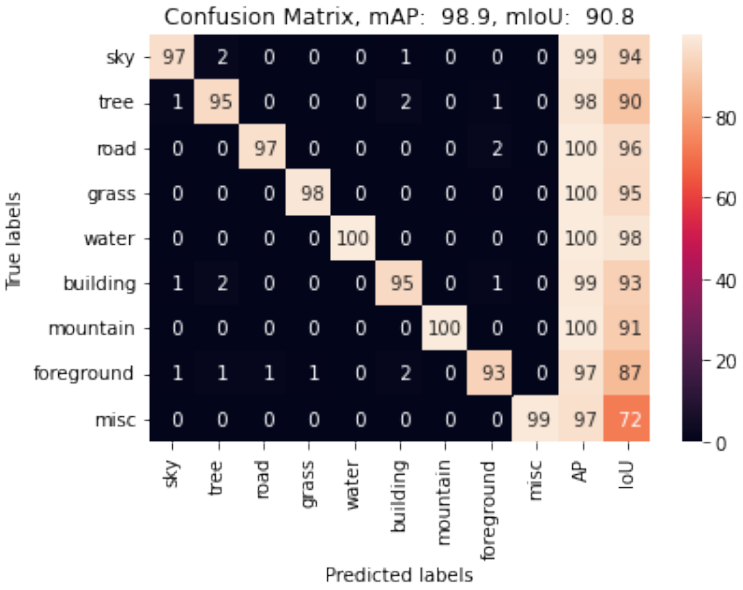


UNet, 128 epochs, Adams, lr = 0.0001

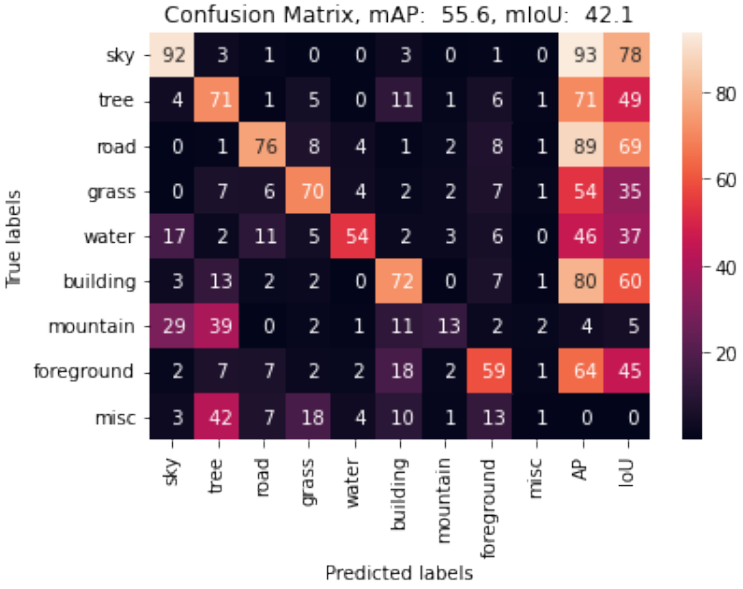


ResUNet, 64 + 128 epochs, Adams, lr = 0.001/0.0001

Overfit test, ResUNet trained without data augmentation or batch normalization layers, 64 + 128 epochs, Adams, lr = 0.001/0.0001:



validation dataset



test dataset

Conclusion:

Overall, UNet is a pretty powerful architecture, although it's easy to get overfit, typically after 32-64 epochs. It's ability to overfit proves the capability.

Data augmentation increases the robustness of neural networks, but needs to take care of the crop rate, since output shape needs to match input shape, and resize is not differentiable.

Batch Normalization contains learnable parameters, need to change between $.train()$ mode and $.eval()$ mode, and it significantly speeds up the training process. One thing worth mentioning, is that batch normalization should be done before ReLU layer, because it further complicates matter if we first do ReLU then batch norm, which is essentially pushing the 75% quantile to 0.

Transfer learning is also a great method, it gets really clear edges even without any training.

It's better if we first freeze the pretrained image net, then fine tune it together with other parameters, because randomly initialized parameters can harm the parameters of pretrained image net at first, but after a few epochs when it approximates a local minima, we can train all the parameters to march towards the overall local minima.

Instance Segmentation (Bonus 4pt)

Now we have a deep semantic segmentation algorithm. However, the model cannot distinguish each instance. Could you use a similar UNet model to build an instance segmentation algorithm? Please download the Upenn-Fudan pedestrian dataset here `https://www.cis.upenn.edu/~jshi/ped_html/` and get their instance labels. There are two types of instance segmentation methods, detection-free or detection-based. Choose either one of them.

Please refer to the Deep Watershed Transform for an example of detection-free method `https://github.com/min2209/dwt`. Your goal is to build a network with two headers, one to predict the binary semantic label similar to your semantic segmentation network and the other to predict the distance transform to the boundary. Once you have these two, the watershed transform could be applied to recover per-pixel instance labels.

For the detection-based method, please refer to the MaskRCNN (`https://pytorch.org/vision/stable/_modules/torchvision/models/detection/mask_rcnn.html`). You will develop a network that predicts detection proposals first. Then within each detection proposal, a binary foreground and background segmentation is conducted to separate each instance.

For each method, you should implement 1) the loss function the paper uses; 2) implement the data loader and post-processing that converts network output to per-pixel instance label 3) train and evaluation each model (you could either reuse your UNet backbone or follow the original paper).