Udacity Self Driving Car Nanodegree

Term 3 Project 2 – Semantic Segmentation

Rubric Criteria

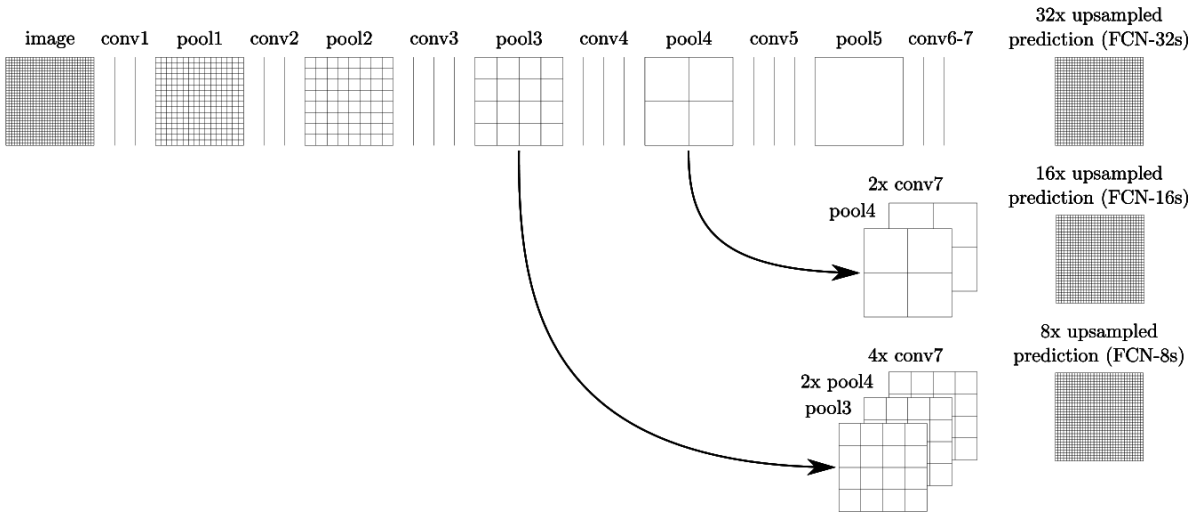
Build the Neural Network

Does the project load the pretrained VGG model (is `load_vgg()` implemented properly)?

The VGG model is loaded in function *load_vgg* at line 38 of main.py. The appropriate decoder layers are exposed and returned in lines 40 to 47 of the same function.

Does the project learn the correct features (is `layers()` implemented properly)?

In the *layers* function (lines 51 to 85 of main.py) I take the exposed decoder layers and create an encoder section according to the structure defined in the original Shelhammer, Long, Darrell paper (see below).



Does the project optimize the neural network (is `optimize()` implemented properly)?

In lines 89 to 104 of main.py the *optimize* function uses softmax cross entropy loss with logits to evaluate the network performance (line 101). To train the neural net I use the Adam Optimizer as suggested in the project walkthrough video (line 102).

Does the project train the neural network (is `train_nn()` implemented properly)?

The network is trained in lines 108 to 134 of main.py. The training proceeds for a user defined number of epochs. Within each epoch the input data is divided into batches with a user defined number of examples in each batch (batch_size). Each batch is trained using *optimize* and the results are printed to stdout.

Neural Network Training

Does the project train the model correctly (loss decreases over time)?

Although the loss for each individual batch bounces around a bit, the overall trend is downward throughout the training process. In the first epoch the loss decreases significantly from about 1.5 down to below 0.4. Subsequent epochs improve the performance but at a lower rate.

Does the project use reasonable hyperparameters?

The hyperparameters used in this project are: learning_rate, keep_probability, number_of_epochs, and batch_size.

Learning_rate was set to a commonly used value of 0.0001. I experimented with different settings for this hyperparameter but, judging by final segmentation performance, I didn't see any benefit to changing from the original value.

I started with a keep_probability of 0.5 but found that a higher value (0.75) helped to mitigate false-positive road identification.

The number_of_epochs used was based on watching the loss value trends. Much of the learning occurs in the first five to eight epochs but I saw slight improvements out to twenty or so epochs. My final value was set to 22 epochs.

For batch_size I settled on eight examples. This seemed to be a sweet spot for segmentation quality. It's been shown that while large batch sizes may fit the training set better but they do not generalize as well as small batch sizes (see: https://arxiv.org/abs/1609.04836).

Does the project correctly label the road?

Examining the results in the runs folder I'm pretty pleased with the results. The road is properly identified in the vast majority of images and the false positives are in areas of the image that have the same color/texture as the roadway.

The false negatives are less common and seem to coincide with confusing shadow patterns or train tracks that understandably make the road look like "not road".

Considering that the network was trained on just 289 images this is a good result. Increasing the number of training images and providing more variation along non-salient axis would likely improve the results further.