Udacity Self-Driving Car Nanodegree

Term 3, Project 2: Semantic Segmentation

Mark Helms, November 27, 2018

Goal/Summary

The goal in this project is to label the pixels in images as being either “road” or “not-road”, using a Fully Convolutional Network (FCN). The FCN is also known as a U-Net, as its shape looks like the letter “U”. The first half of the network is an encoder which compresses and summarizes information in the main picture, and the second half of the network is a decoder which decompresses the compressed information in order to draw a final result that is the same size as the input image (same x and y dimensions, maybe not the same number of channels). Additionally, skip layers should be used. The encoder portion of the FCN will be based off of the pre-trained VGG16 model. All coding will be performed in Tensorflow / Python and tested using a GPU.

Approach

Step 1: Loading the Pre-trained VGG Model (Code: load_vgg)

By loading a pre-trained model, we can perform what is known as Transfer Learning. Transfer learning saves considerable amounts of time and lowers the amount of training data required, if the types of features in your data are similar to those of the data used to train the original network. This has been shown to be the case with images and so it applies very well to this case, where we are analyzing road images.

Once the network is loaded with Tensorflow, certain layers can be extracted. Certain layers are more valuable than others. The layer chosen to end the encoder at (and start the decoder from) should be after all 2D convolutions, but before the fully-connected (aka dense) layers become too small, because too much information is lost. Additionally, since skip layer connections will be used, some layers along the path of 2D convolutions will be extracted as well.

Step 2: Adding on the Decoder Layers (Code: layers)

In order to upsample from the smaller-sized layers, undoing the previous 2D convolutions and restoring the size of the image, a series of 2D transpose convolutions are appended. In order to accommodate the skip layer connections, layers in the decoder with sizes matching those of the encoder layers are added together. In order to do this, we must set the shapes and number of channels in decoder layers to exactly match those of the encoder. These skip layers are valuable, because they provide more information about the original image to the decoder.

After using the skip connections, a couple more convolutions are necessary to restore the size to the original input size. The number of channels in each of these layers should exponentially converge on the final number of channels, in order to allow the network enough flexibility to make the final decisions for labeling.

Step 3: Picking a Loss and Optimizer (Code: optimize)

The loss used in this project is the cross entropy loss, which expresses how far a probability (between 0 and 1) is from the correct value. In this case 0 is not a road and 1 is a road and the probability is the FCN’s guess as to the likelihood of a pixel being one of those labels. The optimizer is a gradient descent optimizer variant called the Adam optimizer, which adjust the learning rate as loss improves in order to avoid dithering between a larger loss value during training.

Step 4: Training the FCN (Code: train_nn)

In order to train the network, the tensorflow training function needs to be fed a dictionary of variables and the training data. The training data includes the images and their true labels.

Training is performed across the same data for multiple epochs. In each epoch, a batch of training data is supplied to the training operation. The batch size should generally be as big as will fit in memory. Also important is that the training data supplied be shuffled every epoch. Udacity has provided a generator which offers this functionality.

As the network is trained, it's possible to retrieve the cross entropy loss for display, in order for the user to visualize the progress the network is making. If the loss never drops or changes wildly, there is likely an error to be addressed. Ideally, the loss should smoothly drop down to a convergence level. The convergence level is controlled by a few things, but mostly the learning rate and the suitability of the network design.

Parameters chosen to train the network were:

Epochs	25
Batch Size	8
Learning Rate	1e-3
Keep Probability	0.75

These values were shown to allow the loss to converge nicely, while avoiding overtraining the network (i.e. having it "memorize" the correct answer for inputs), which is a risk due to the somewhat small size of the training data.

Step 5 Analyzing the Output

Once the network has been trained across multiple epochs and the loss has converged, the code provided by Udacity will run the network with weights fixed and feed it the test images. It takes the output and superimposes the road labeling onto the image using a somewhat opaque layer of green. For a successful run, images should be mostly labeled in green where roads exist and not labeled where there are no roads.

Some example outputs are shown below:

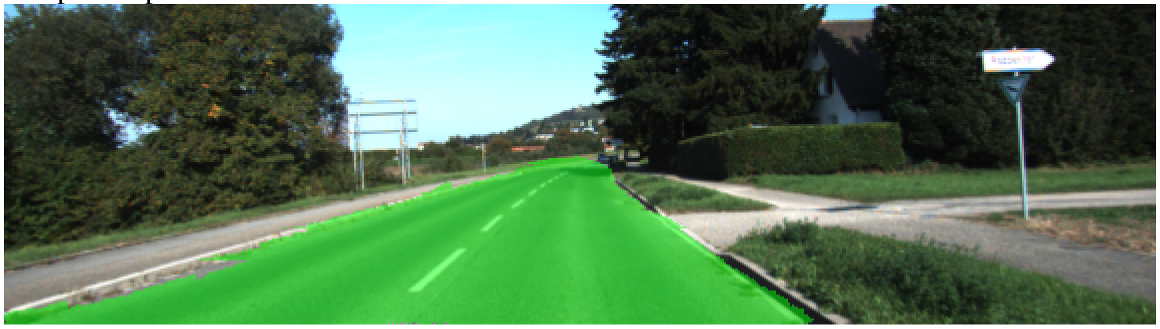

Figure 1. A clear road is easier for the network to classify.

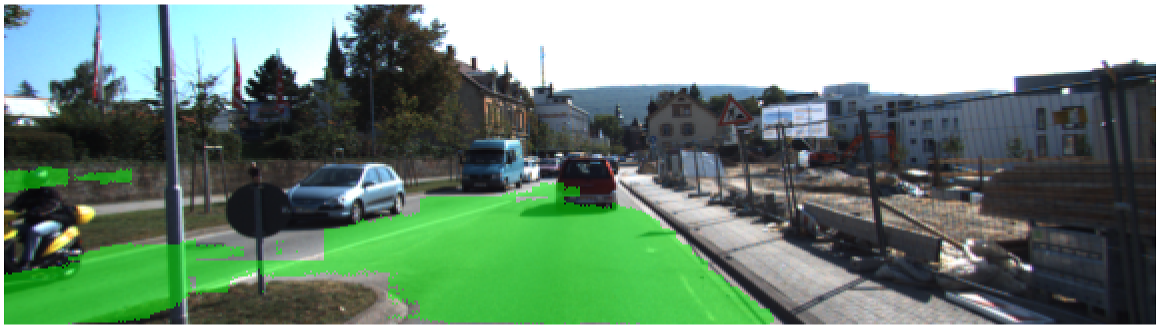

Figure 2. A "busy" road creates more problems.
(The "blockiness" of the solution is a giveaway that an encoder/decoder network is being used.)