Udacity Project: Follow Me

Brad Huang

June 11, 2018

1 Introduction

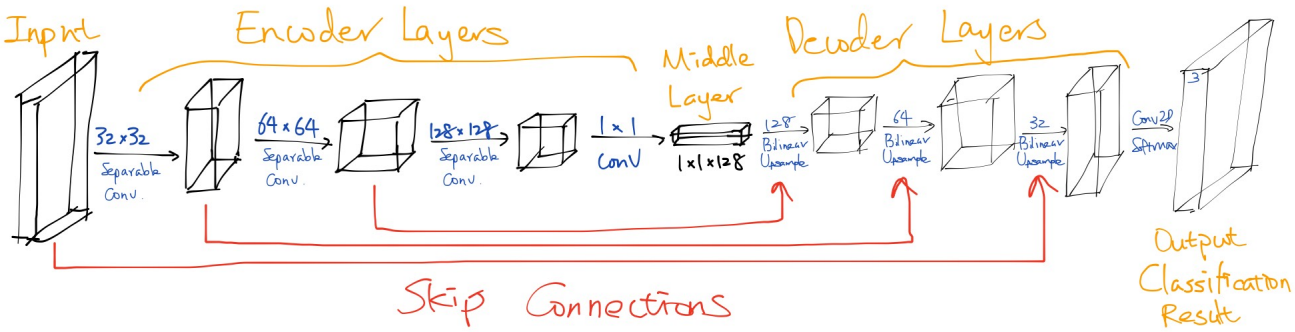
The Follow Me project uses deep learning to perform semantic segmentation in a quadrotor simulation environment that allows a quadcopter to find and follow a certain target in a city scene. The quadcopter's virtual camera would be looking for the target with a red outfit, the "hero", while patrolling around the city. It needs to classify each pixel of the captured images as non-human, random passengers or the target. Our goal is to build and train a neural network that maximizes the Intersection over Union (IOU) metric by comparing the segmentation result and ground truth label images.

2 Neural Network Architecture

After some trial and errors, I built a fully convolutional neural network (FCN) for the task. The dataset comes from Udacity, and the training is done through Jupyter notebook on an AWS machine. I opted for an FCN since the Semantic Segmentation lecture mentioned that an FCN, with encoders, a 1x1 convolutional layer and decoders, can preserve spatial information of the input image. In contrast, a convolutional neural network with convolutional layers and fully connected layers would not have spatial information preserved and thus unable to perform well on segmentation tasks. The main difference is in the 1x1 convolutional layer - while a fully connected layer flattens the input and lumps all the values into single values, the 1x1 convolutional layer is a convolution that reduces the input dimensionality but does not eliminate the spatial information.

The FCN contains the following components:

1. Three convolutional layers with separable convolution and batch normalization as the encoders.
2. One 1x1 deep convolutional layer to finalize the encoder part.
3. Three transposed convolutional layers with skip connections to all encoder layers as the decoers.
4. One final output layer using Softmax for classification.



The encoder layers are the same as regular classification neural networks that contain fully connected layers following the encoders. The encoder layers use separable convolutions to spot out spatial distribution features in the image, and the deeper the encoder layers go, the more complex the shape they can identify. For example, the first encoder layer would identify lines, the second encoder layer would identify arms, legs and torsoes, and the last encoder layer would composite them into the shape of a human. The separable convolution technique, compared to naive convolutional layers, reduces the problem of overfitting by requiring less parameters.

The middle layer is a 1x1 convolution layer. As discussed before, the layer would reduce the dimensionality of the data while preserving the spatial information. Reshaping the input also allows the decoder layers to perform upsampling and eventually restore the spatial information.

The decoder layers are transposed convolutional layers that upsample the deep output from the middle layer into the desired dimensionality. Here I used bilinear interpolation upsampling and used skip connections from the input layer and the encoder layers 1 and 2 for each decoder layers. Skip connections can enhance the fidelity of the results by feeding in the original and uncompressed spatial information to the decoders.

Finally, the output layer is a convolutional layer that uses the softmax function to classify each pixel as one of three classes. Since we have three classes, we can easily visualize the result as an RGB image!

3 Parameter Tuning and Selection

I tuned the FCN based on the training and validation losses and the three evaluation items in the notebook. The evaluation items are:

1. Scores for while the quad is following behind the target.
2. Scores for images while the quad is on patrol and the target is not visible.
3. Scores measuring how well the neural network can detect the target from far away.

A few notes on the selection of architecture and parameters:

- I tried deeper networks with 4 and 5 layers of encoders and decoders each. However, the validation performance was worse than the network with 3 layers. With some tuning efforts, the network with 4 layers had validation loss of 0.0532, and the network with 5 layers had validation loss of 0.1041. Both performed worse than the network with 3 layers. Since the training losses of the deeper networks were also much higher, I believe that my training dataset did not contain enough data to fit these deeper networks, and thus resulted in underfitting the model.

- The filter sizes of the encoder blocks started out as 8, 16 and 32. However, I found that the small filters did not capture much of the information when the target and passengers occupy larger areas of the images. Especially for the first evaluation metric, the IOU was as low as 0.305. I gradually increased the filter sizes by a factor of 2 every time, and reach the final sizes of 32, 64 and 128. Filter sizes larger than that do not perform well become the dataset consists of 256x256 images. However, it is interesting that even with the small filter sizes, the network did not perform better in the third evaluation metric, where target from far away needs to be correctly identified.

Selection of the hyperparameters:

- The batch size was selected to be 64. The dataset contains 4131 training images, so one epoch would contain 65 batches. There are 1185 validation images, so the validation step would contain 19 batches.

- The batch size was comparatively smaller than what the memory of the machine can hold. The reason is that with a higher batch size, even though the training loss is similar (between 0.02 and 0.03), the validation and test performances were worse than networks trained with smaller batch size. I looked online and found that the reason for this phenomenon is that large batches usually lead to sharp minima of the loss function, and thus losing the ability to generalize.

  Tradeoff batch size vs. number of iterations to train a neural network on StackExchange

- The original optimizer was the Adam optimizer, and I switched the optimizer to a Nadam optimizer. The Nadam optimizer utilizes the Nesterov Accelerated Momentum, which in theory can accelerate convergence to minima. This pairs well with my lower learning rate than the recommended 0.002 from the keras documentation. I chose a lower learning rate because during trial and error, I found that the losses would converge fast but could not settle to a minima. Thus, I lowered the learning rate for it to converge to lower losses.

- The epoch number of 40 was also chosen by trial and error. I ran the network with 128 epochs and the model seemed to be overfitted to the data by then. I thus reduced the epoch number to be closer to the point where the losses were stable, and ran it a few times to make sure that it converged to a reasonable minima.

4 Improvements

I ran the training experiments many times to tune the hyperparameters and the FCN filter sizes. I found that validation losses are usually fluctuating between 0.03 and 0.04 when the training losses reduces slightly over epochs. As a result, I believe the network is still suffering from some inability to generalize. This could possibly be improved by augmenting the dataset to create a deeper and more robust neural network.

In addition, the network right now does not classify anything beyond non-human, passengers and heros. Since the dataset does not label any other objects, such as cats, dogs and cars, the trained neural network does not identify other objects. To make it possible for the network to segment other objects in the scene, the dataset needs to be appropriately labeled into more than 3 classes, and the output layer of the network should correspondingly increase the number of classes to identify.