Follow Me Project Review

By: Raymond Andrade

Project Overview:

The objective of this project is to control a quadcopter (in simulation) to locate a target object, in this case a person, and then follow it as it moves. This is done through processing images that come from the quadcopter camera and then using deep neural networks, created with Tensorflow and Keras, to process the data received from the images and identify objects of interest. The deep neural network in my project was trained on my computer which has a Nvidia graphics card with CUDA installed using the given Udacity training set, as well as recorded data from the simulator. The training could have also been done on an AWS EC2 instance to complete the training in a more efficient manner.

Neural Network Overview:

Neural networks are used to allow computers to achieve complex decision making. In this project, a neural network is created to read images from the quadcopter camera and identify if a particular person is present in the image. The neural network does this through a model which it classifies as it is trained. Since we need to keep spatial information about the image after it has been classified, this project uses a fully convolutional network, these convolutions help keep the spatial information about the picture which would be lost if fully connected layers were used instead.

Some of the neural network's main parameters used for this project include learning rate, batch size, and number of epochs. The learning rate is the multiplier of the weight adjustment, in this case it is a very small positive number (usually less that one) to ensure the weights of the classifier don't drastically change each time they are adjusted. The batch size is the number of images which are processed with each pass of the neural network. Number of epochs is the amount of times all the data used for training is run through the neural network.

Another main portion of the fully convolutional network is the encoder and decoder in it. The encoder extract features from the image which will then lead to a classifier. The decoder then scales the output of the encoder back to the size of the original image. This allows the network to not only classify if there is a particular object in an image, but identify where it is in the image. The encoder and decoder will also work for any size image since the image always gets scaled back to the original size. This mean this network can be interchanged to find different objects in pictures as long as the training data reflects information about that object and is used to train the model.

Neural Network Application:

In this project, the fully convolutional neural network was applied by using two encoders, which then go to a 1x1 convolution classifier, and then ran through two decoders. The first encoder has a filter depth of 32, the second encoder has a filter depth of 64, the 1x1 convolution classifier has a depth of 128, the first decoder has a depth of 64, and lastly the second decoder with a depth of 32. This second decoder image is then put through a conv2d layer which gives us our segmented image.
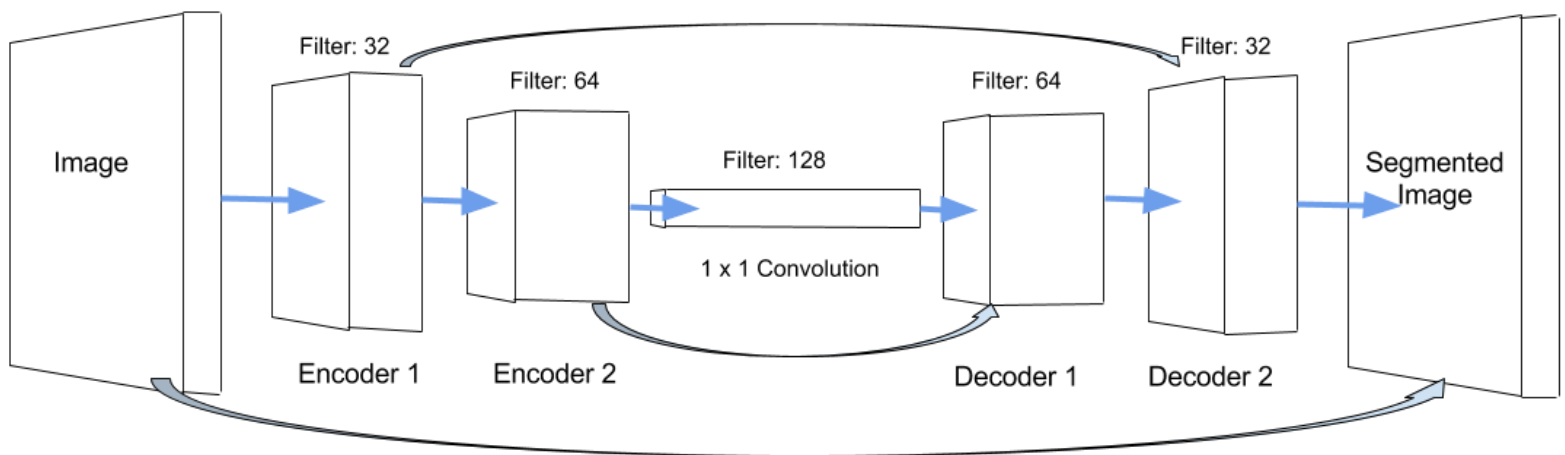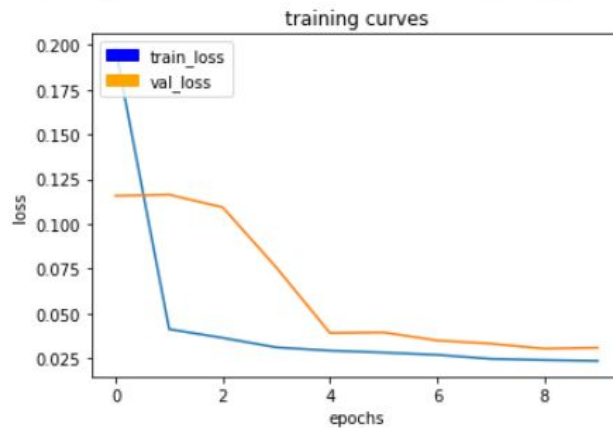
Project Runtime:

After completing the project notebook, I trained the data on my computer using the CPU version of tensorflow. This took a very long time and I left it to train overnight. I unfortunately was unable to fully utilize the GPU version of tensorflow till after I had trained valid CPU data which had a final score greater than the 0.4 needed (using tensorflow-gpu with Windows has proven to be quite the pain to configure, Linux dual-boot is just great that way). I was able to get a final score of 0.419 with the fairly non-resource intensive parameters of learning_rate = 0.005, batch_size = 16, num_epoch = 10, stacks_per_epoch = 125, validation_steps = 50, and workers = 3 with the given training set. These parameters are a lot lower than I would have liked, but being that I was training on CPU at the time, I wanted to keep the training time as short as possible. This data can be viewed in the model_training notebook.

```
Epoch 10/10
124/125 [=============================>.] - ETA: 34s - loss: 0.0236
```

training curves

```
125/125 [==============================] - 4659s - loss: 0.0236 - val_loss: 0.0310
```

Using the Follow Me option in the simulator with the follow.py script and weights from the project notebook running, I was able to successfully follow the "hero" target. It seemed to be able to identify the hero in the crowd fairly easily, only having disconnects if someone were to walk directly behind the hero and hindering the drones view of the hero completely. However, the instant the person blocking the hero got out of the way the drone would resume following the hero.

Future Enhancements:

      Some improvements that could be made to this fully convolutional neural network would be to decrease the learning rate while increasing the amount of epochs that are run. This will most likely increase accuracy, with the drawback of an incredibly long training time. Another improvement would be to increase the batch size, which would also increases training time for more accuracy. Increasing the filter sizes on the encoders, decoders, and 1x1 convolution would also lead to more in depthly classified images, this would also increase the time it takes to classify each image which again leads to a longer training time. Using the simulator record option, I would also like to have processed my own data (which I did record some of my own trials, but didn't want to wait to train over CPU) through the network, which would improve my overall score as long as it contained more images of the hero to train off of.