Follow Me Project Write-up

This project consisted of a lot of waiting and frustration while training the model but I ended up with a final score of **0.456806907266**

The write-up is organized by:

1. Collecting Data
2. Building/Modifying my Segmentation Network
3. Training and Results
4. Conclusion
5. Future Enhancements

Before I started collecting the data, I built a simple fully convolutional network which will be explained more in detail in the section '**Building/Modifying my Segmentation Network**'. This was meant for a dry run with the current data set so I know what it lacks and collect more of it (i.e. hero and non-hero images). This dry run resulted with a final score of 0.405622465323 which was passing but not satisfying so I collected more data and added more layers in my network.

Collecting Data

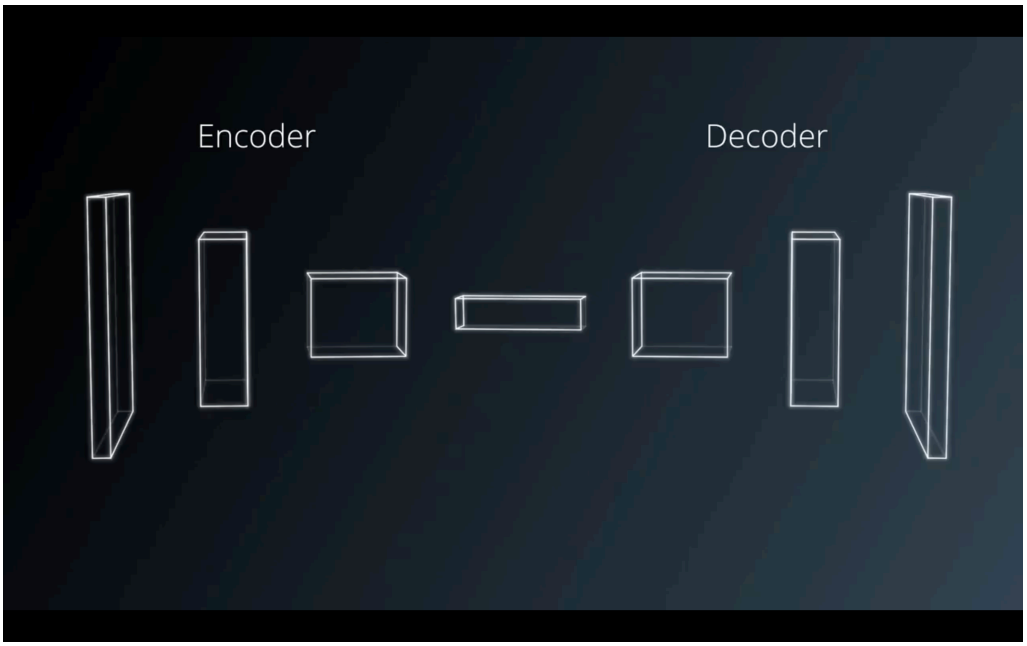
I took the project suggestions after running my first attempt and observed that the average IOU of other people for all three scores were very low as well as the score of the hero from far away. I collected another 750 images of the hero in a crowd and just of other people for each set (training and validation) so a total of 1500 more images. After comparing results of the same network architectures with the same hyper parameters, I managed to gain a better score than the previous one.

Building/Modifying my Segmentation Network

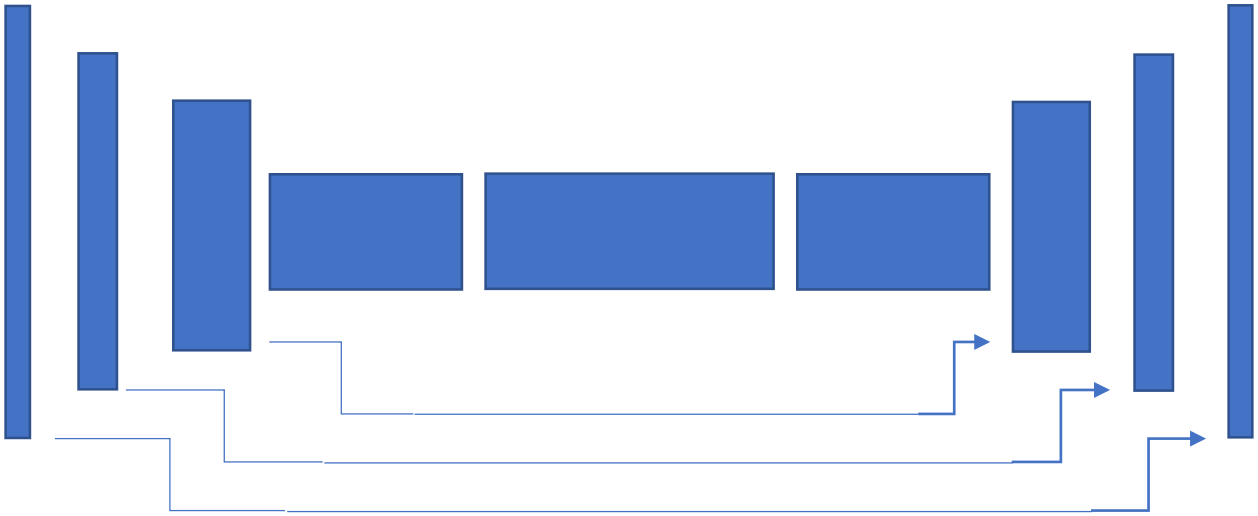
A typical CNN consists of a series of convolutional layers followed by a fully connected layer then a softmax activation function. CNNs are great for image recognition tasks and really shine whenever the data has some sort of hierarchal structure. Deeper networks in general, not only CNNs, has a natural tendency to capture hierarchal structure. At the lowest layers, we'll see simple patterns like lines or edges, then moving up we'll see patterns such as shapes, and as we traverse deeper into the layers, we'll see more and more complex patterns such as objects or faces.

So, for this project we'll use a CNN, though it won't have a fully connected layer in the end instead we'll use a 1x1 convolution and connect it to layers of transpose convolutions to create a fully convolutional network or FCN. Unlike a CNN with a fully connected layer, an FCN preserves the image's spatial information which is an important attribute for this project. Though fully connected layers are great for image classification, it only helps identifying **what** a certain object is in a given image. If we want to know **where** an object is in a given image, we'll have to turn our CNN into an FCN.

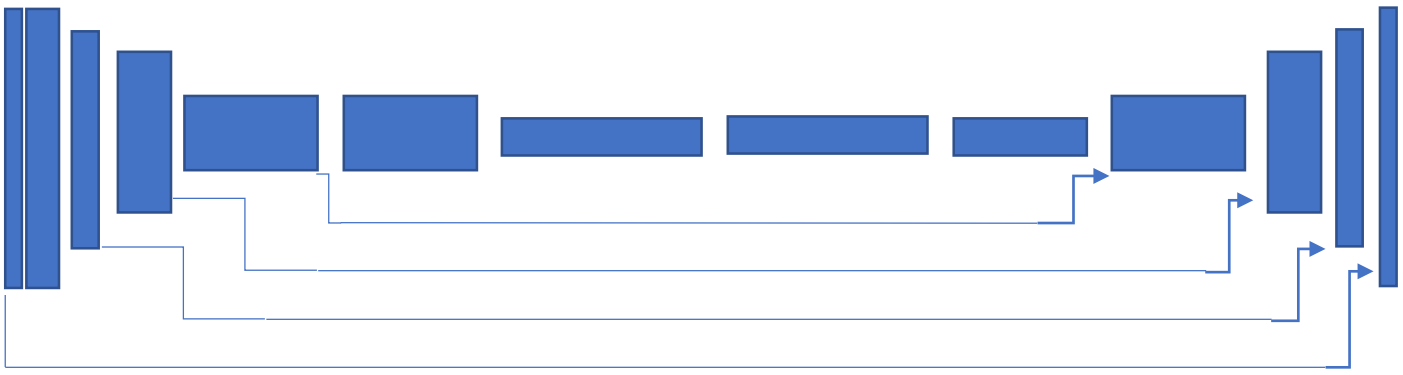
An FCN is comprised of two parts, an encoder and a decoder. The encoder block is similar to a typical CNN but instead of using a fully connected layer at the end, we connect the convolutions from encoder to the decoder using a 1x1 convolution to preserve the image's spatial information and help reduce the dimensionality of the layer. The other part of FCN is the decoder block, this block is composed of transposed convolutions to upsample layers to high resolutions using bilinear interpolation. This block reconstructs the spatial resolution from previous layers and performs a convolution.



The first FCN I built, I decided to use 3 encoder layers in the beginning, followed by 2 1x1 convolution layers, and finally 3 decoder layers. I wanted to capture a solid number of features as the images pass through the encoder block, however, as a side-effect some information is lost due to looking to closely on some features and as a result upscaling in the decoder block will produce inaccurate results. To counter this, in the decoder block, skip connections are added. Skip connections are a way to retain information from multiple resolutions by connecting the output of a transposed layer in the decoder to a non-adjacent layer from the encoder and the results are fed into the next layer.



This yielded to a passing score of 0.40 but I wanted to improve the score and aim for the previous passing score of 0.45 and it seemed like that was the best score I received after multiple attempts with different configurations. I decided then to make my network deeper creating another layer in the encoder block and adding a 1x1 convolution in the middle of the encoder block instead of just limiting 1x1 convolutions in the middle of the network. This enables the model to gain more spatial information while making the network deeper. I also took a suggestion mentioned in Slack and added a regular convolution before all the encoders to extract features and preserve image size, adding this layer yielded to an increase to better results and helped me reach 0.45.



Training and Results

This part was the most tedious part of the project, though it solidified my understanding in tuning parameters. For this I used AWS EC2 to train my model. After modifying my network, I initially used the configuration as follows:

```
learning_rate = 0.01
batch_size = 128
num_epochs = 10
steps_per_epoch = 100
validation_steps = 30
workers = 4
```

This configuration yielded to a score of 0.41, which was a great start. I chose the initial learning rate of 0.01 because this is what I've using in the labs and has produced satisfactory results. For the batch size, I wanted to train the model fast enough so I can experiment more but I also want to maintain a good accuracy of error gradient estimates. After a few runs, I decided 128 was the best value to go with. I changed the recommended value for steps_per_epoch due to the length of training. I wanted to shorten training time so I first tried the recommended method of dividing the number of training data by the batch size but this did not yield to great results so instead I cut the recommended value in half which led to better results. I decided to stick with 10 epochs first and use this as the base to check the loss graphs for every configuration and analyze where I could do better.

After realizing that the loss was learning too fast I decided on the next configuration to lower the learning rate to 0.001 and keeping the rest of the parameters the same from the previous run. This increased the score by another 1%. Still could do better. I decided then to increase the number of training data that goes through 1 epoch to 150 and lower the learning rate down to 0.0085 which yielded to an increase, though not significant. I started noticing from the graphs that the loss could still decrease if I increase the number of epochs and from the lessons, we learned that lowering the learning rate would require more epochs which leads to better accuracy, so I did just that. I increased the number of epochs first to 12 which led to 0.44, which is close but not the score I was aiming for. Looking at the graphs, it seemed like it can possibly lower the loss more, so I increased the number of epochs to 13. Finally, it yielded to the results I was aiming for. My final score is **0.456806907266** with configurations as follows:

```
learning_rate = 0.0085
batch_size = 128
num_epochs = 13
steps_per_epoch = 150
validation_steps = 30
workers = 4
```

After running the model in the simulator, I was pleased to see that the drone was able to successfully follow the hero.

Conclusion

This model wouldn't work very well for other objects such as a car. This model was meant to only recognize our human hero from other people. If we want to target other objects like a car, we'd have to create a new model and train it with plenty of data of our target to distinguish it from other non-target objects just like we did with our hero. We'll still be able to use the same network architecture created for this project; however, we'll need to train a new model to follow a different object along with a new data set of our target object.

This project can be useful for other applications that encodes and decodes images such as object recognition, obstacle avoidance, and other video analysis applications. I can see this to be very beneficial for self-driving cars since they go through semantic segmentation to distinguish road markings and objects around the car such as pedestrians.

Future Enhancements

Based on my model, I would add more images to the training data set, mainly images of the hero in a crowd and images of the hero from far away. I would've also built a deeper network to extract more features and given enough resources, I would increase the batch size, lower the learning rate, and increase the epoch to achieve a better accuracy.