Follow Me

1 Overview

Our goal in this project is to train a Fully Convolutional Neural Network (FCN) to be used for the task of helping a drone follow a “hero” around a simulated city. As the simulator and corresponding control systems are provided for us, we will be focusing on the FCN and related ideas in this paper.

2 Fully Convolutional Neural Network

2.1 FCN Overview

As mentioned in the overview, we will be using an FCN to solve the problem of identifying the hero so that the drone can follow it along in the world. We will start by providing a brief overview of what an FCN is and how it works.

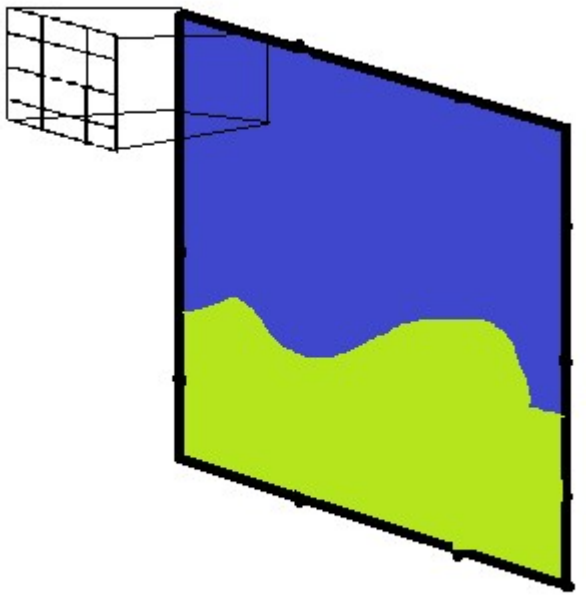

Figure 1 Convolution

An FCN, as the name suggests, is a neural network architecture where all the layers are convolutional layers. In a convolutional layer, rather than connecting all the inputs to each neuron in the layer below, we instead arrange the neurons in a kind of filter which we "convolve" over the input layer. The output of such a layer is a result of the convolution of the learned filter and the input.

Typically, we would have more than one filter in any given convolutional layer, with the output of each filter constituting its own output channel in the result. For example, if we were to have a convolutional layer with say, 32 3x3 filters, then the output of the layer would have 32 channels. The advantage of doing this is that it allows us to learn features that are invariant to their position within the image (or other suitable input).

When performing the convolution step we also to consider the stride of the convolution. This essentially controls how many pixels we skip over when moving the convolution filter over the input. By increasing the stride, we can effectively down sample the size of the input space, compressing the data for the next layer and reducing the number of parameters. For example, we could run a 2x2 filter over a 32x32 grayscale image with a stride of two and get out a 16x16 result from the convolution.

The base example (provided by Udacity's course notes) uses these convolution layers in stages. First, we have the encoding stage, in the encoding stage we chain numerous convolutional layers together, downsizing the output as we go along – while increasing the number of filter channels used. (This downsizing is achieved by using a stride > 1 as explained in the previous paragraph). These encoding layers essentially use the filters to encode the features of interest during training.

At the end of the encoding layer we insert one (or possibly more) 1x1 convolutions. This serves to reduce the dimensions of the problem, and possibly as kind of non-linear co-ordinate transform from the feature space into the output space.

Our end goal with an FCN is to output an appropriate image (in our case one that has segmented the pedestrians and the hero from the background information). To get this image we stack an equal number of "decoder" layers as we had encoder layers to the end of the FCN. In each stage of the decoder portion of the network we up sample the input to the layer by the same factor that the data was down sampled in its corresponding encoder layer. We will use a technique called "bilinear up sampling" to achieve this, using the code provided by Udacity.

A final feature of our "prototypical FCN" is the skip connection. This attempts to alleviate the possible loss of important spatial information during the encoding phase caused by down sampling, by bypassing some of the encoding layers and concatenating their output with the input to decoding layers later down the line. This allows the decoding layers to work with multiple different resolutions of the original input.

Figure 2 shows a schematic diagram of the setup we have described above.

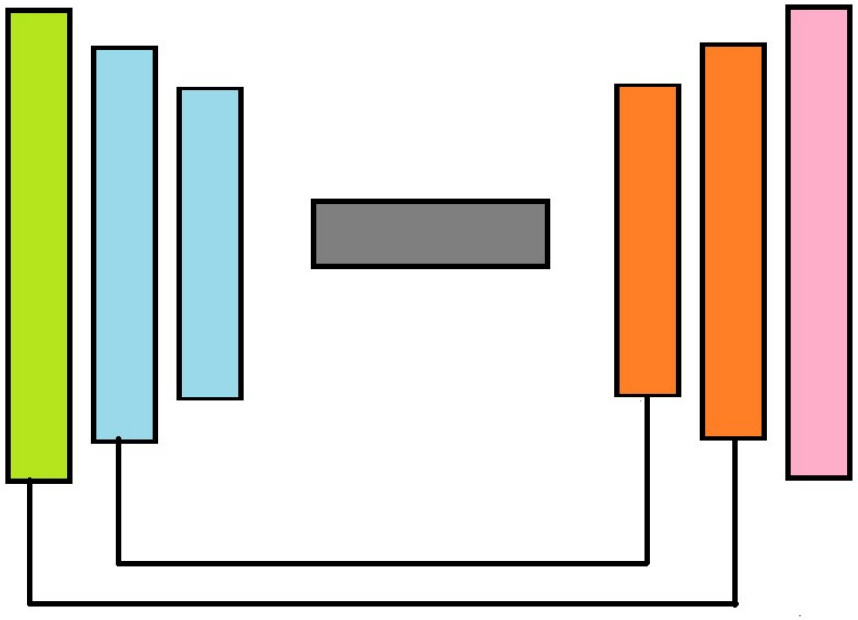

Figure 2 Basic Layout of FCN

2.2 Training Data

To properly evaluate the performance of each of the architectures that were considered, the same data set was used in the training of each. This data set was constructed by downloading the updated data set provided by the Udacity course materials (some original experimentation was done with the first data set provided, but given the increased quality of the newer data set it was decided to switch to this newer data set instead), and then augmenting the data by running a flip procedure on each of the images and their respective masks. This data augmentation was done using the augment_data.py script, which has been included in this projects submission.

With the data augmented, this provided us with approximately 8200 training examples, with examples from each of the three main uses cases:

- Images while following the target
- Images while not following the target
- Images with the target “faraway”

2.3 Architecture

Several network configurations were tried to determine a good solution to the problem. Each of the attempted solutions was trained using the same data described in the previous section, for four 25

epoch cycles. This training method was chosen to help determine when the network would start "overfitting" the given data.

All configurations were trained using the same set of hyper-parameters, shown in the table below. Optimization was done using the NAdam[1] optimizer in Keras

PARAMTER	VALUE	NOTES
LEARNING RATE	0.002	Recommended learning rate for the NAdam optimizer.
BATCH SIZE	128	Largest batch size that did not result in Resource allocation issues during testing.
NUMBER OF EPOCHS	25	Model was trained in bursts of 25 epochs at time, then evaluated for performance (and overfitting).
STEPS PER EPOCH	64	Roughly the number of training examples divided by the batch size
VALIDATION STEP	12	Roughly the number of validation examples divided by batch size
WORKERS	2	Default value, seemed to get good results.

Performance for each of the configurations experimented with are found below. Note, that while the configuration was varied considerably, the data never was. This oversight was because the experimenter hit his compute budget on AWS.

2.3.1 Base Architecture

This is the first configuration that was evaluated and found to reach a satisfactory solution. The configuration used here essentially mimicked the example layout used in the overview. To avoid potential pitfalls with overfitting, the filter channels for each layer were kept comparatively small (at least compared to the other models that were tried). The number of channels per layer is annotated on the top of each layer in figure 3.

As can be seen from figure 3, this configuration consists of an input layer (green), followed by two down sampled encoding errors (blue). These are then followed by a single 1x1 convolution of depth 256. Our

[1] Incorporating Nesterov Momentum into Adam, Timothy Dozat, http://cs229.stanford.edu/proj2015/054_report.pdf

decoding setup is a mirror of the encoding setup, with skip connections to the outer most encoding layer and the input. Finally, a standard convolution layer is applied to the result of the decoders and used as the output of the network.

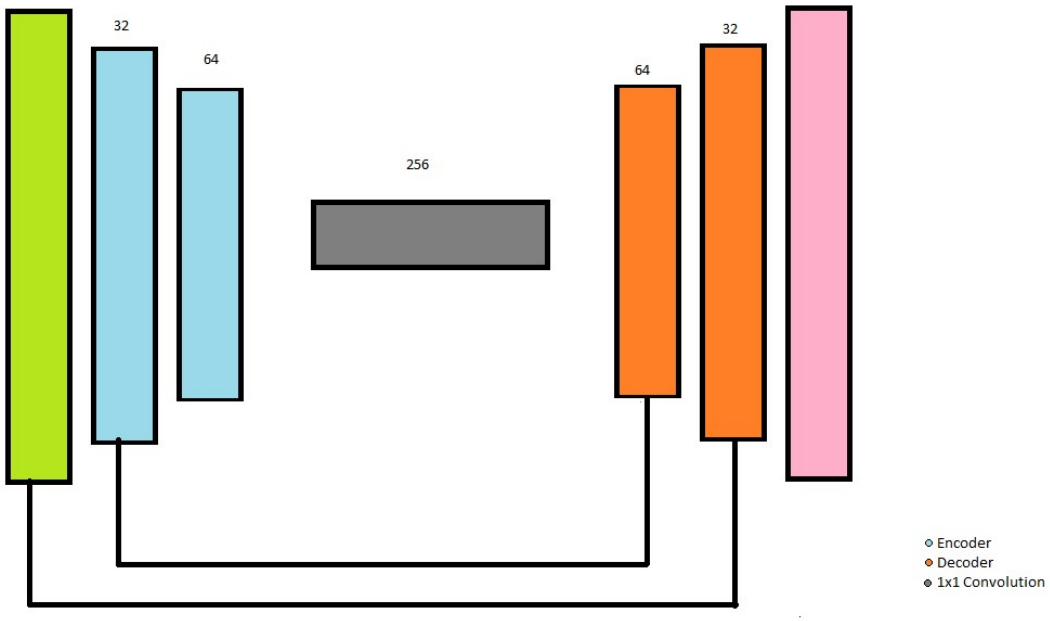

Figure 3 Base Architecture

The final score for this configuration at 50, 75, and 100 epochs is detailed in the table below:

EPOCHS	FINAL SCORE
50	40%
75	41.5%
100	41.9%

2.3.2 Deep Convolutions

A natural extension to the Base Model was to simply pile on more layers to the network, in hopes that making it larger would solve the problem with better accuracy. Unfortunately, possibly due to not having sufficiently diverse training examples, this model was found to overfit the training set, reaching a maximum final score of 38.6%. As before, results after each set of epochs are outlined in the following table:

EPOCHS	FINAL SCORE
25	36%
50	36%
75	38.6%
100	31.4%

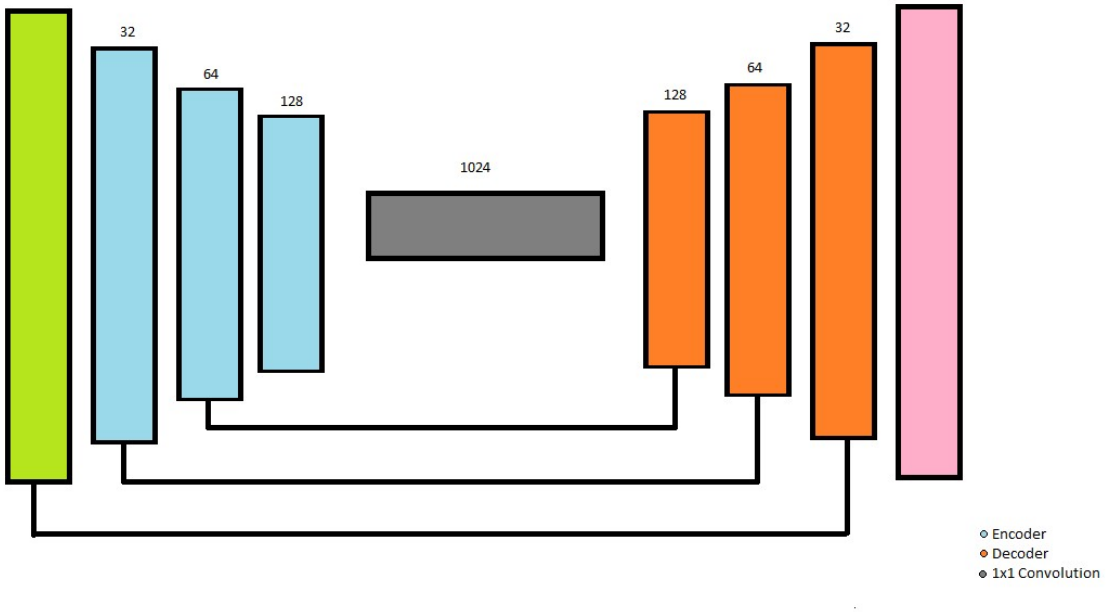

Figure 4 Deep Convolutions Architecture

2.3.3 Network in Network

Inspired by the approach taken in the "Network in Network" [2]paper that first introduced 1x1 convolutions, an approach involving multiple 1x1 convolutional layers was attempted. This configuration extends the base model by adding two additional layers of 1x1 convolutions to the bridge between the encoder and decoder layers. The intuition behind this is that these chained 1x1 convolutions will allow for a non-linear mapping between the encoding and decoding stages, allowing the network to learn more complex filters.

EPOCHS	FINAL SCORE
25	34.2%

[2] Network in Network, Min Lin et all 2014, https://arxiv.org/pdf/1312.4400v3.pdf

50	35.1%
75	41.6%
100	43.17%

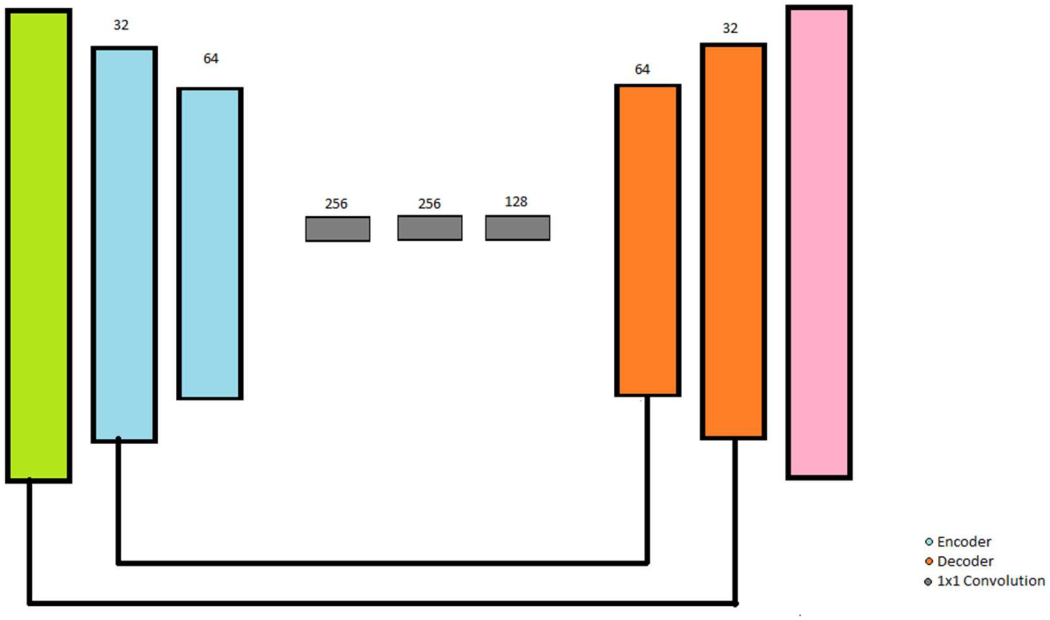

Figure 5 Network in Network Architecture

2.3.4 Increased Filter Depth

Another variation of the original base model that was tried was to simply increase the number of filter channels used for each convolution layer. In case configuration, experimentation started by taking the base model from 2.3.1 and increasing the size of the filter channels until resource errors began to occur. This resulted in the configuration shown in figure 5, and achieved a maximum final score of 41.9% after 100 epochs.

EPOCHS	FINAL SCORE
25	35%
50	39.5%
75	41.2%
100	41.9%

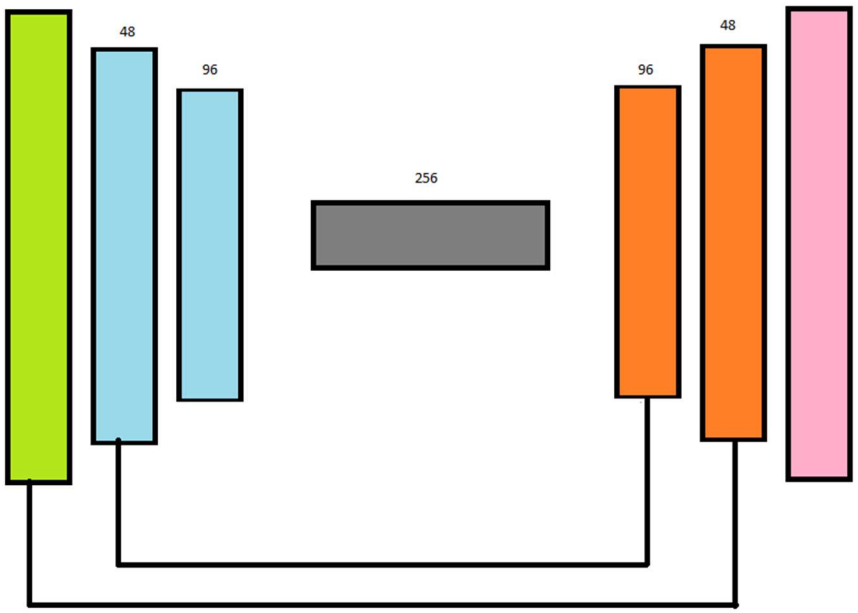

Figure 6 Deep Filter Configuration

2.3.5 Reduced sampling

Previous iterations of the model all appeared to have the same weakness – whenever the hero was at periphery of the camera's vision, it would either be misidentified or omitted completely from the models output. It was hypothesized that this defect could be the result of too much information being lost during down sampling.

To get around this problem, Strategy 3 from a conference paper on "Squeeze Nets"[3] was attempted. Unfortunately, only a very trivial attempt at this was possible as without the use of downsizing, the model very quickly exhausted the available resources on the AWS instance. As such, although this technique may have been able to solve the problem, it seems that the network was not big enough for it to make a difference. The results of experimentation with this failed attempt are included below:

EPOCHS	FINAL SCORE
25	21.4%
50	33.1%
75	36.6%
100	36.1%

[3] SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS, Forrest N. Landola et all, https://arxiv.org/pdf/1602.07360.pdf

3 RESULTS

Given the results collected, the so called “Network in Network” model was selected as the FCN to be used with the drone. This model was chosen primarily because it had the best performance of the evaluated models using the data provided (43.2% versus 41.9% for the base model after 100 epochs). This extra bit of performance did however come with a size tradeoff – the Network in Network model is about 592KB while the base model was only 225KB. In applications where model size was a serious issue, the second model would have been chosen instead as it has comparable performance at half the size. Code for this network has been included in the corresponding juptry notebook (and its html companion) attached to the project submission.

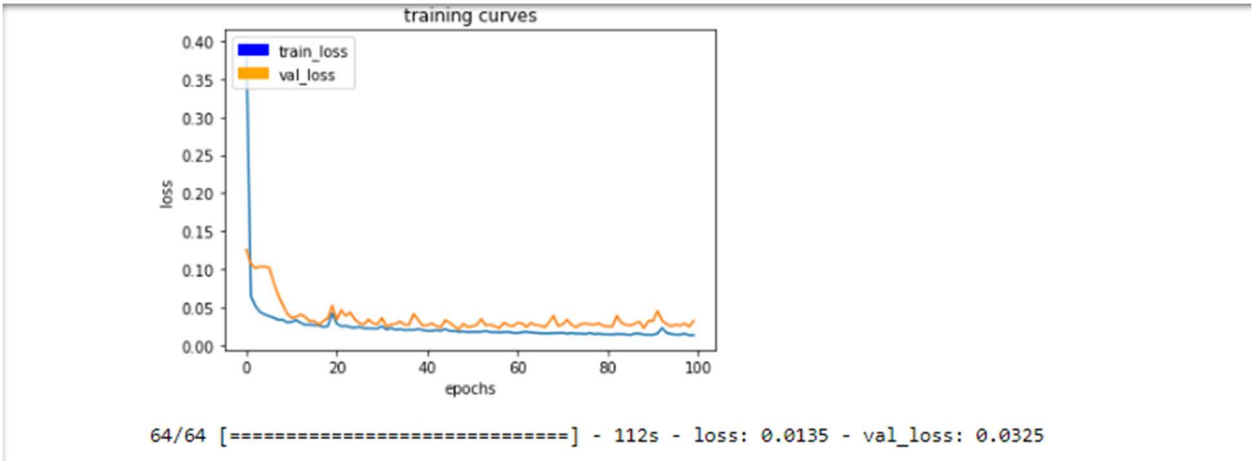

Figure 7 Training Curve

As can be seen in Figure 7 “Network in Network” model achieved a value loss of approximately 0.0325 after 100 epochs using the training parameters discussed in the previous section. Most of the value / loss optimization was obtained in the first 20 – 40 epochs, with the remaining epochs being used to explore the output space.

No further training was done on the network beyond the 100 epochs discussed here, so it’s still possible that further iterations may further improve the achieved score without overfitting the data.

Figures 8, 9, and 10 show some random samples from the validation set used to evaluate the models final score. As can be seen from the examples, despite being the highest scoring of the models used, there is still a lot of “noise” in the final output. In addition to the noise scattered about, the model has trouble identifying the hero consistently when the hero is at the edge of the its field of view. Possible methods for mitigating these issues will be discussed in the next section.

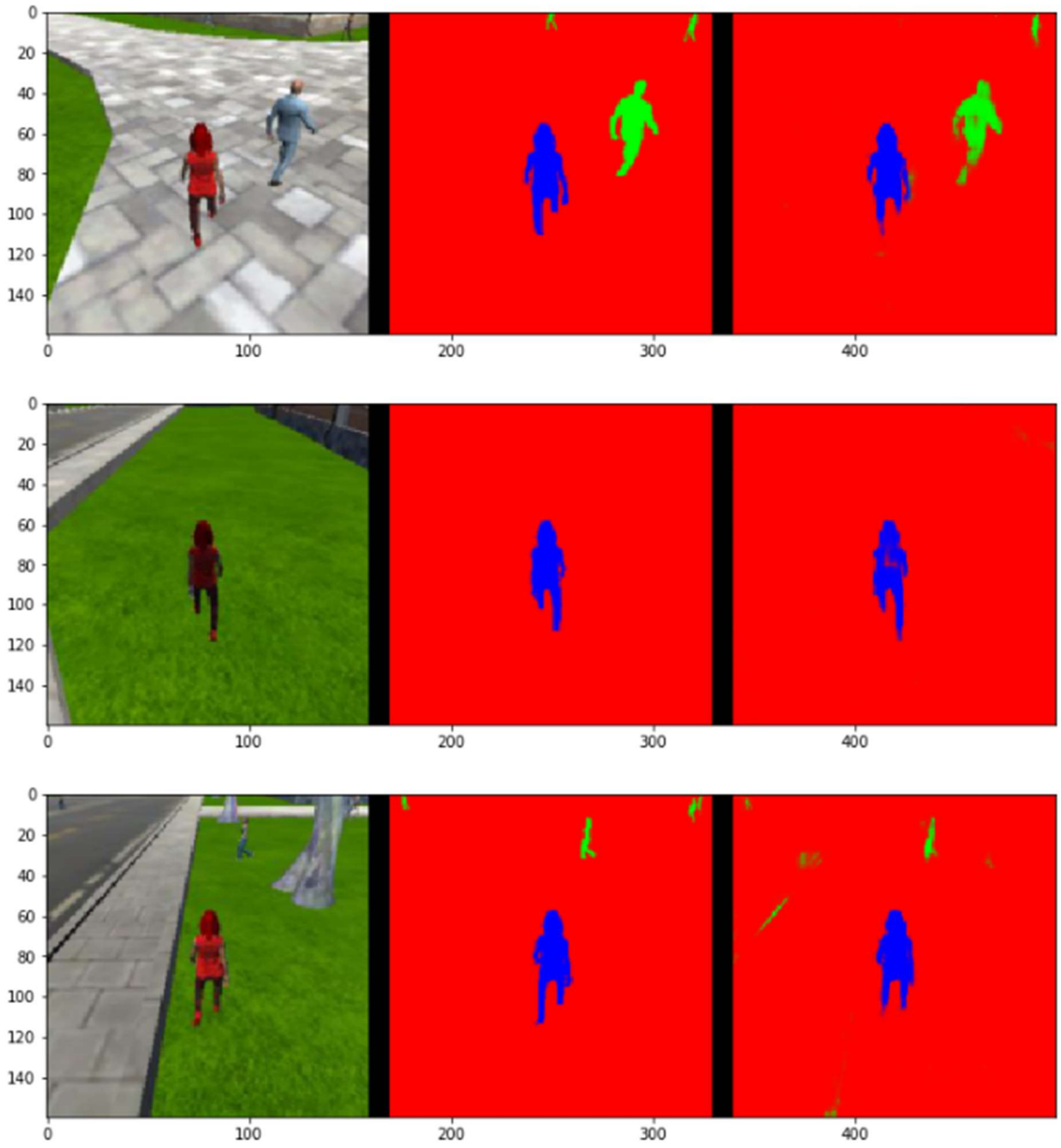

Figure 8 Following Output

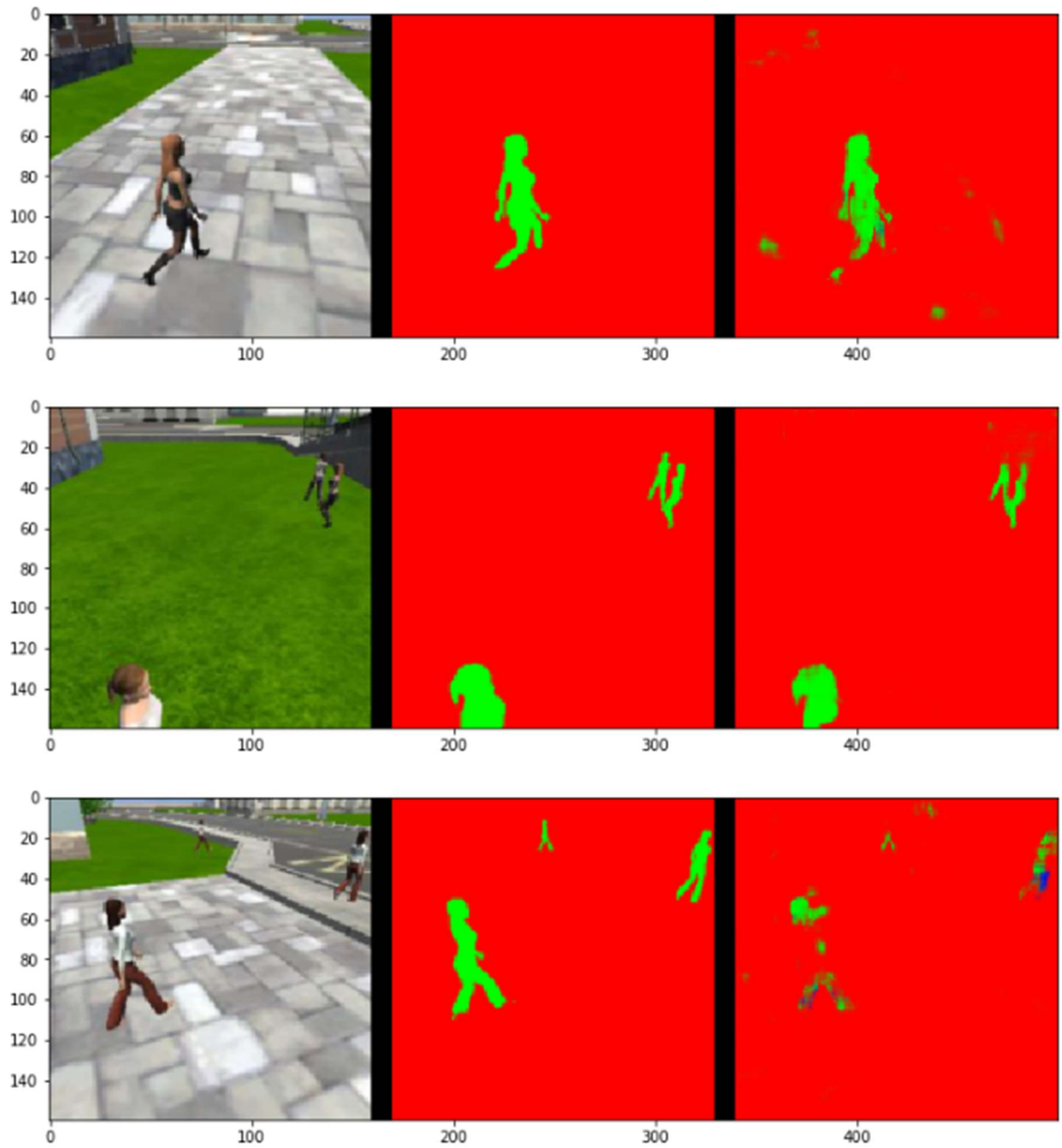

Figure 9 Patrol Output

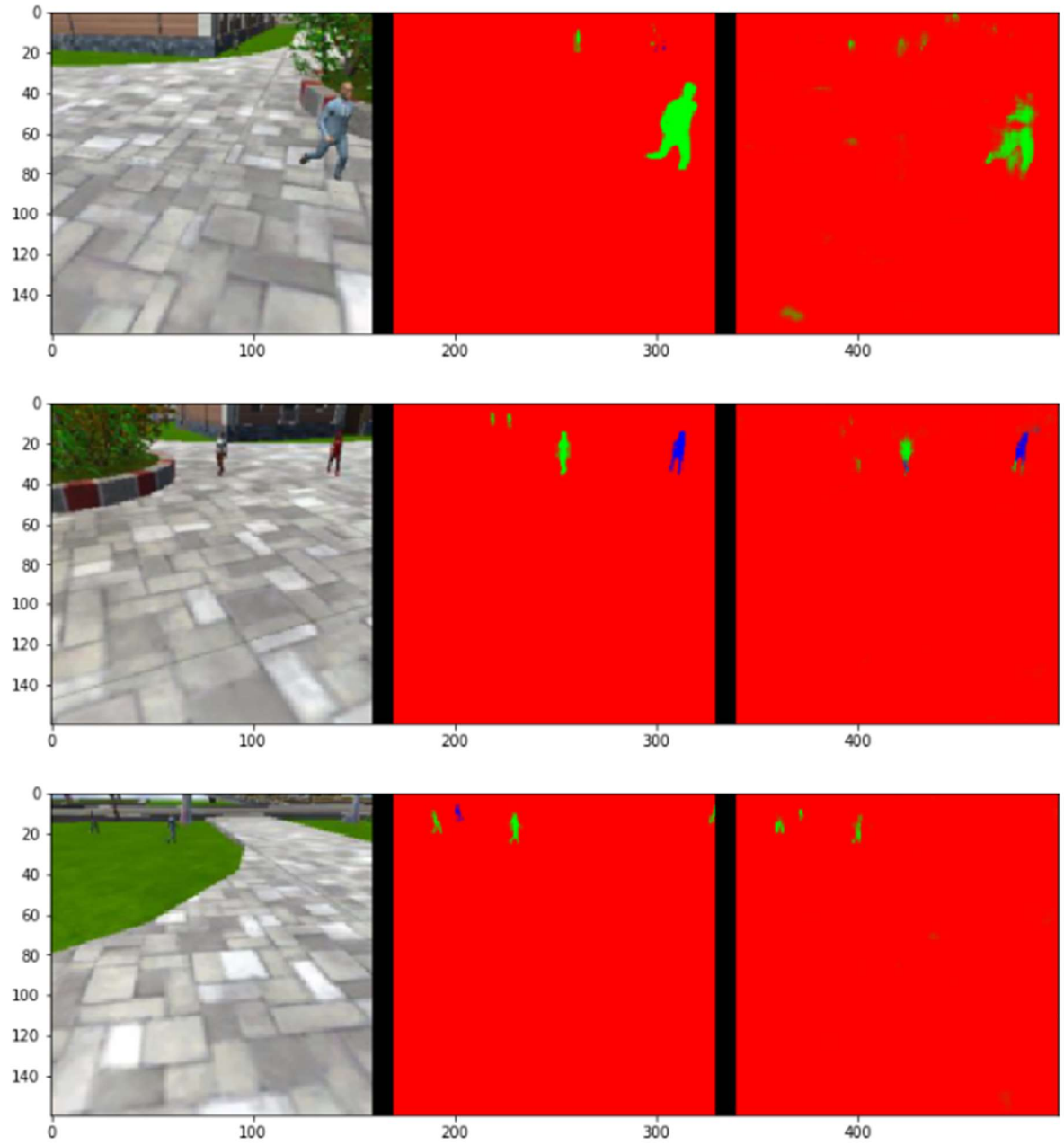

Figure 10 Patrol with Target

To confirm that the model was functioning as expected, it was also tested inside the provided simulator. Insider the simulator the drone successfully used this model to successfully follow the hero through the crowded streets and complete it patrol. There were occasions where the drone would temporarily lose

track of the target, but it was generally found again quite quickly. Figure's 11 and 12 below show examples of the drone performing its following behavior in the simulated environment.

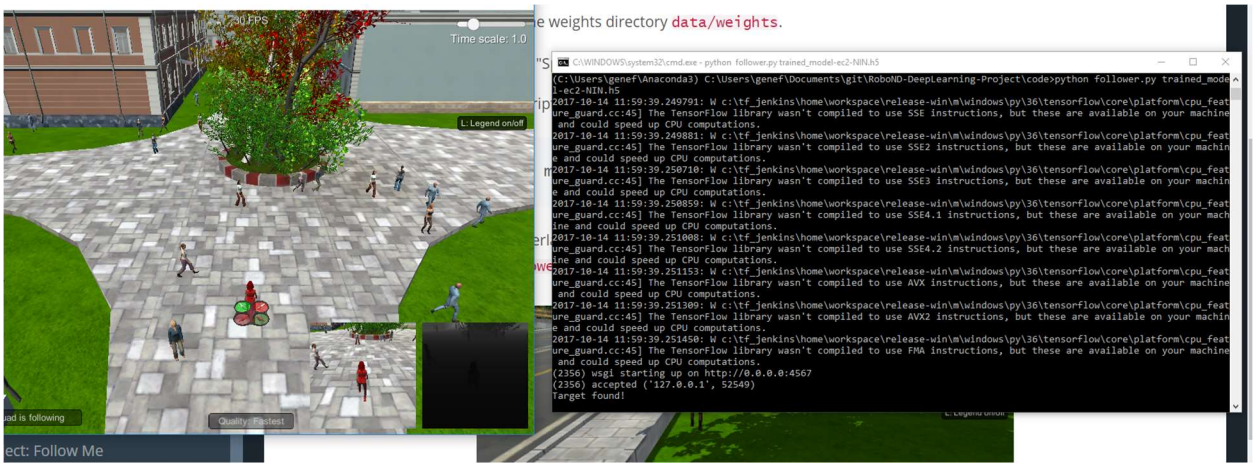

Figure 11 Drone Following Hero 1

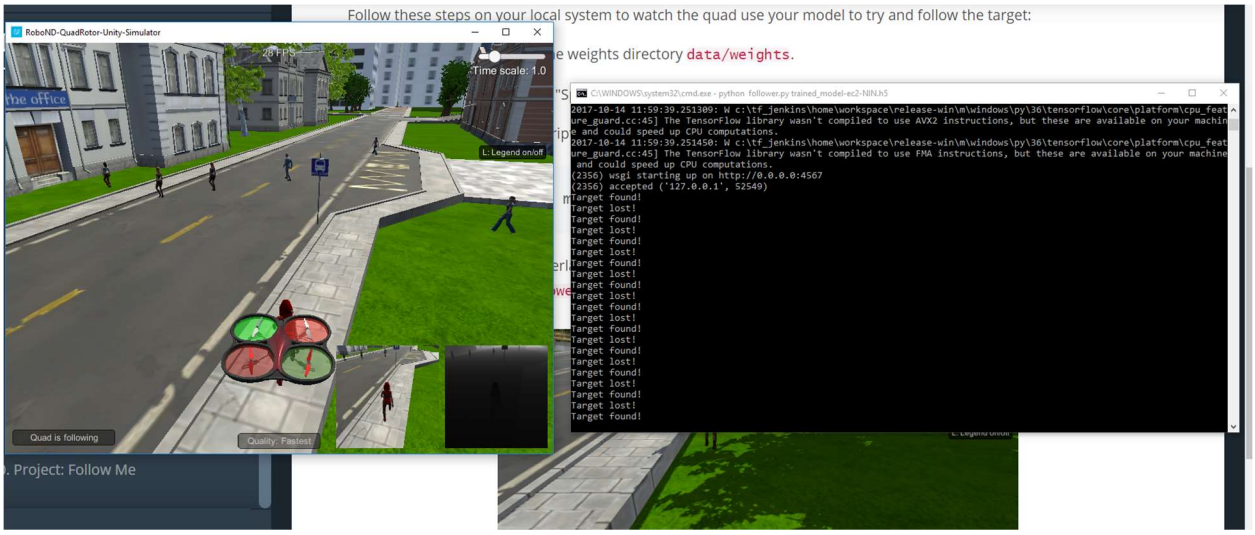

Figure 12 Drone Following Hero 2

4 Discussion

We've developed a model for semantic segmentation that segments human pedestrians from the background in a simulated environment. It is likely with a bit of further refinement that this model could be extended to work with real world images with an appropriate dataset as the fundamental shapes we are looking to extract in both the real and simulated environments are similar.

Hypothetically, the experimenter expects that a real-world application of this system may suffer with reduced performance due to increased noise – birds flying around, pieces of garbage laying about, cars, wind, camera noise – all factors that do not appear to be included in the simulation. Furthermore, there is the danger of the drone being fooled by real world department stores – fundamentally the model present is only taking spatial properties into account, not temporal, so any statue or manikin may cause false positives. Double so if for some reason the hero's fashion sense matches said inanimate objects (say for example Halloween – that could be a very confusing time for the model).

Granting the above limitations, there is no reason why this system could not be trained to any roughly human sized object for which training data is available. Smaller, or incredibly fast-moving objects may cause an issue however. As was seen in the results section, the camera has trouble picking out the hero when its profile is very small (which in this case is when its far away), it could well be the case that even with training it will have difficulties with small heroes regardless of their locations making a "follow me" for birds, insects, cats, small dogs, and perhaps even small children unreliable with the given network.

While the solution presented can solve the problem at hand, there is still room for improvement. There are a few enhancements that have not been evaluated here which may improve the systems performance:

- Further Pre-Processing may be possible for this application. For example, in the sample collection project at the start of this term we implemented some classical algorithms for extracting the "road" from the input. As pedestrians are assumed to be one the road (and not say flying in the sky) it may be possible to exploit these techniques to reduce the size and complexity of the input space.
- The kernel size could be increased (or maybe even decreased). To keep the number of permutations of the network configuration low enough that it wouldn't blow the AWS computer budget this was not tried here, but there was at least one member in the Udacity slack channel for this project that seemed to make reasonable gains by use of this method.
- More data could have been collected – this would have been especially useful for examples of the hero at the periphery of the cameras vision, where as shown earlier this model is at its weakest.
- An increase in GPU power to allow for the Reduced Sampling setup briefly discussed still might offer improvements in picking out the hero when it is far away. The extra filters trained before the inputs are downsized away may help alleviate the current weakness.
- We could increase the number of 1x1 convolutions used, perhaps including some variation of drop out (simply setting random filter outputs to zero during training) to avoid any overfitting issues.
- Application of post-processing, which is currently not being done at all. We could use a Morphological filter (OpenCV Open / Close) or Gaussian Blurring to remove the noise in the output and fill in incompletely recognized objects. This could allow us to use a model trained with far fewer epochs (or even a small model) and perhaps get the same results.