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 by using skip connections, rather than a fully connected layer.

Skip connections work by taking the output of one layer, and connecting it to a layer that it's not directly in contact with. While each layer of the network reduces the spatial information of the original image as it goes through the encoder, the image is classified and then decoded, then due to the skip connections it is able to retain more detail, like the picture from the notebook on the right, of what the original

image looked like and match the classification accordingly. Without the skip connections, the segmented image would be more “blurry” and lacking in detail, so instead of classifying only the pixels the object is in, would classify a broader area of neighboring pixels around the object as well.

This project also takes batch normalization into account using the functions separable_conv2d_batchnorm(), used for the encoder and decoder separable layers, and conv2d_batchnorm(), used for the 1x1 regular convolution. Batch normalization is when each layer’s input is normalized, generally by taking into account the mean and variance of the values in the data. This generally creates a model that is more centralized, and allows for faster model training and provides regularization.

Lastly, a main portion of the fully convolutional network is the encoder and decoder. 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 the objects in different pictures and different sized pictures.

Neural Network Parameters:

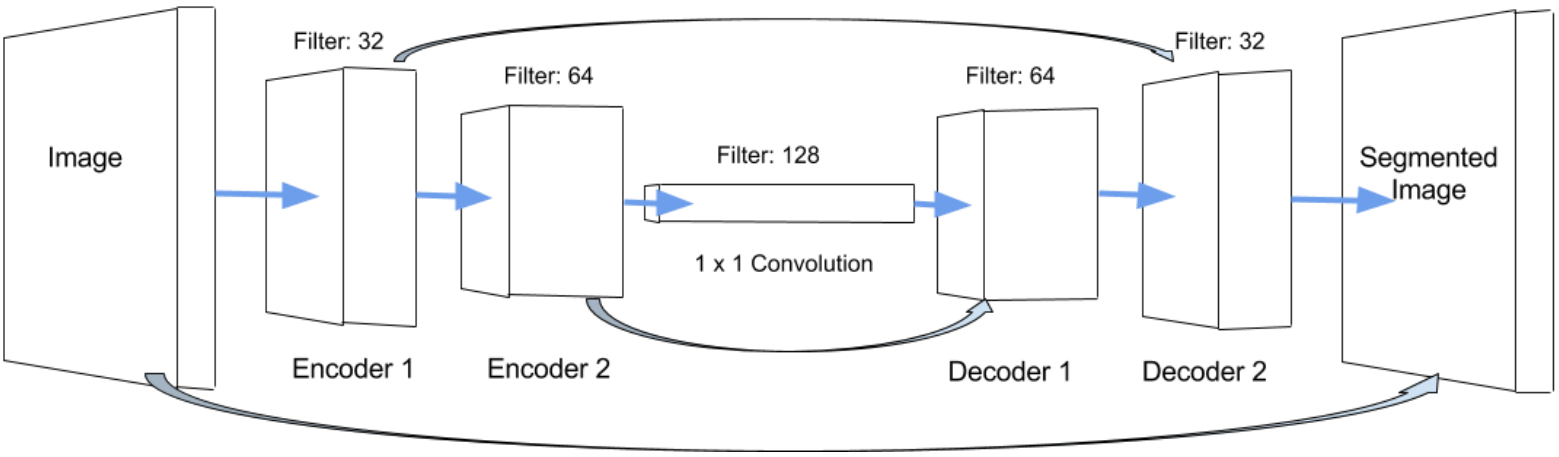
Some of the neural network’s main parameters used for this project include learning rate, batch size, and number of epochs. The learning rate is the multiplier of the weight adjustment, in this case it is a very small positive number (usually less that one) to ensure the weights of the classifier don’t 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.

In this project I used the following parameter values: learning_rate = 0.005, batch_size = 16, num_epoch = 10, stacks_per_epoch = 125, validation_steps = 50, and workers = 3. These parameters are not as ideal as they could be, but a big drawback of choosing parameters that would lead to better accuracy is that the training time will increase quite drastically. The

learning_rate = 0.005 allows for improvement as the model scores each epoch, but not throw off the training too badly if a batch with very little helpful data (images with the hero) is run through the network. I chose batch_size = 16 because 16 pictures images through the network in a pass is a good value with the amount of images given to train with, if the amount of images were increased too much, it might cause "imprinting", which causes the model to remember the features of the images in the dataset rather than the features of the object. If bath_size is too low, it will lead to a higher chance of batches which might contain no useful information toward improving the model. For num_epoch, I chose the value of 10 because after playing with different sizes, found that the accuracy changes minimally past ~8 epochs. This is a minimalistic approach so that a model can be created quite quickly with enough accuracy to properly identify the target and have decent scores at the end of the notebook. More information regarding better parameters can be found in the "Future Enhancements" section of this report.

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.



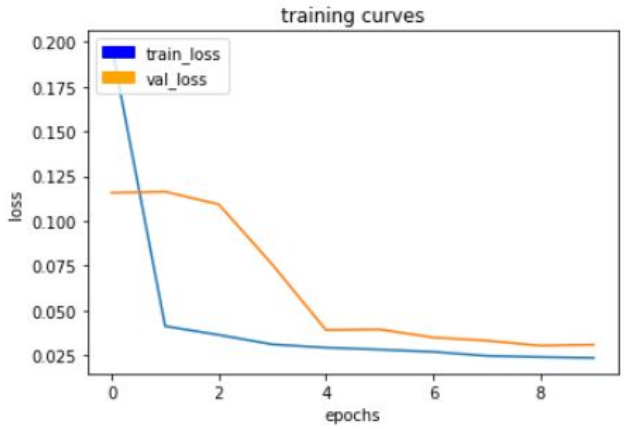
In this neural network used for this project, a skip connection is used to connect the input layer to

the output layer, encoder 1 to decoder 2, and decoder 2 to encoder 1 as shown in the picture above.

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

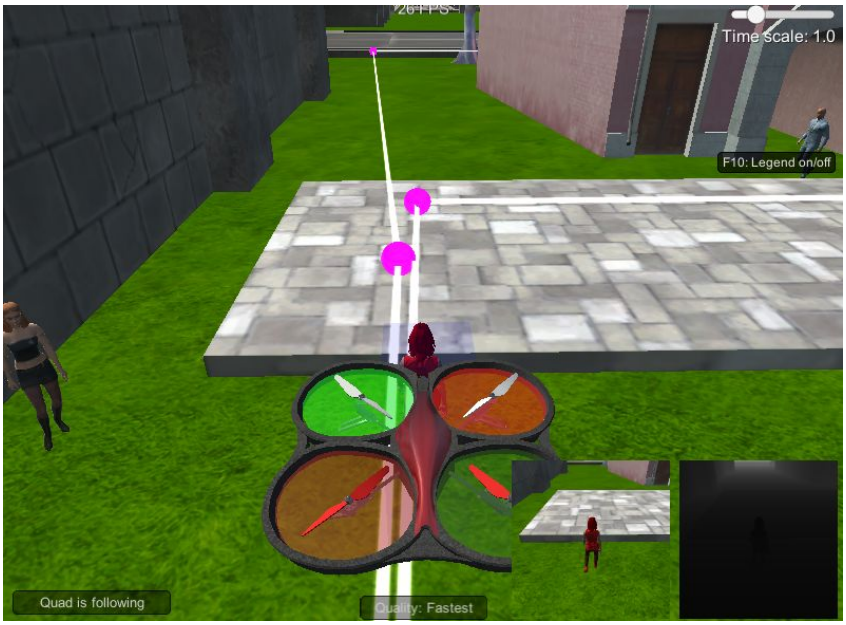
```
Epoch 10/10
124/125 [============================>.] - ETA: 34s - loss: 0.0236
```




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

```
In [54]: # Sum all the true
         true_pos = true_pos
         false_pos = false_p
         false_neg = false_n

         weight = true_pos/(
         print(weight)

0.7387096774193549

In [55]: # The IoU for the d
         final_IoU = (iou1 +
         print(final_IoU)

0.567561051662

In [56]: # And the final gra
         final_score = final
         print(final_score)

0.419262841389
```

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.



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. This could also be implemented by making the learning rate decrease over 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.