Follow Me project.

Introduction:

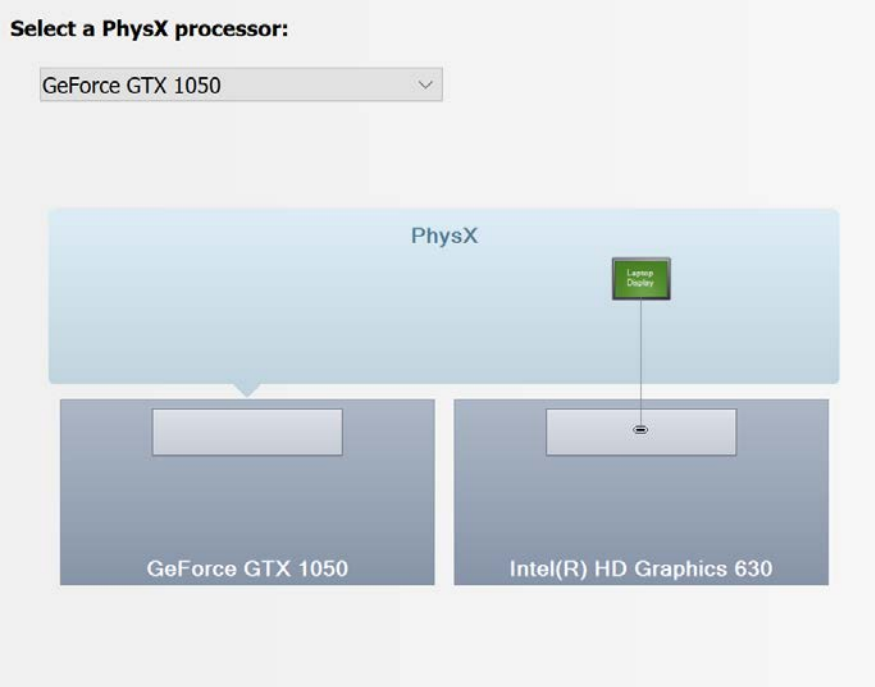
The objective of the follow-me project is to train a fully convolutional network architecture on two sets of data.

a) Training data
b) Validation data.

The dataset I used is the new and improved datasets provided by Udacity. I have tried to improve my score as well by generating more pictures using the provided Udacity simulator but I did not continue down that path. Later in this write-up, I will elaborate on what I have done to more around creating large training and validation datasets at the end of this report. I used the intersection over union IOU metric to measure how well the model is dividing the prediction pixels and ground truth pixels and divided by the "Target" hero with red dress.

I trained my network using my local GPU NVIDIA card, which was not working in the beginning, and then I tried using AWS. Amazon was very late in replying to my email (requesting increase in instances limit) and after I received a positive response from them, I still could not connect to my provisioned server instance from my windows 10 using Putty application and using AWS CLI.

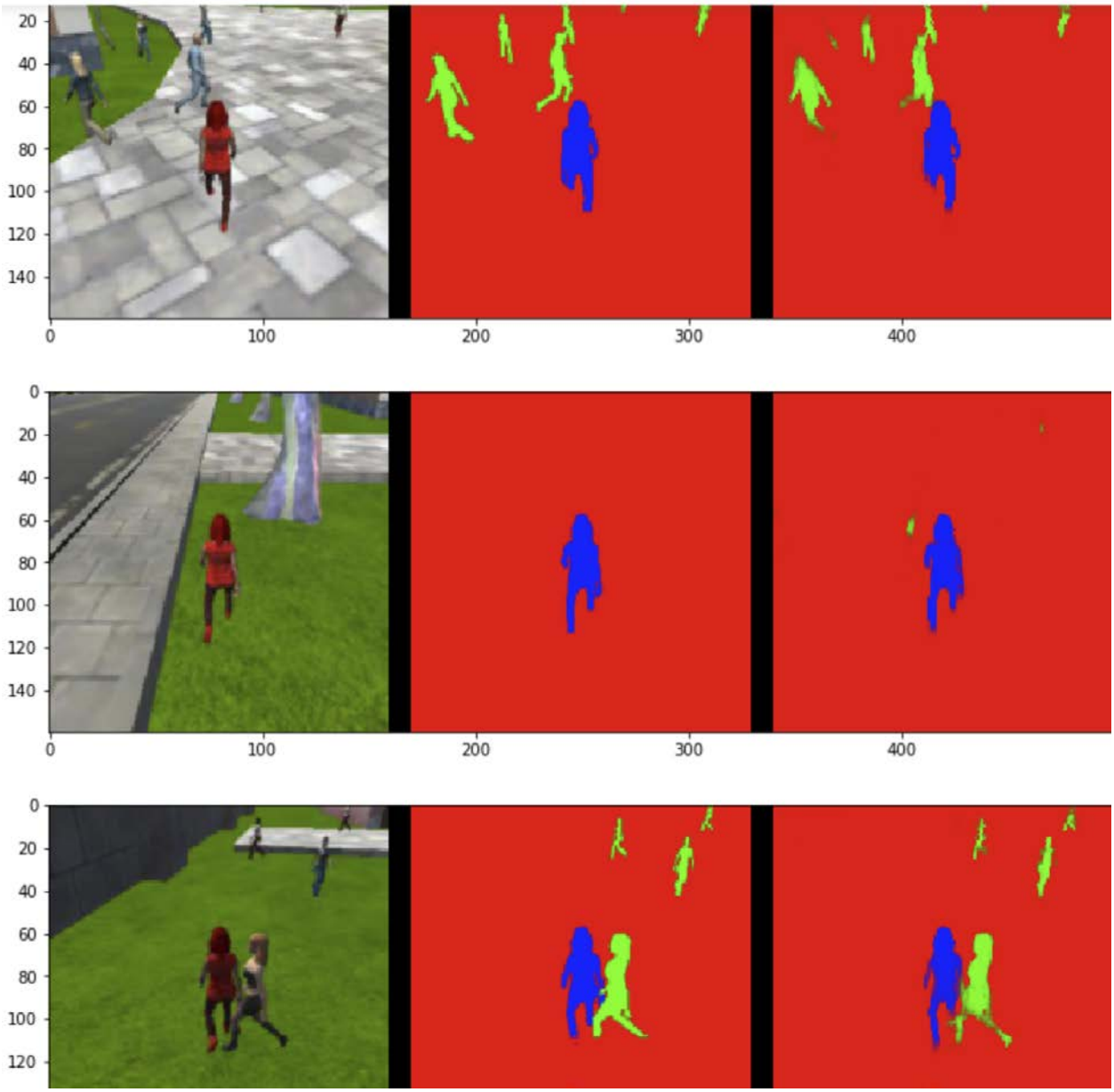
I ended up installing and uninstalling tensorflow several times along with cuDNN 5.1 and CUDA 8.0



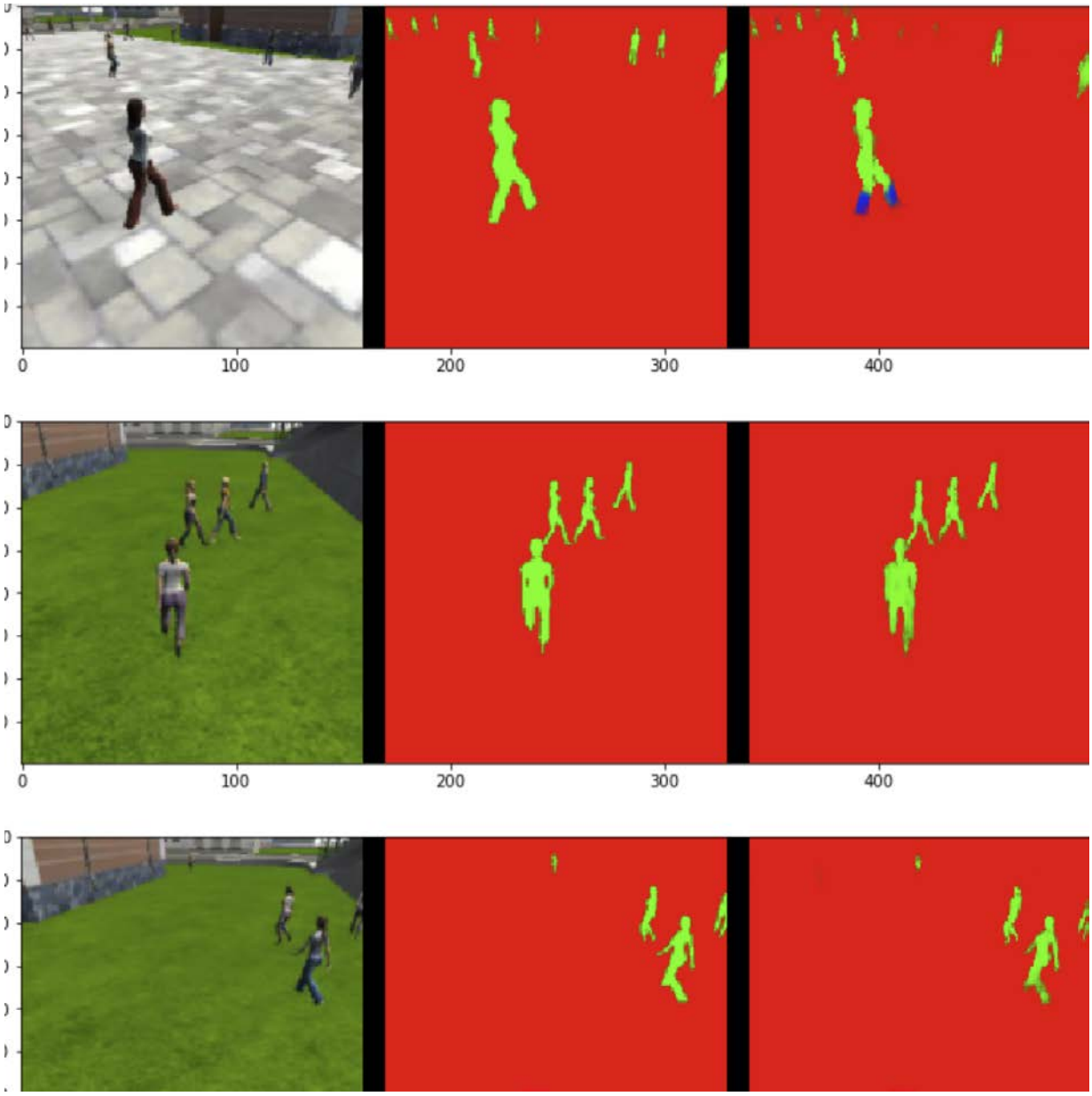
Modelling & Network Architecture:

Normally in computer vision scenarios we start with convolutional neural network with fully connected layer but in the follow-me project scenarios we are not detecting static objects but moving target in a

high density of other not targeted moving objects. This is a challenge for the CNN with fully connected layer because with CNN the spatial information get lost and may get significantly reduced which means in practice the result trained model might lead to a drone which doesn't follow the hero and goes into different unexpected direction/path. Here comes the benefit of using fully convolutional neural network FCNN. CNN is efficient for filtering dimension space rather than spatial dimension space. Standard CNN is good in generalising the model for images that it has not trained on before, which makes overfitting less of an issue but the spatial information gets lost or reduce.



Target hero in red outfit



Non Target people (No Hero in the pictures).

Using the encoder-decoder architecture the FCNN does the following:

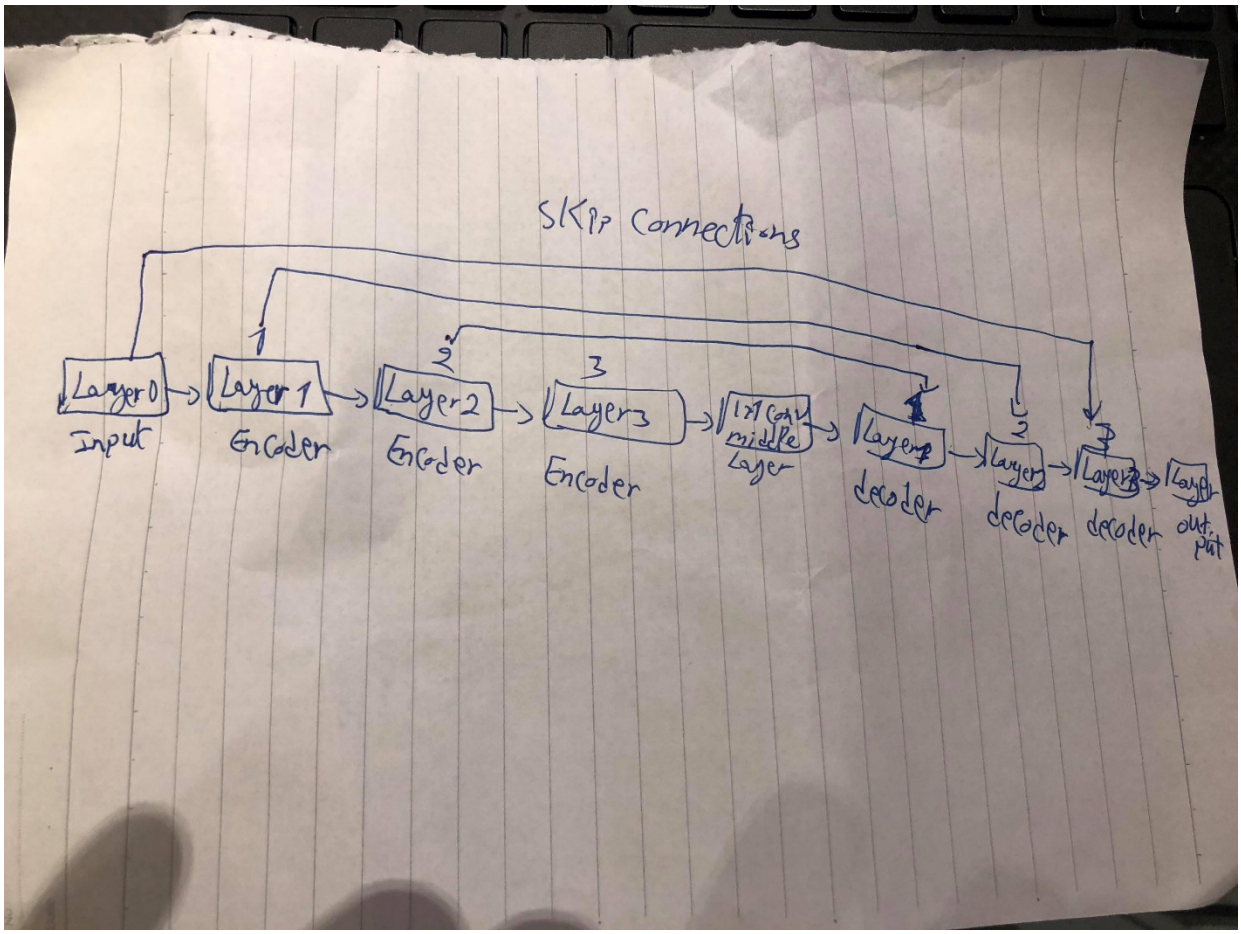
The encoder down sample the high resolution input images(s) to a low-resolution image(s), which the decoder up sample again by mapping it to full resolution input, image(s) for pixel-wise classification.

I started the FCN model with a random choice of 3 encoding layers plus 3 transposed convolution & decoding layers) with possibility of increasing the layers of the overall training loss is high and the IOU score is way above 40%. However, the training and validation of the model have shown less loss value and goo IOU by changing the hyper parameters only and keeping the network layers to three layers only.

In [29]:
```
def fcn_model(inputs, num_classes):

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
    f = 32
    encoder_layer1 = encoder_block(inputs, f, 2)
    encoder_layer2 = encoder_block(encoder_layer1, 2*f, 2)
    encoder_layer3 = encoder_block(encoder_layer2, 4*f, 2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    mid_layer = conv2d_batchnorm(encoder_layer3, 4*f ,kernel_size=1,strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decoder_layer1 = decoder_block(mid_layer, encoder_layer2, 4*f)
    decoder_layer2 = decoder_block(decoder_layer1, encoder_layer1, 2*f)
    decoder_layer3 = decoder_block(decoder_layer2, inputs, f)

    # The function returns the output layer of your model. "x" is the final layer obtained from the last decoder_block()
    outputs = layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(decoder_layer3)

    # Print network shapes
    print(inputs)
    print(encoder_layer1)
    print(encoder_layer2)
    print(encoder_layer3)
    print(mid_layer)
    print(decoder_layer1)
    print(decoder_layer2)
    print(decoder_layer3)
    print(outputs)

    return outputs
```

1x1 convolutional layer is used above to retain the spatial information and

There was some information that was captured in the initial layers and was required for reconstruction of the network during the up-sampling done using the FCN layer. If I would have **NOT** used the skip architecture then the information from input layer would have been lost (or I should say would have turned too abstract for it to be used further). So the information that we had in the primary layers can be fed explicitly to the later layers using the skip architecture.



Skip Connections

Lessson 5.10 & Lession 5.11 (Skip connections and transposed convolutions).

HyperParamters:

```
learning_rate = 0.003
num_epochs = 70
batch_size = 40
number_of_validation_images = 1184
number_of_training_images = 4131
learning_rate = 0.003
num_epochs = 70
batch_size = 40
steps_per_epoch = number_of_training_images
validation_steps = number_of_validation_images
```

While tuning the hyperparamters does not have a standard way to follow and it is more of keep trying and tuning until reaching the lower gradient descent observed through lowers loss in the training and validating of our model. However, the following are what the basis of tuning my parameters was:

Batch size:

The larger the dataset is the longer time it takes to train your model and thus additional hardware might be required to training and validate your model in a reasonable and timely manner. The batch size I used in the first tries was below 30 and above 50 and in both cases, I was not completing the training for all batches because the loss value was highly increasing with no sign of decreasing until more than half of the batches. Large batches usually leads to a generalised model with lots of detail not captured. This may result into a need for increasing the learning rate.

Small batches in contrary leads to model overfitting which may result into a need for decreasing the learning rate.

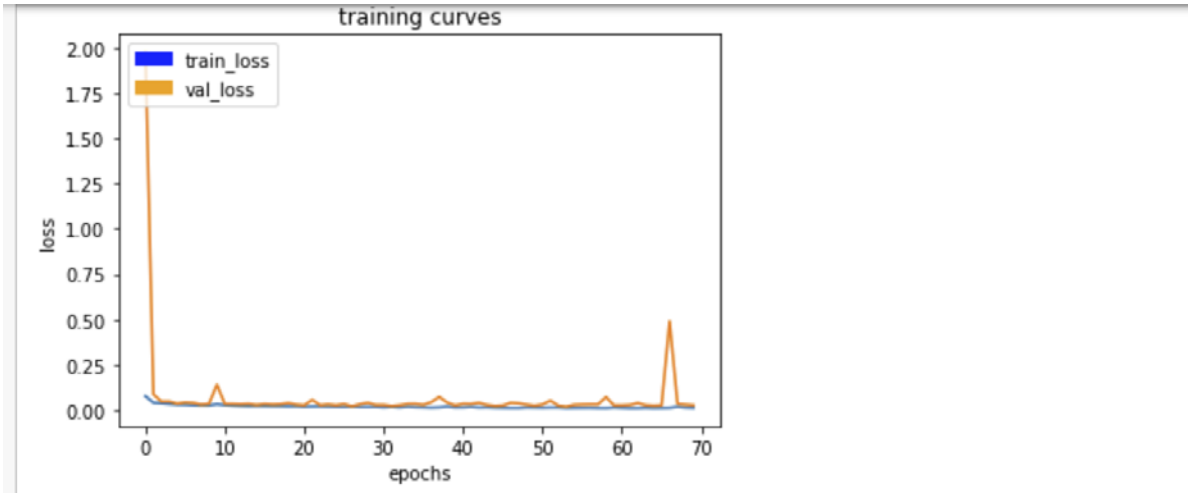
Epochs: More iterations on the data would result into more learning of the data and overfitting could happen while less iteration on the data would result into less learning of the data and a generalised model.

Epochs steps: Number of input layers / images, which normally should be the number of objects to train i.e. in this example. The same applies for validation/test steps per Epochs.

Learning rate: Obviously higher learning == overfitting & lower learning rate == under fitting / generalized.

I did few runs and tuning to the hyper parameters. I started with very low learning rate and then started to increase the learning rate and number of batches.

I have attached in my submission two notebooks showing the last training and evaluation process achieving a very low training anad validation loss. [loss: 0.0785 - val_loss: 1.9752]




```
104/104 [==============================] - 130s - loss: 0.0135 - val_loss: 0.0275
```

And achieving and IOU over 40%. As shown below.

```
In [62]: # And the final grade score is
         final_score = final_IoU * weight
         print(final_score)
```

```
0.411754952259
```

Future enhancements:

This model can be more inclusive to include more types of objects such as pets, cars, birds but the model should be trained to detect these other object types.

Also to improve the performance of the model we can generate more photos from the simulator which different position for the hero such as having the patrol path far from the hero and make sure that such difficult vision situation will be captured as well. I have tried many times in generating such photos using Udactiy simulator but I face some issues with the preprocess_img python script giving me error in the 2nd time I generated images again. I also tried to increase. I also increased the density of people to make a tough challenge for the drone to detect the hero. However I didn't take this to the end because of the very limited time I had.

Addition as per the reviewer request:

I cannot think of anything to capture more objects at this stage other than adding more images with the other objects captured in these images then classify the objects in these images to car, dog, people and hero. Perhaps I would think of change the model to make it more generalised by decreasing the learning rate as the possibility of difference in the shape of objects would increase. As the dataset will increase, I would create more batches to speed up the learning process. I can't think of anything else at this stage.


RoboND-QuadRotor-Unity-Simulator
30 FPS
Not Recording
F10: Legend on/off
RoboND-QuadRotor-Unity-Simulator
30 FPS
Not Recording
F10: Legend on/off