FOLLOW ME PROJECT

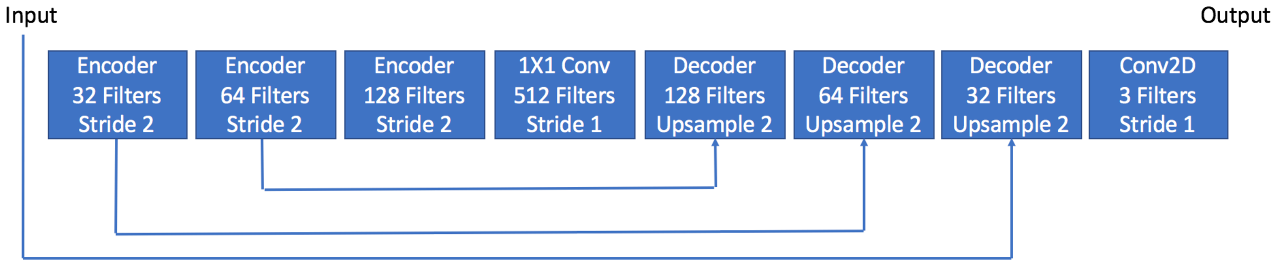
The following sections outline my implementation of the segmentation network used in the Follow Me project.

NEURAL NETWORK ARCHITECTURE

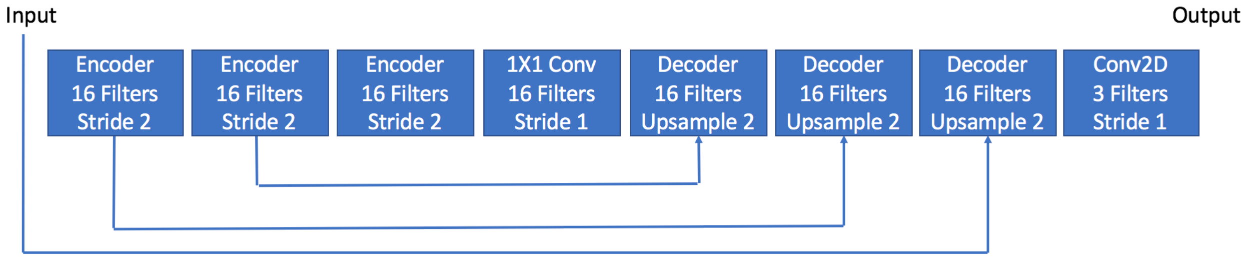
The diagram below illustrates the fully-convolutional neural network architecture with which I was able to obtain a final weighted IOU score of 0.42. The network takes as input a 160x160x3 input image and passes that image through 3 encoders. Each encoder applies a different number of 3x3 kernels with a stride of 2 followed by batch normalization. Since we use a stride of 2, the output of the final encoder is 20x20x128. At this point we've gone from a wide and shallow input (image with 3 channels) to a narrow and deep representation (128 channels).

The decoding section of the network begins with a 1x1 convolution using 512 filters which is necessary for preserving spatial information (something that would be lost using a fully-connected layer). The output is then fed into the three decoders. Each decoder up-samples the input by a factor of 2; concatenates the result with an input from the encoder stage using a skip connection; and applies two convolution and batch normalization layers using a different number of 3x3 kernels with a stride of 1.

The output of the last decoder has dimensions 160x160x32. This tensor is passed into a final convolutional layer that is responsible for producing an image segmentation of dimension 160x160x3 (3 channels for each segmentation class).


Input
Output
Encoder 32 Filters Stride 2
Encoder 64 Filters Stride 2
Encoder 128 Filters Stride 2
1X1 Conv 512 Filters Stride 1
Decoder 128 Filters Upsample 2
Decoder 64 Filters Upsample 2
Decoder 32 Filters Upsample 2
Conv2D 3 Filters Stride 1

An architecture that I explored that did not meet the performance requirement (reached a score < 0.40) is illustrated below. The difference between this architecture and that shown above is the constant number of filters (16) used at each stage of the network. This results in a shallow representation at the output of the third encoder (20x20x16 vs 20x20x128 above) which does not allow the network to effectively perform the segmentation tas


Input
Output
Encoder 16 Filters Stride 2
Encoder 16 Filters Stride 2
Encoder 16 Filters Stride 2
1X1 Conv 16 Filters Stride 1
Decoder 16 Filters Upsample 2
Decoder 16 Filters Upsample 2
Decoder 16 Filters Upsample 2
Conv2D 3 Filters Stride 1

TRAINING HYPERPARAMETERS

Learning Rate: I chose a learning rate of 0.001 to ensure that network learning could progress at a reasonable rate without the risk of failing to converge.

Batch Size: I chose a batch size of 32 so as to avoid having very noisy gradient estimates as well as to well utilize the GPU (and avoid excessive copying of data from host CPU to GPU).

Steps per Epoch: I had approximaly 4000 training images, and I wanted each epoch to pass all the data through the network. With a batch size of 32, this constrained my choice for steps per epoch to be 130.

Number of Epochs: I initially trained the network for 30-40 epochs and found that my score hovered between 0.36-0.38. Training the network for 130-140 steps (an additional 100 steps) pushed my score upt to 0.42.

Validation Steps: I had approximately 1000 validation images, and I the validation score to encompass all of them. With a batch size of 32, this constrained my choice for validation steps to 30.

OTHER APPLICATIONS

The architecture presented here is general enough to be used for segmentation of other objects (e.g. roads, trees, pets). In order to do so, one must however provide additional input images and masks as training data. Once that is done the network can output a channel for each target class (e.g. one channel for the hero, another for roads, a third for tree etc).