FOLLOW ME PROJECT

This is the Deep learning project of Nanodegree Robotics Software Engineering at Udacity. The projects use Fully Convolutional Network (FCN) to build a segmentation network to identify and track a target person from simulated drone camera video stream. The network was built using TensorFlow and Keras and trained using p2.xlarge GPU instance from the Amazon Web Service (AWS).

NETWORK ARCHITECTURE

The project uses the architecture called Fully Convolutional Network (FCN). The FCN model answers much more difficult question like where in the picture is the certain object. Because while performing convolution FCN preserve the spatial

information throughout the entire network. FCN employs some major techniques which include 1X1 convolution layer, upsampling and skip connection for its network segmentation. And these techniques are encapsulated in two parts of FCN called encoder and decoder which are discussed in detailed below.

Encoder

The goal of encoder is to extract features that will be later used by the decoder. Encoder is a series of separable convolution layer that reduce convolutional layer to deeper 1X1 convolutional layer, this has effect of preserving spatial information from an image. And separable convolutions used by encoder help to reduce number of parameters need, hence increase efficiency for the encoder network.

```
def encoder_block(input_layer, filters, strides):


    # TODO Create a separable convolution layer using the separable_conv2d_batchnorm() function.

    output_layer =separable_conv2d_batchnorm(input_layer, filters, strides)


    return output_layer
```

Decoder

Decoder is a second component of FCN that upscale or upsample the output of the encoder such that it has same size as the

original. There are many ways to achieve upsampling but this project make use of bilinear upsampling.

The decoder consists of three main parts:

- ✓ **Bilinear Upsample**

  The bilinear upsample is a resampling techniques that utilizes the weight average of four nearest known pixel located diagonally to a given pixel to estimate a new pixel intensity value. The weight average is usually distance dependent.

- ✓ **Layer Concatenation Step**

  This is similar to skip connection. The upsampled layer is concatenated with a layer that has more spatial information than the upsampled one.

- ✓ **Separable Convolution Layer**

  Some additional one or two separable convolution layer is applied to extract some more spatial information from prior layers.

```
def decoder_block(small_ip_layer, large_ip_layer, filters):


  # TODO Upsample the small input layer using the bilinear_upsample() function.
  upsampled_layer = bilinear_upsample(small_ip_layer)
```

```
    # TODO Concatenate the upsampled and large input layers using layers.concatenate
    large_op_layer = layers.concatenate([upsampled_layer,large_ip_layer])

    # TODO Add some number of separable convolution layers
    output_layer = separable_conv2d_batchnorm(large_op_layer, filters)

    return output_layer
```

FCN Model Put Together

After you have the encoder and decoder blocks, next thing is to build your FCN architecture. Building FCN architecture involves three steps:

- ✓ Add encoder block layer(s). This is the same way of how you add a regular convolutional layer in CNN
- ✓ Add 1x1 convolutional layer. Note the 1x1 convolution one kernel and one stride.
- ✓ Add decoder layer(s)

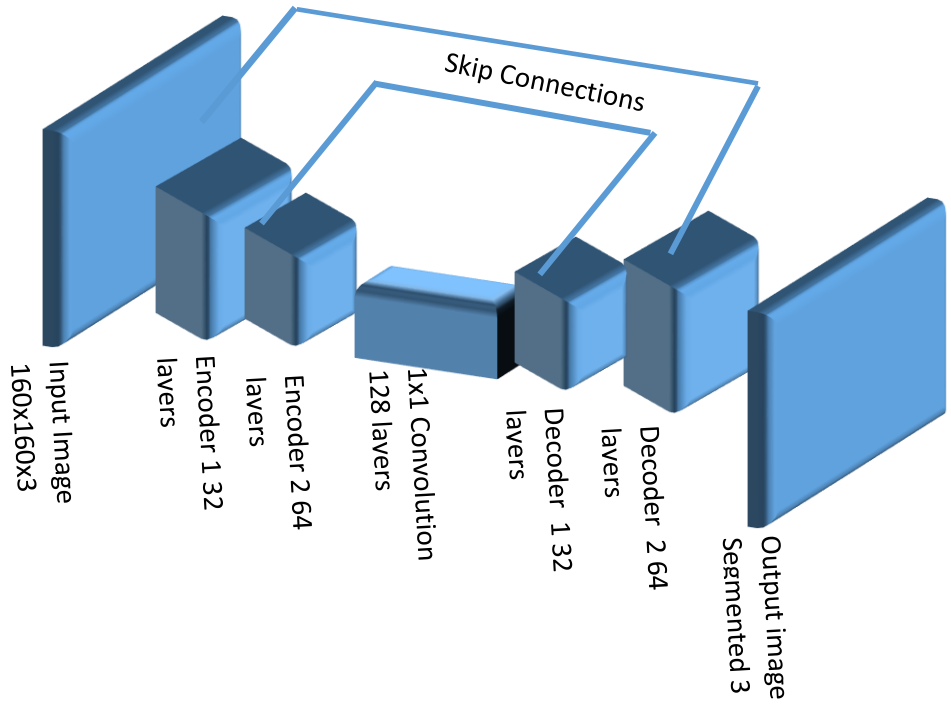
```
def fcn_model(inputs, num_classes):

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
    conv1 = encoder_block(input_layer=inputs, filters=32, strides=2)
    conv2 = encoder_block(input_layer=conv1, filters=64, strides=2)
```

```
    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv_1x1 =conv2d_batchnorm(input_layer=conv2, filters=128, kernel_size=1, strides=1)


    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    deconv1 = decoder_block(small_ip_layer=conv_1x1, large_ip_layer=conv1, filters=64)
    deconv2 = decoder_block(small_ip_layer=deconv1, large_ip_layer=inputs, filters=32 )


    # The function returns the output layer of your model. "x" is the final layer obtained from the last decoder_block()
    return layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(deconv2)
```



Hyperparameters

The optimal hyperparameters used for training this model are:

- Learning rate: 0.001

- Batch size: 100
- Number of epochs: 10
- Steps per epoch: 200
- Validations steps: 5
- Workers: 2

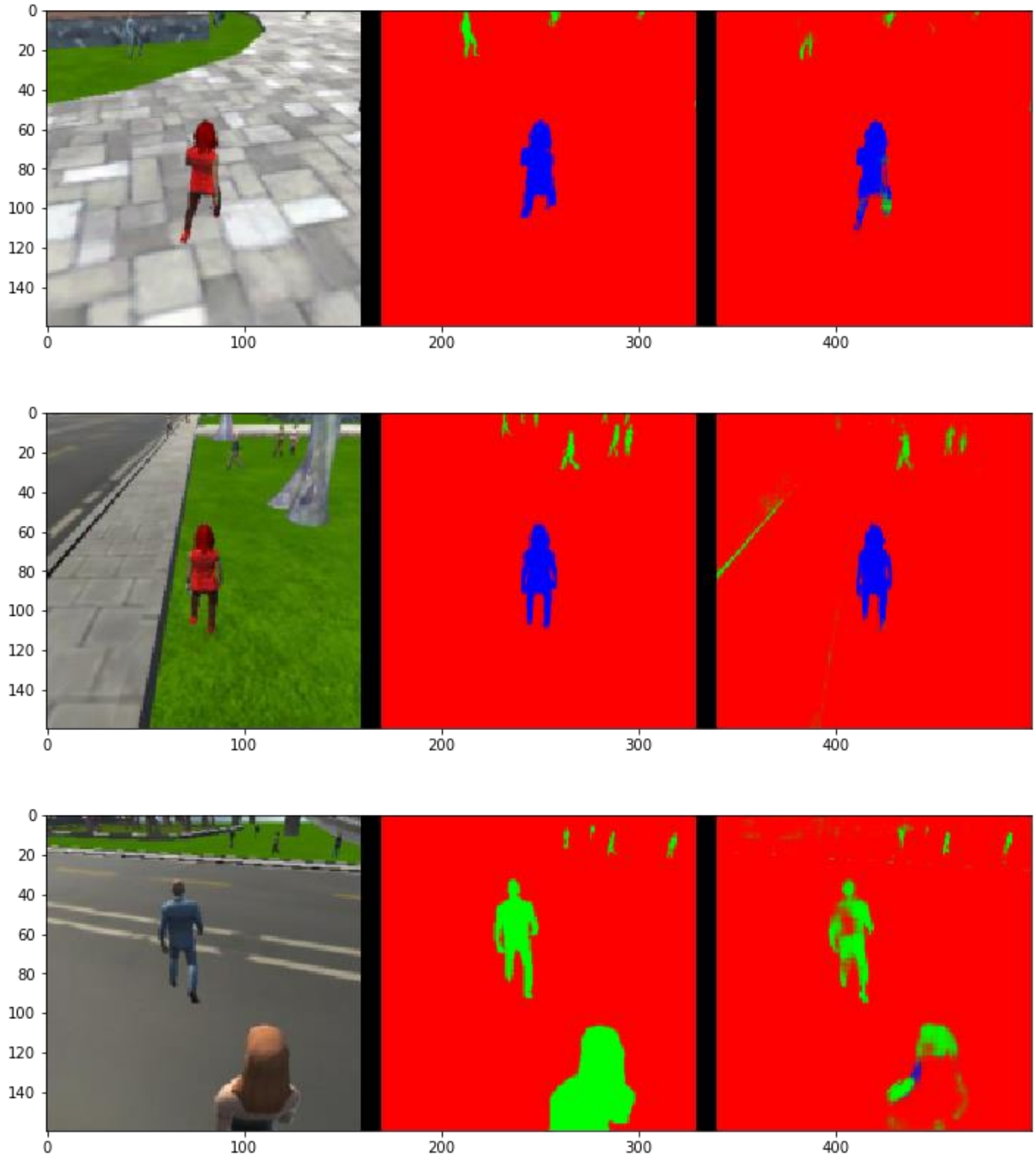
Training

The model was trained on AWS cloud services using px2.large GPU instance.

Performance

The final score of the model is 0.402 while the final IoU is 0.541.

Below are the test images for better visualization about the performance of the model. The left frame is the raw images, the middle frame is the ground truth and the right frame the model prediction. The goal is to have right frame to be as close as possible as to the middle frame. If the target is identified, he is shaded blue while non-targets are shaded green. When the target is at close range the IoU is 0.863, which is really good.



Future Enhancement

This model is particularly trained to identify persons, it could be improved for other object identification such as animals, roads, cars and etc.