Robotics ND - Follow Me Project

Updated Nov 25, 2017

Project Objective

The objective of the project is to buld a deep learning based target tracking system. This serves as a base for autonomus capabilities for Drones and Self driving cars in the days to come.
The Deep learning algorithm implemented is a fully convolutional neural network that lables each pixe in the frame with the classes of interest or background class aka Semantic segmentation. Semantic see gmentation algorithms are more relavant for describing the scene as against the convetional convolutional neural networks that are capable of image classification and object detection.

Architeture Details

The Semantic Segmentation algorithm used for this project is based on the technique called **Fully convolutuonal neural networks (FCNN)** that takes arbitrary sized image input and produces same sized output with each pixel masked with corresponding label. The flexibility of FCNN comes from th fact that it has no fully connected layers there by not having the restriction of fixed input size. The key aspects of FCNNs are:

- Encoder block
- Decoder block
-

Encoder block

- The encoder block resembles usual convolutional neural networks but without a fully connected layer at the end of the network. The idea of encoder network is to generate the latent representatio of the input image that can be later transformed into desired representation. Specific to the current project, I have used Separable 2D convolution instead of conventional Convolution layers. The advantage Separable conv layer has is that it acts on each input channel separately there by capturing the depth specific features along with the spatial characteristics. This is analogus to the inception module proposed in GoogLeNet
- To take advantage of transfer learning, various benchmark networks like GoogLeNet, ResNet can used in the encoder block, this helps the network in faster convergence
- The separable conv 2d definition is shown below, this is the basic building block of the encoder layer in the FCNN architecture built for Follow Me project.
- The batch normalization on the output of separable conv 2d acts as a network regularizer

```
                            padding='same', activation='relu')(input_layer)

    output_layer = layers.BatchNormalization()(output_layer)
    return output_layer
```

- The encoder block using spatial convolution layer is shown below

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

    # TODO Create a separable convolution layer using the separable_conv2d_batchnorm() functio
    output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)
    return output_layer
```

Decoder block

- The decoder block is similar to encoder block except that the output of each layer is spatially large than the input and the final output is of the same spatial dimension as the input to encoder block
- A bilinear upsample layer is used to scale up the input from the encoder there by helping the network to project the input to higher spatial dimensions - this operation is also called transpose convolution
- Cancat layer in the decoder block takes the input from bilinear upsample layer and concatenates it with the corresponding layer in decoder block
    - The skip connections that helps network to combine coarse, high layer information with fine, low layer information can be created using the concat layer. However, skip connections are no used in the current network, adding them can further improve the IoU
- The Separable 2d conv layers as stated in encoder block are used to generate the final projection output of the decoder block
- The upsampling and decoder blocks are shown below
- Two Separable 2d conv layers are used in decoder block that has helped with faster convergence

```
def bilinear_upsample(input_layer):
    output_layer = BilinearUpSampling2D((2,2))(input_layer)
    return output_layer
```

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

    # TODO Upsample the small input layer using the bilinear_upsample() function.

    # TODO Concatenate the upsampled and large input layers using layers.concatenate

    # TODO Add some number of separable convolution layers
    upsampleLayer = bilinear_upsample(small_ip_layer)
    concatLayer = layers.concatenate([upsampleLayer, large_ip_layer])
    output_layer1 = separable_conv2d_batchnorm(concatLayer, filters)
    output_layer = separable_conv2d_batchnorm(output_layer1, filters)
```

```
    return output_layer
```

Network Architecture

- The FCN architecture used for this is a stack of above stated encoder and decoder blocks with a 1X convolution layer between them
- 1X1 layer helps to preserve the spatial information of the tensor as compared to fully connected layer

```
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.
    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    encode1 = encoder_block(input_layer=inputs, filters=32, strides=2)
    encode2 = encoder_block(input_layer=encode1, filters=64, strides=2)
    encode3 = encoder_block(input_layer=encode2, filters=128, strides=2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv_bnorm = conv2d_batchnorm(input_layer=encode3, filters=256, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decode1 = decoder_block(small_ip_layer=conv_bnorm, large_ip_layer=encode2, filters=128)
    decode2 = decoder_block(small_ip_layer=decode1, large_ip_layer=encode1, filters=64)
    decode3 = decoder_block(small_ip_layer=decode2, 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_
ock()
    return layers.Conv2D(num_classes, 3, activation='softmax', padding='same')(decode3)
```

- A network with three encoder blocks stacked followed by a single 1X1 convolution layer followe by three decoder blocks as shown above is used
- Various combinations of encoder decoder blocks with varying filter sizes are tried and concluded above architecture which has given >40% aggregated IoU

Hyper-parameter tuning

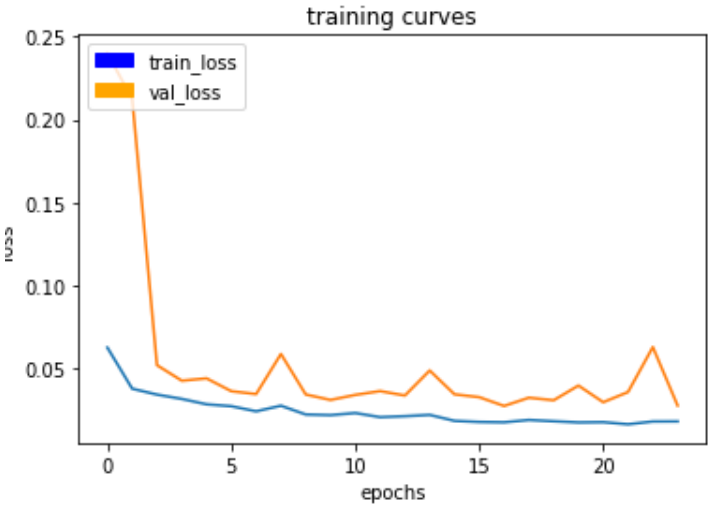
Several combinations of hyper-parameters are experimented for achieving acceptable IoU score. The details of individual hyper-parameters are discussed below

- **Learning Rate**
  - Learning rate is key to network learning - higher learning rates can lead to faster convergence but overfitting, lower learning rates delays the convergence but is less prone to overfitting.
  - Learning rates of 0.01 and 0.001 are tried with varying number of epoch and finalized using **0.01**
- **Batch Size**
  - Batch size represents the number of training samples that can be fed to the model in one step
  - This has minimal impact on performance of the network as compared to learning rate
  - Depending on the batch size time per epoch/step changes
  - Batch size of **100** is used for above network
- **Epochs**
  - Represents the number of time the training samples can be passed through the network during

training
    - Number of epochs plays a key role in learning
    - Lower number of epochs means the network cannot generalize well and the higher number of epochs makes the network prone to over fitting
    - A careful choice of number of epochs along with the choice of learning rate is needed to ensu convergence and generalization
    - Have trained the network for **30** epochs, however the model weights at the end of 24th epoch are used for achieving the desired results
- **Steps per epoch**
    - Represents the number of times the 'batch size' samples to be passed through the network to complete an epoch
    - **50** steps per epoch has been used
- **Validation steps**
    - Represents the number of batches of validation images that go through the network after each training epoch
    - 50 validation steps is used

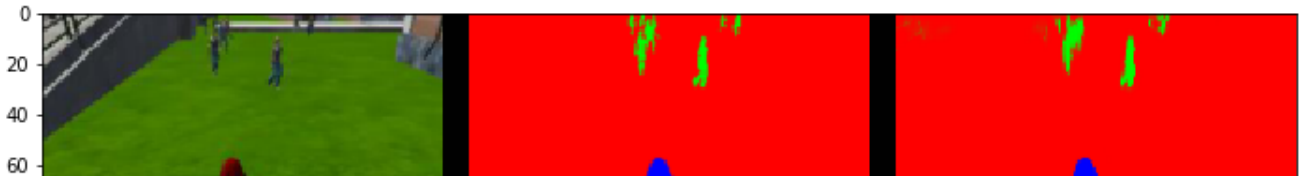
Results

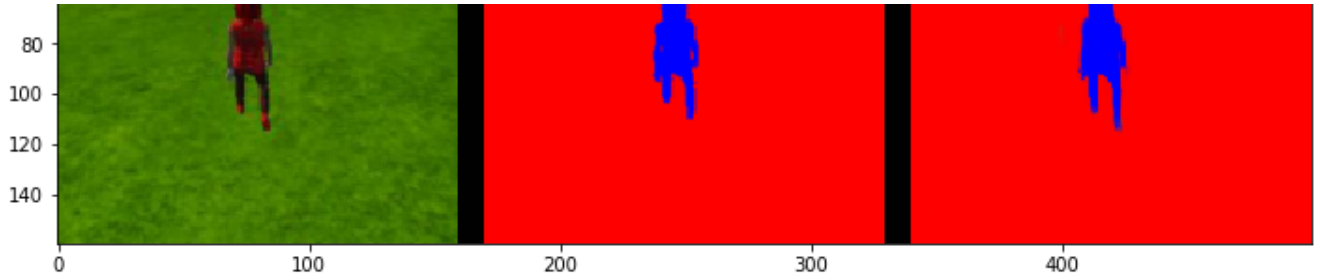
- The FCNN network built here is very compute intensive given that each pixel has to be classified
- Care has been taken to ensure the smallest possible network is used to achieve desired results
- The loss at the end of 24th epoch is shown below
- The final grade score achieved: 0.42187

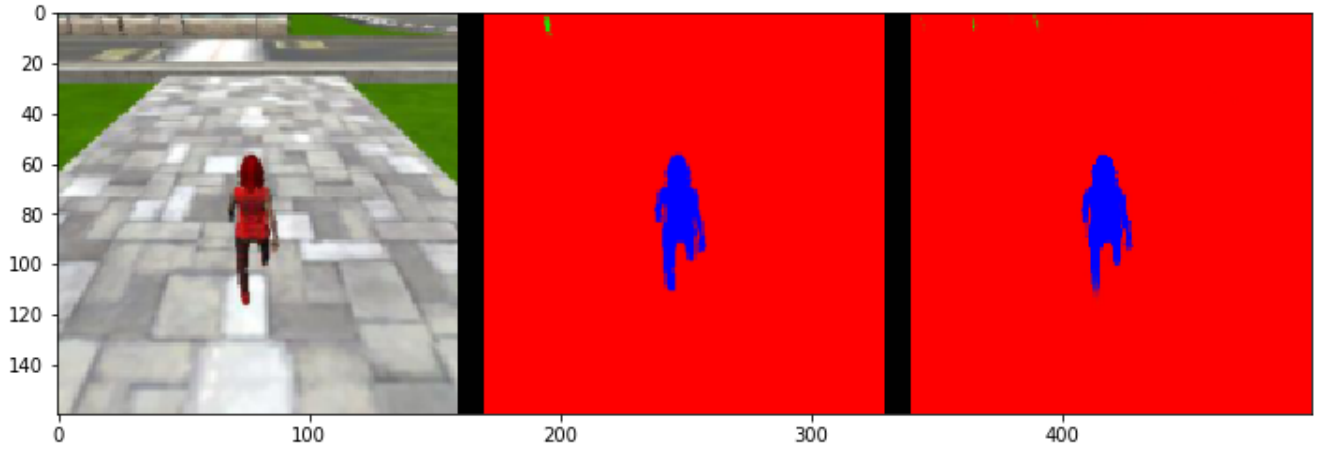


Model performance

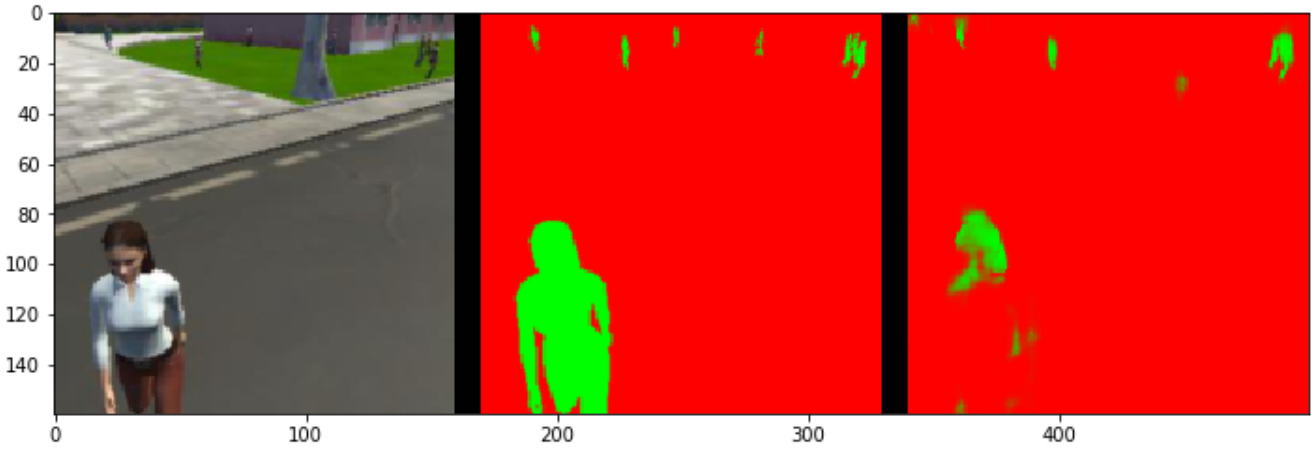
- Results while following the target

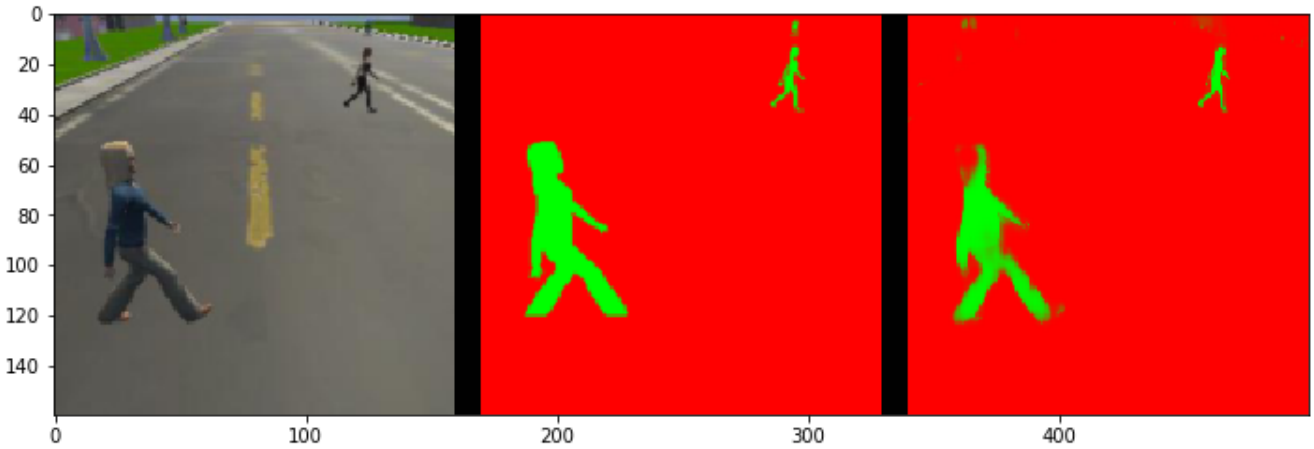




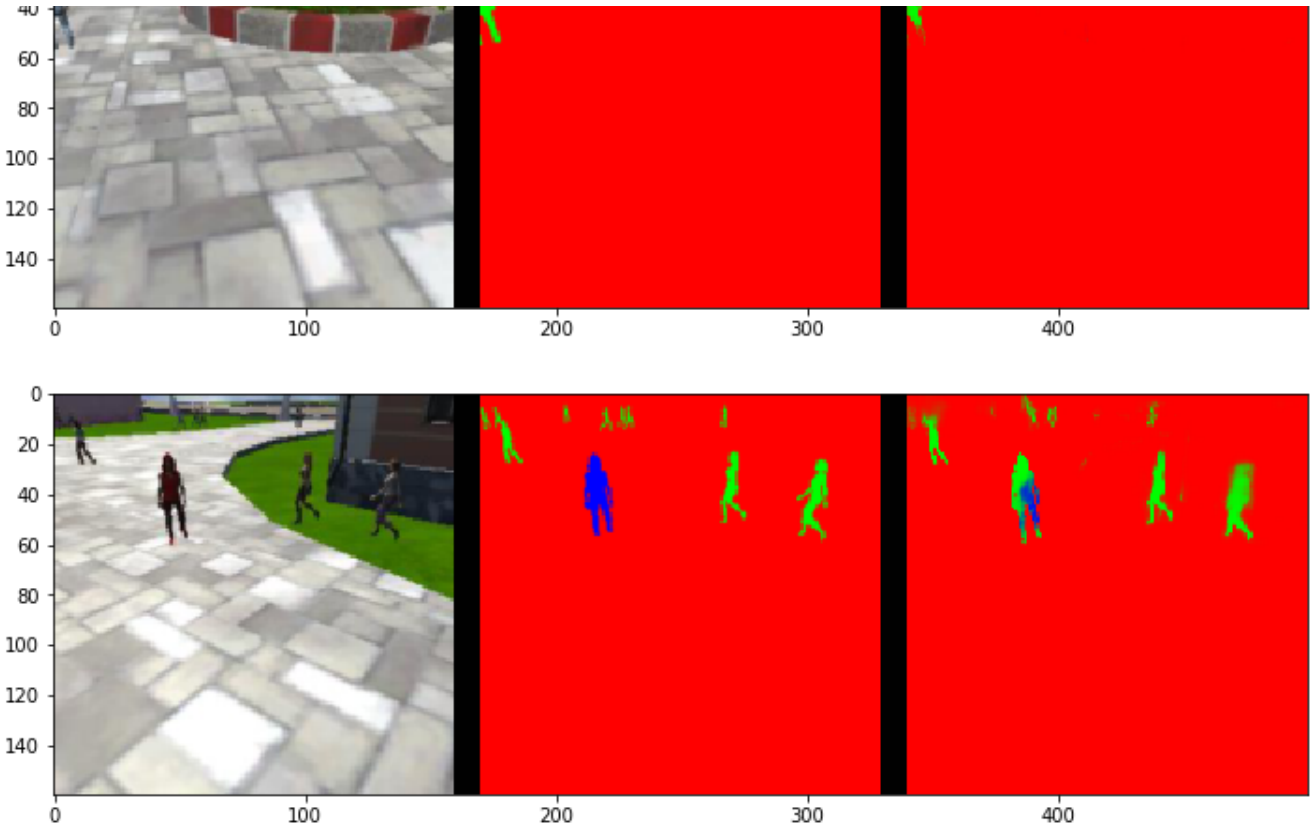


- Results while at patrol without target





- Results while at patrol with target



Further improvements

- Additional data for training can help improve the model performance
- Using pre-trained networks for transfer learning can help improve the performance. GoogLeNet, ResNet can be considered for the same
- Skip connections can help improving the performance of the model