Robotic Nanodegree
Project: Follow Me

Manuel Huertas López

December, 1, 2017.

Introduction

In this project a drone in simulation is flyed. The goal is to locate and follow a moving target, a woman with a red dress. Individual camera frames coming from a front-facing camera on the drone are analysed. Each pixel of each frame is classified using a fully convolutional neural network; the result of the network pipeline is used to follow the target.

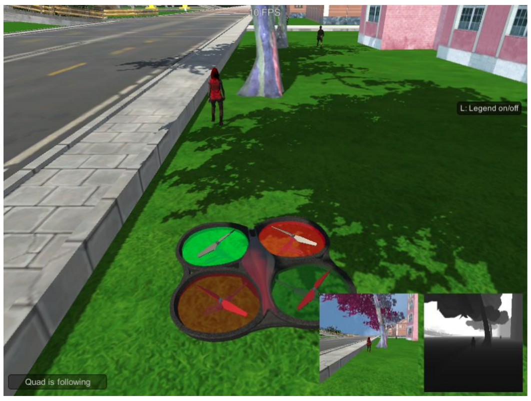


Figure 1*: A view of the drone and target in the simulation environment.*

Data Collection

A training and validation dataset have been provided, with 4131 and 1184 images respectively. Each image have got a label image associated to it with the classification for each pixel in three different classes: hero, anybody else and background.

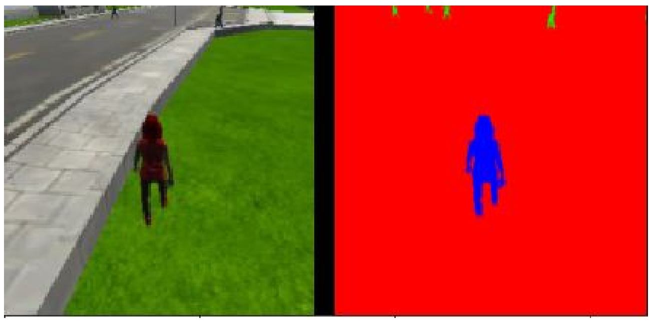


Figure 2*: On the left the image with a hero, on the right the same image segmented.*

It must be notice that these labeled images are specific for out current project. In case we would like to extend the functionality, to detect a dog or a cat for example, the label images must contain the classification for each pixel for this new features.

The training set was increased, in order to successfully train the neural network, by creating a new set of images. These new images were obtained flipping the original ones. The final total number of training images were 8262.

Network Architecture

A Semantic Segmentation Network was built to run a real time pixel-wise classification on the images coming from the camera in the drone.

Rather than building a convolution neural network (CNN), a series of convolution layers followed by a fully connected layer and finally a softmax activation function, a fully convolutional neural network (FCN) has been built because it is necessary to preserve the spatial information, where the hero is, rather than identify if the hero is in the image or not.

The FCN is comprised of two parts: encode and decoder.

The encoder is a series of convolutional layers; in each layer the network goes deeper in order to extract the features from the image. In the architecture selected the encoder part contains three layers with a filter size of: 32, 64, 128 respectibitilly. Each convolutional layer is created using a separable convolution with kernel size of 3 and stride of 2, following by a relu function to add non-linearity and a batch normalization to help to train the network.

The decoder layer was built by adding a 1x1 convolutional layer, followed by three Bilinear upsample layers; a technique that utilizes the weighted average of four nearest known pixels, located diagonally to a given pixel, to estimate a new pixel intensity value. In order to add information from the previous layers to the decode layer a concatenation is done. Finally, every decode layer contains a Separable Convolutional Layer to extract some more spatial information.

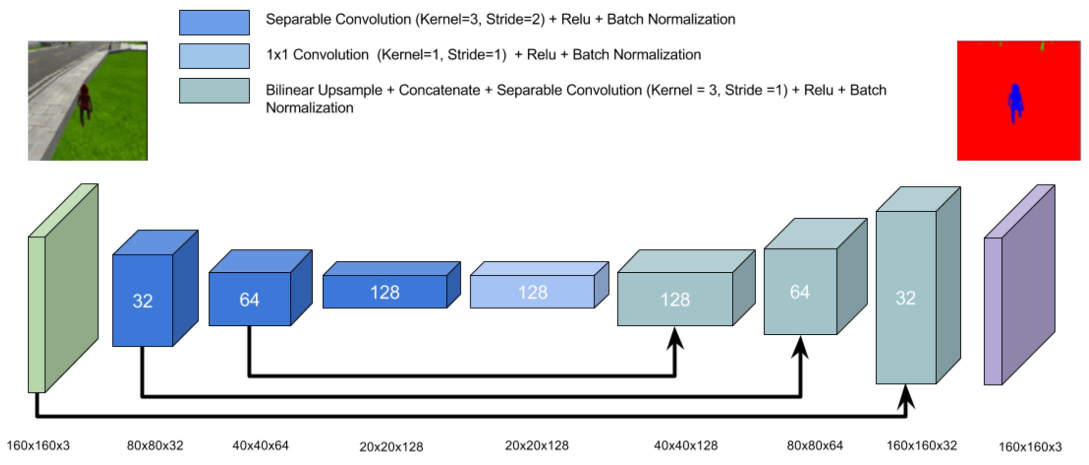

Separable Convolution (Kernel=3, Stride=2) + Relu + Batch Normalization
1x1 Convolution (Kernel=1, Stride=1) + Relu + Batch Normalization
Bilinear Upsample + Concatenate + Separable Convolution (Kernel = 3, Stride =1) + Relu + Batch Normalization
32
64
128
128
128
64
32
160x160x3
80x80x32
40x40x64
20x20x128
20x20x128
40x40x128
80x80x64
160x160x32
160x160x3

***Figure 3**: Fully Convolutional Network Architecture.*

```
def separable_conv2d_batchnorm(input_layer, filters, strides=1):
    output_layer = SeparableConv2DKeras(filters=filters,kernel_size=3, strides=strides,
                         padding='same', activation='relu')(input_layer)

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

def conv2d_batchnorm(input_layer, filters, kernel_size=3, strides=1):
    output_layer = layers.Conv2D(filters=filters, kernel_size=kernel_size, strides=strides,
                   padding='same', activation='relu')(input_layer)

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

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

***Code 1**: Helper function to create the layers.*

```
def encoder_block(input_layer, filters, strides):
    output_layer = separable_conv2d_batchnorm(input_layer, filters, strides=strides)

    return output_layer

def decoder_block(small_ip_layer, large_ip_layer, filters):
    upsampled_layer = bilinear_upsample(small_ip_layer)
    output_layer = keras.layers.concatenate(inputs=[upsampled_layer,large_ip_layer],axis=-1)

    output_layer = separable_conv2d_batchnorm(output_layer, filters, strides=1)
    output_layer = separable_conv2d_batchnorm(output_layer, filters, strides=1)

    return output_laye
```

***Code 2**: Encoder Block (Separable Convolutional Layer), Decoder Block (UpSample, Concatenate and Separable Convolutional Layer)*

```
def fcn_model(inputs, num_classes):

    #Encoder Blocks.
    layer_encoder_1 = encoder_block(inputs, filters=32, strides=2)
    layer_encoder_2 = encoder_block(layer_encoder_1, filters=64, strides=2)
    layer_encoder_3 = encoder_block(layer_encoder_2, filters=128, strides=2)

    #1x1 Convolution
    layer_1_x_1 = conv2d_batchnorm(layer_encoder_3, filters=128, kernel_size=1, strides=1)

    #Decoder Blocks
    layer_decoder_1 = decoder_block(layer_1_x_1, layer_encoder_2, filters=128)
    layer_decoder_2 = decoder_block(layer_decoder_1, layer_encoder_1, filters=64)
    layer_decoder_3 = decoder_block(layer_decoder_2, 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')(layer_decoder_3)
```

***Code 3**: FCN: Encoder, 1x1, Decoder.*

Training

In order to train the network is was necessary to use a computer with a GPU. The Amazon Web Service provided as part of the course was used. The training process took approximately 2 hours each run. It was a process of refining the hyperparameters until the result was satisfactory. The algorithm used to train the network was Adam, an extension to the classic Stochastic Gradient Descent algorithm. Few parameters must be set in order to obtain a correct solution.

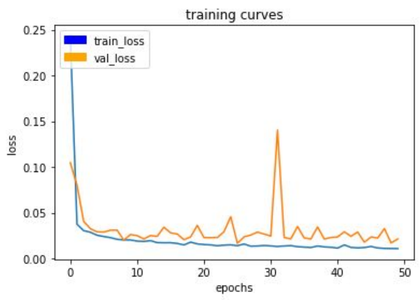
- **Number of epoch:** one epoch is a single forward and backward pass of the whole dataset. A final number of 50 epochs was selected with a final loss value of 0.0107.

- **Batch Size & Step Per Epoch:** because it could be computer inefficiency to compute the loss using every input from the train set, a small set of randomly imagen was used in every step of the Gradient Descent. A batch size of 32 and 256 step per epoch has been used because we have a total of 8262 images. This means that 32 images will be selected to calculate the loss, updating the weight and repeating the process 256 times.

- **Validation steps:** since the validation set contains 1184 images, a validation step of 36 was selected.

- **Learning Rate:** The learning rate selected was 0.002. The learning rate is the factor we apply to the correction, based on the gradient descent in order to minimize the loss. It seems that using a low learning rate help the gradient descent algorithm to find the correct solution.

The following diagram show the evolution of the loss and validation loss over the epochs.



```
256/256 [==============================] - 141s - loss: 0.0107 - val_loss: 0.0212
```

Figure 4: *Fully Convolutional Network Architecture.*

Prediction

After training the model the prediction on the validation data set are done. The prediction will be compared with the mask images, in order to check how well the model is doing. It can be seen from the above image that the model is doing quite well.

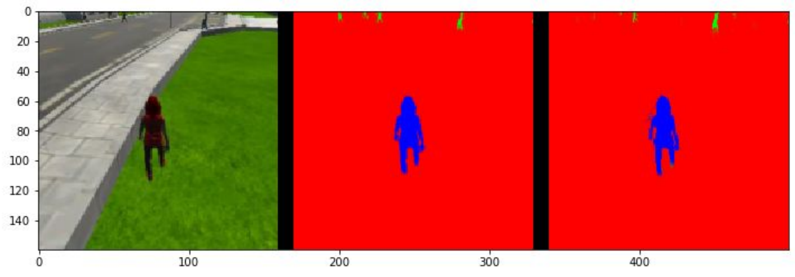


Figure 4: *Images following the target; left image, middle mask, right prediction.*

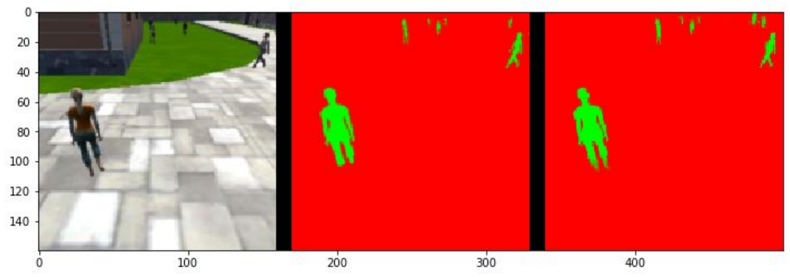


Figure 5: *Patrol without target; left image, middle mask, right prediction.*

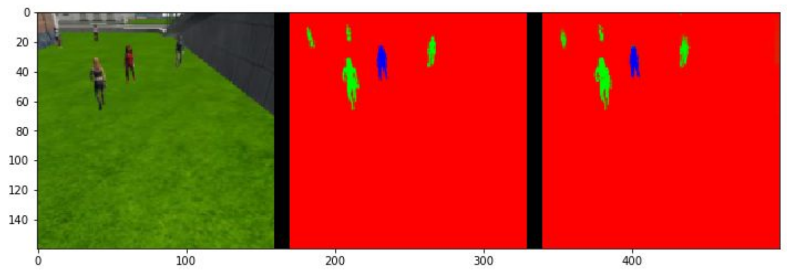


Figure 6: *Patrol witht target; left image, middle mask, right prediction.*

Evaluation

The final step is to measure how well the model is doing for the total of validation set. The intersection over union score is calculated for the three classes and under three different scenarios.

A. **Patrol with target:** test how well the hero can be detected from a distance.
B. **Patrol without target:** test how often the network makes a mistake and identified the wrong person as target.
C. **Following images:** test how well the network can identify the target while following them.

Patrol with target:

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9968644413026413
average intersection over union for other people is 0.485854374027251
average intersection over union for the hero is 0.21808863806275428
number true positives: 117, number false positives: 1, number false negatives: 184
```

Patrol without target:

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9880196239293199
average intersection over union for other people is 0.7643979803039603
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 33, number false negatives: 0
```

Following:

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9963000689930401
average intersection over union for other people is 0.4112369942674416
average intersection over union for the hero is 0.9271808112211917
number true positives: 539, number false positives: 0, number false negatives: 0
```

The final grade score is 42.9 %

Simulator

After training the network the simulator was used to check how well the drone can follow the target. In the following figure can be seen, on the right top, what the drone camera is seeing and how the image was segmented using the neural network to identify the target.

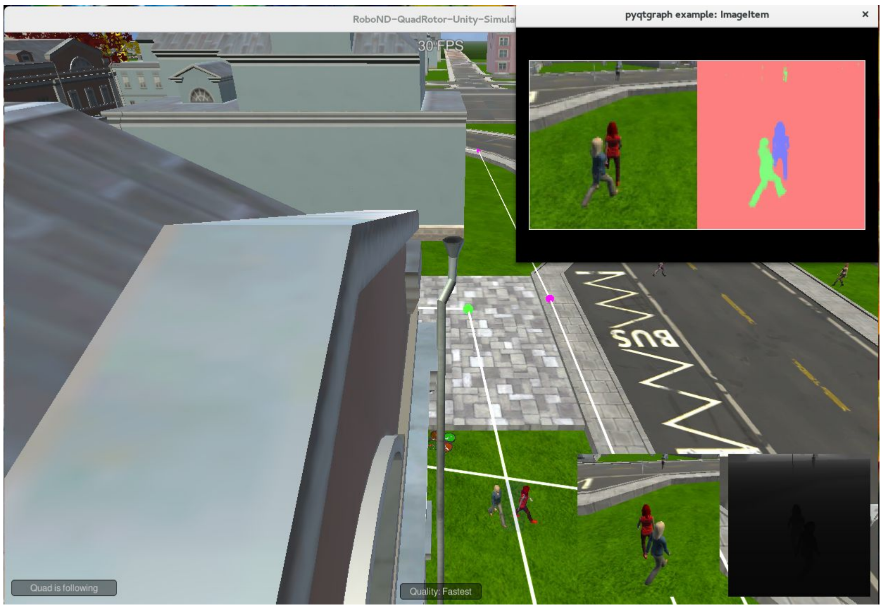


***Figure 7**: The drone is following the hero.*

Future Enhancements

There are several things that can be done in order to improve the performance of the network:

1. **A better training set:** generate a training set with more images and at a higher resolution can improve the train of the network.
2. **A neural network architecture going deeper:** in order to go deeper in the image classification when the hero is far way, a deeper architecture can help, adding more layers with deeper filters.