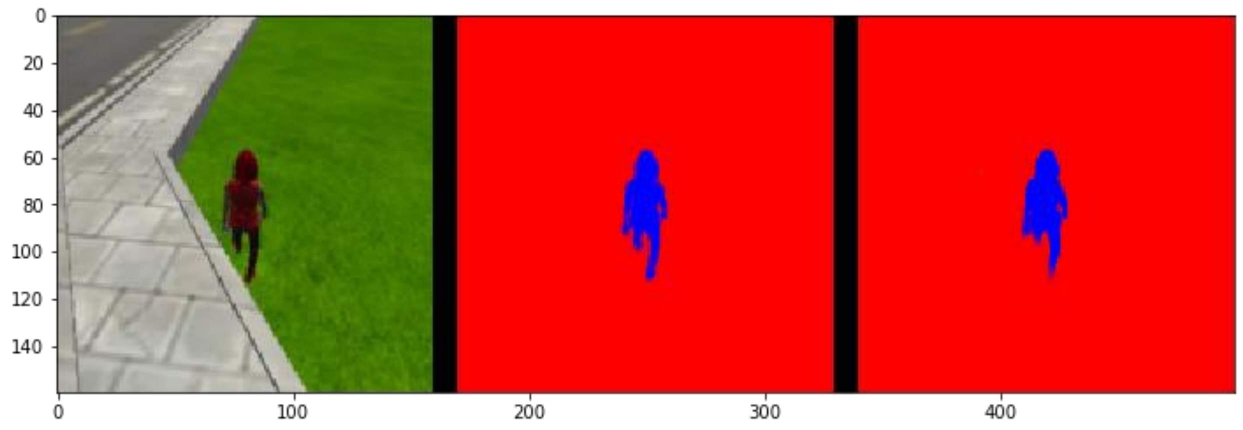
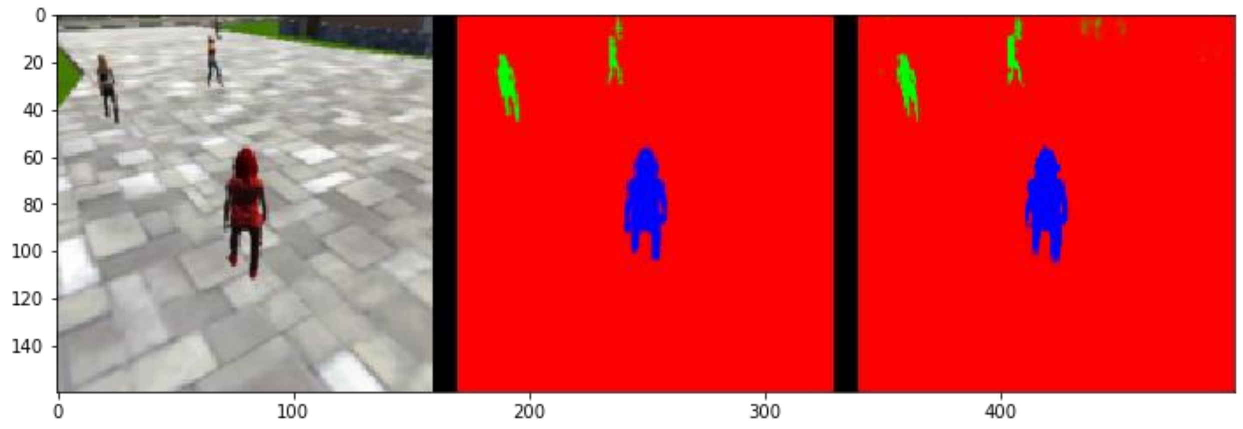
Project4: Follow Me Project, Udacity Robotics Nanodegree

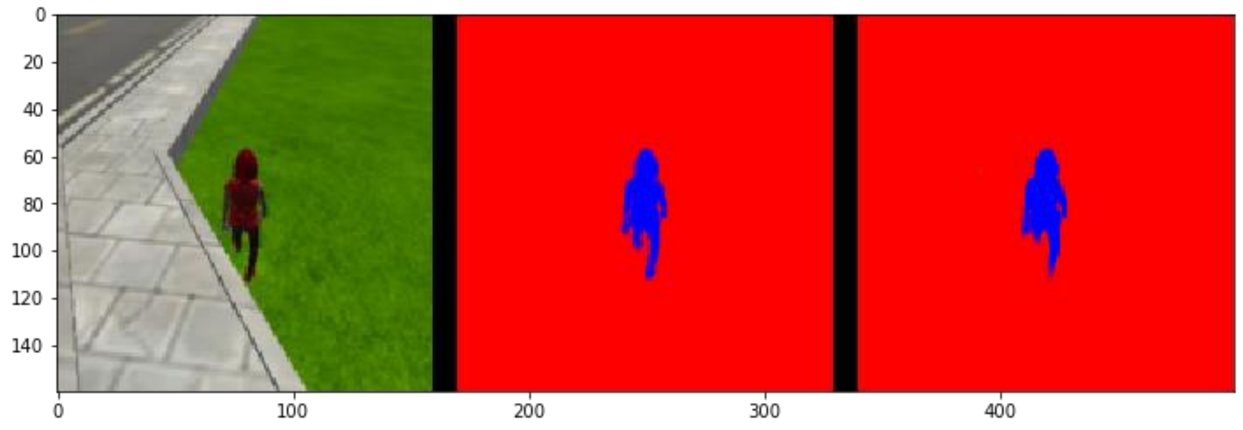
Bahadır Özkan, December 2017

Introduction

A FCN (Fully Convolutional Neural Network is used in this project to identify people in the given data set and identify a “hero” person for the drone to follow. The dataset (training and validation sets) provided by the Udacity was used as is. More on this in the Future Enhancements section.



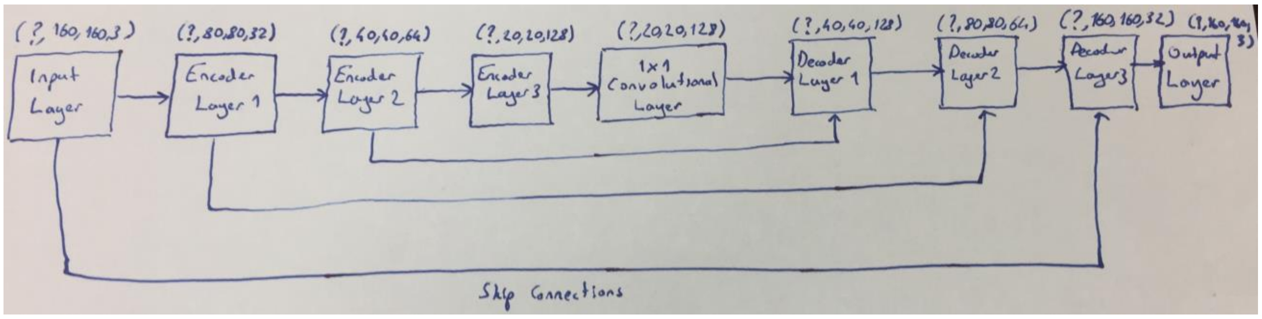




An example identification of the “hero” can be seen above. Semantic Segmentation technique is used to identify the target. Semantic Segmentation is advantageous in scene understanding where different decoders can be trained to learn various features. The network architecture will be addressed in the next section. Semantic Segmentation can be slower to compute versus the bounding boxes technique which may be a downside according to the task or your hardware. In this project AWS Udacity Robotics Laboratory Community AMI was used to train the network which decreased the training time dramatically.

Network Architecture

FCN network that is implemented for this task has 3 encoder and 3 decoder blocks connected with a 1x1 convolution layer as shown below. Encoder/decoder architecture is especially good when it is needed to preserve the spatial information. The encoding section learns from the input while the spatial part gradually decreases whereas in decoding section this spatial information is being recovered. If this was a classification task encoder part would be sufficient. One downside of FCN (encoder/decoder architecture) can be stated as the speed since without decoder part the algorithm will run much faster.



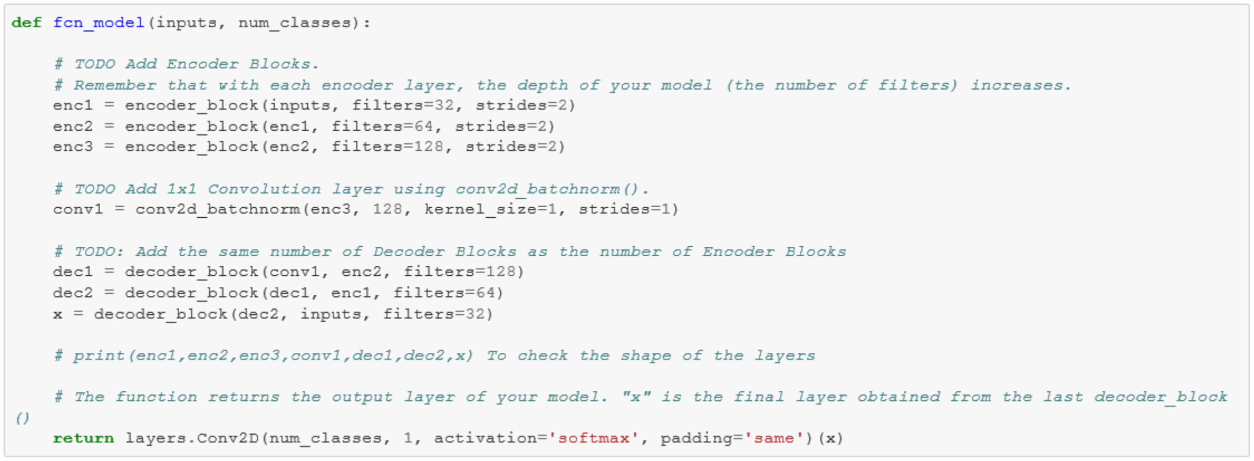
Skip connections are connecting input layer with 3rd decoding layer, 1st encoding layer with 2nd decoding layer and 2nd encoding layer with 1st decoding layer. Skip connections help us to retain the information where could be lost throughout the encoding operation. Skip connections connect two non-adjacent layers and help the network to make more precise segmentation decisions.

Spatial information is preserved by using the above architecture as opposed to simple convolutional networks. Spatial information is needed in complex tasks like scene understanding.

Encoder sections learn from the input. This operation involves pooling which is followed by application of a RELU function. Built-in Keras models are used to implement encoding and 1x1 convolution.

Decoder block comprised of bilinear upsampling , concatenation and a convolutional parts. Advantage of 1x1 convolutions against fully connected layer is instead of flattening the layer from 4D to 2D the spatial information is preserved by sweeping the all layer and keeping the 4D shape.

As it can be seen in the code below, filter size was chosen to be 32, 64 and 128 respectively for the encoding blocks 1, 2 and 3 and the opposite for decoding blocks to complete the fully connected architecture with the help of 1x1 convolutional layer with the same filter shape as encoder 3 and decoder 1 layers.



```
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.
    enc1 = encoder_block(inputs, filters=32, strides=2)
    enc2 = encoder_block(enc1, filters=64, strides=2)
    enc3 = encoder_block(enc2, filters=128, strides=2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv1 = conv2d_batchnorm(enc3, 128, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    dec1 = decoder_block(conv1, enc2, filters=128)
    dec2 = decoder_block(dec1, enc1, filters=64)
    x = decoder_block(dec2, inputs, filters=32)

    # print(enc1,enc2,enc3,conv1,dec1,dec2,x) To check the shape of the layers

    # 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')(x)
```

Hyper Parameter Selection

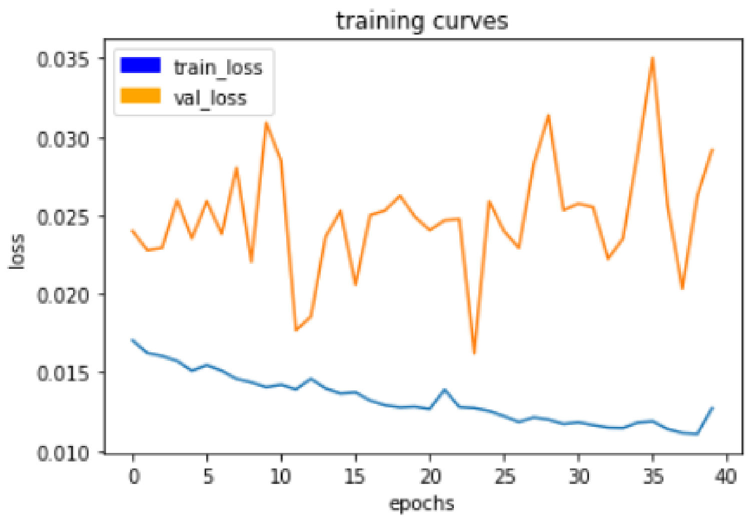
Batch number selection depends on the task and needs to be correlated with the learning rate. Batch size was chosen 64, a traditionally common number for this project. A larger size could be selected until you see the effects of overfitting and performance degradation however 64 worked and no other number was tested.

Learning rate is another critical parameter in deep learning and as it was mentioned above needs to be correlated with the batch number. Learning rate needs to be reduced if the gradients become unstable. Gradients can become unstable if the batch number is chosen very small. Learning rate was chosen to be 0.001 for this project. Before settling on this value higher learning rates were tried. First 0.1 and then 0.01 gave final scores around 30s.

Number of epochs affects how good your model is fit. A high number would cause overfitting whereas small number would cause underfitting. In this project number of epochs was chosen to be 40. 40 epochs gave better results than 20 keeping all other parameters unchanged.

Other hyper parameters are steps per epoch and validation steps were adjusted according to the batch size. Number of training and test images were divided by the batch size respectively for steps for epoch and validation steps. Dividing the number of test images to batch size was found to be around 18 but was rounded to 20. Finally, workers parameter was kept as 2 and since other parameters gave a satisfactory result there was no need to try other values for workers.

Resulting loss graph can be found below. The spikes seen on the validation loss are an unavoidable consequence of Mini-Batch Gradient Descent in Adam.



Discussion

Final score evaluated was 0.414899270736 with the hyper parameters described above. This model was trained to identify the target person “hero” so in order for this model to identify other objects (dog, cat, car, etc.) it must be trained with the images of the other objects also. If the problem was to distinguish dogs from cats, we would need to provide training, test and validation data for dogs and cats and train the network with the training images of dogs and cats. However, like mentioned in the network architecture section if this was a classification problem, we would use a simpler architecture.

Future Enhancements

There model can be improved in many ways but these are left to the reader. First, input images could be augmented and more data could be gathered from the simulator to improve the performance. More training data with the useful information for the goal would make the difference. There are several techniques for that. Examples are cropping various parts of an image and feeding them. Also flipping the images is another technique that can be used for data augmentation. Also, some images that do not contain people could be deleted for model to focus on person information only.

The architecture and hyper parameters can be modified to improve the performance. Deeper encoding/decoding or several branches of encoder/decıders could be used to learn different features.