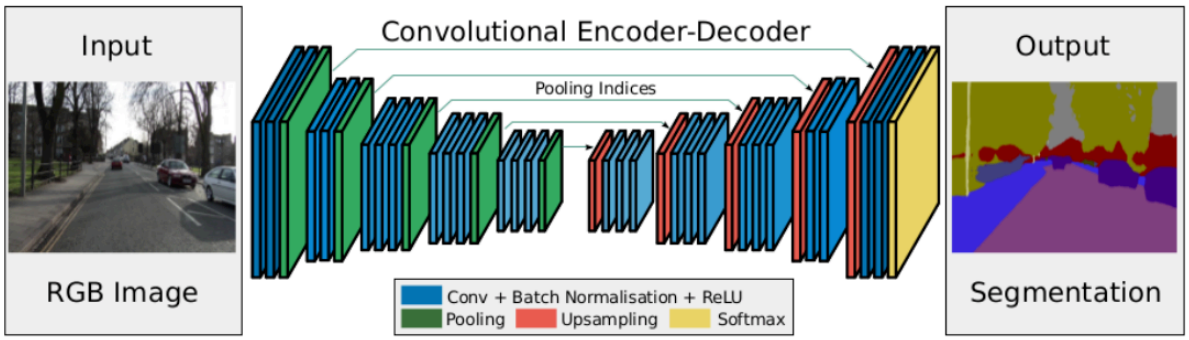
Table Of Contents

Introduction

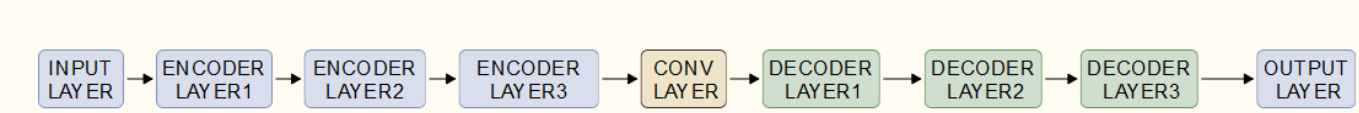
In this project I have used an FCN (fully-convolutional neural network) to paint pixels that is part of a person. FCN needs to recognize two types of person which are lead and random person. This kind of data would be very useful to create a follow me function for drones. The training images are from a 3D simulator and given from Udacity. I have used IOU - Intersection Over Union metric to measure performance of the model. I have Used Amazon EC2 for training.



Note: This model would not work on any dogs, cats, or cars, as a specific person who has a red outfit is used for training so the model recognize just this person from the 3D Simulation.

Network Architecture

I have used encoder-decoder architecture of Fully-Convolutional Neural Network . The encoder part reduces spatial dimension and decoder part gradually recovers spatial dimension. Encoding is a step that learns details about the image for classification. In this step, image gets down-sampled and generalized to reduce the risk of overfitting. The disadvantage of this step loss of details and spatial dimension. The decoder step solves these disadvantages by mapping the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification.



I used three encoder and decoder layers to obtain the target score and it worked. However, to increase the score, layer number would be increased even more.

It has the following features:

- **Encoder**
- **1x1 Convolutional**
- **Skip Connections**
- **Bilinear Interpolation Upsampling with Transposed convolutional layers in the decoder part**

```
def fcn_model(inputs, num_classes):

    flt = 32

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
    encoder_layer_1 = encoder_block(inputs, flt, 2)
    encoder_layer_2 = encoder_block(encoder_layer_1, flt*2 , 2)
    encoder_layer_3 = encoder_block(encoder_layer_2, flt*4 , 2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    mid_layer = conv2d_batchnorm(encoder_layer_3, flt*4, kernel_size = 1, strides = 1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decoder_layer_1 = decoder_block( mid_layer, encoder_layer_2, flt*4 )
    decoder_layer_2 = decoder_block( decoder_layer_1, encoder_layer_1, flt*2 )
    decoder_layer_3 = decoder_block( decoder_layer_2, inputs, flt )

    # The function returns the output layer of your model. "x" is the final layer obtained from the last decoder_block()
    outputs = layers.Conv2D(num_classes, 3, activation='softmax', padding='same')(decoder_layer_3)
```

```
Tensor("input_2:0", shape=(?, 160, 160, 3), dtype=float32)
Tensor("batch_normalization_8/batchnorm/add_1:0", shape=(?, 80, 80, 32), dtype=float32)
Tensor("batch_normalization_9/batchnorm/add_1:0", shape=(?, 40, 40, 64), dtype=float32)
Tensor("batch_normalization_10/batchnorm/add_1:0", shape=(?, 20, 20, 128), dtype=float32)
Tensor("batch_normalization_11/batchnorm/add_1:0", shape=(?, 20, 20, 128), dtype=float32)
Tensor("batch_normalization_13/batchnorm/add_1:0", shape=(?, 40, 40, 128), dtype=float32)
Tensor("batch_normalization_15/batchnorm/add_1:0", shape=(?, 80, 80, 64), dtype=float32)
Tensor("batch_normalization_17/batchnorm/add_1:0", shape=(?, 160, 160, 32), dtype=float32)
Tensor("conv2d_4/truediv:0", shape=(?, 160, 160, 3), dtype=float32)
```

NN Operators Batch normalization - normalizing layers of the neural net. Intuition is that given input normalization is valuable, we might also want to normalize information across NN layers. This generally avoids the need for dropout Skip Connections - combine data from different layers so as to combine information from multiple levels of granularity 1x1 Convolution - Instead of a fully connected layer that aggregates all neurons into a single output array, we preserve the spatial context via 1x1 convolution but instead shrink down to smaller layers.

Hyper Parameters and Other Design Choices

Learning Rate - I have used 0.002 as it was improved faster than 0.0002 and more stable than 0.2.

Batch Size - I have used 40. Generally, optimized via trial and error.

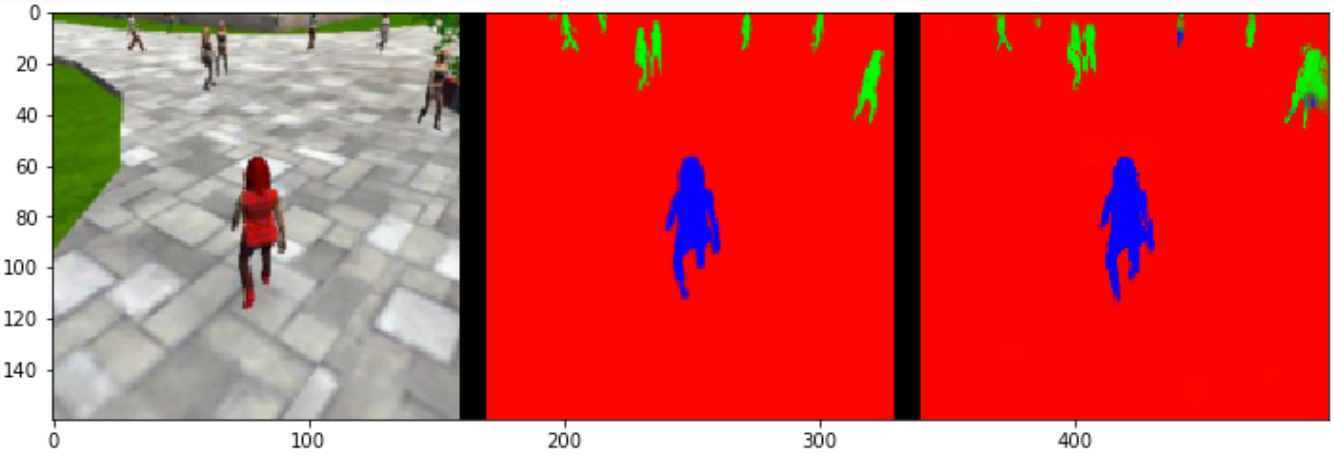
Optimizer - I have used Nadam Optimizer instead of Adam , it increased the score from 0.39 to 0.42.

Score

My final score is 0.421553640294

Future Enhancements

My model sometimes confuses parts of a random person and the lead person, this would be solved by increasing the training dataset.



For further improvements - Some of the students from Udacity has improved their model by flipping datasets (Data Augmenting) and deleting images that has no labelled objects inside. I will be trying in the future as well.

Generalizability

It is possible to update this model for recognizing other objects, but new datasets with labels has to be added.