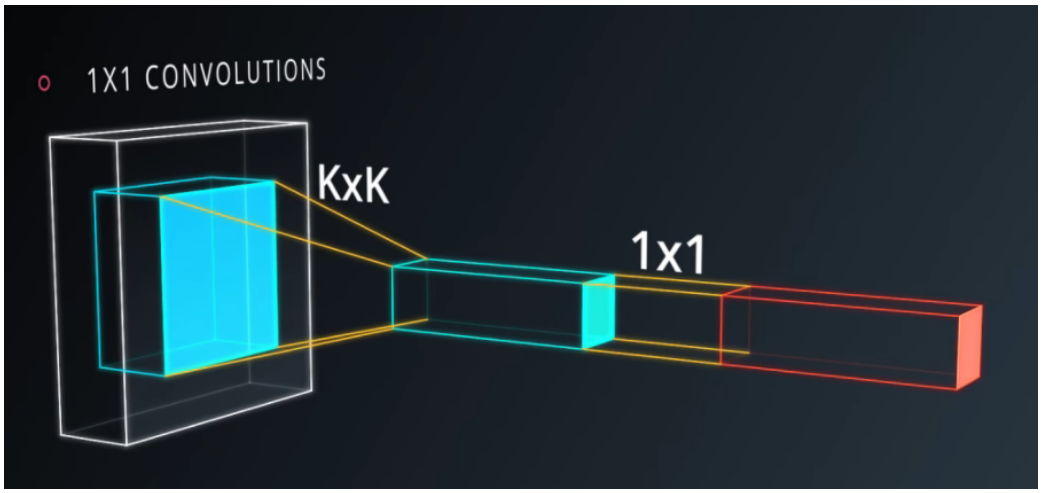
Follow Me : write-up

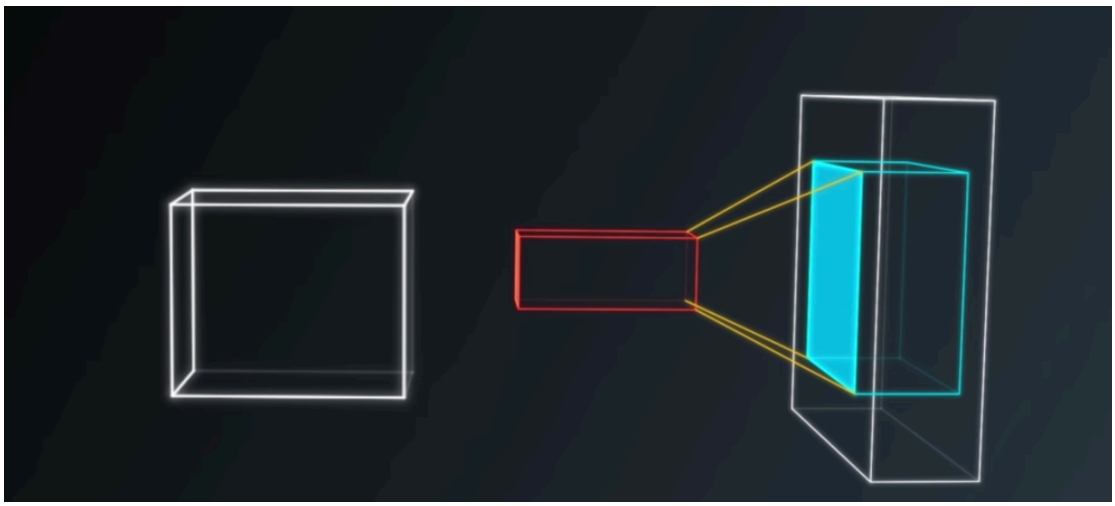
Zhanghanwei

Network architecture

In this project, I build a fully convolutional network (FCN's) with many different layers. This is a great architecture for a classification task. It can work on images of any size. Fundamentally don't care about the size of the input.

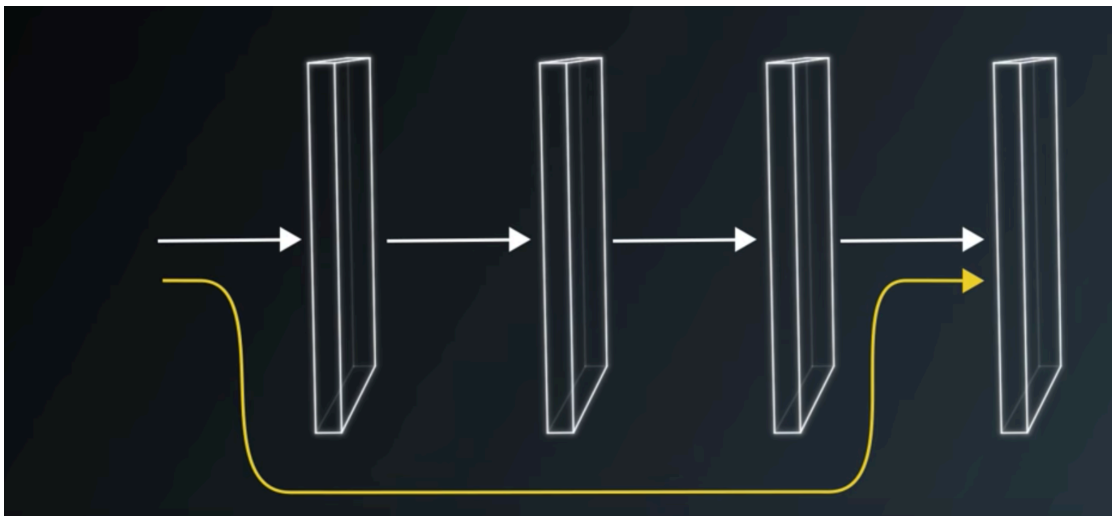


1. Replace fully connected layers with one by one convolutional layers. This will result in the output value remain 4D instead of flattening to 2D and spatial information will be preserved.

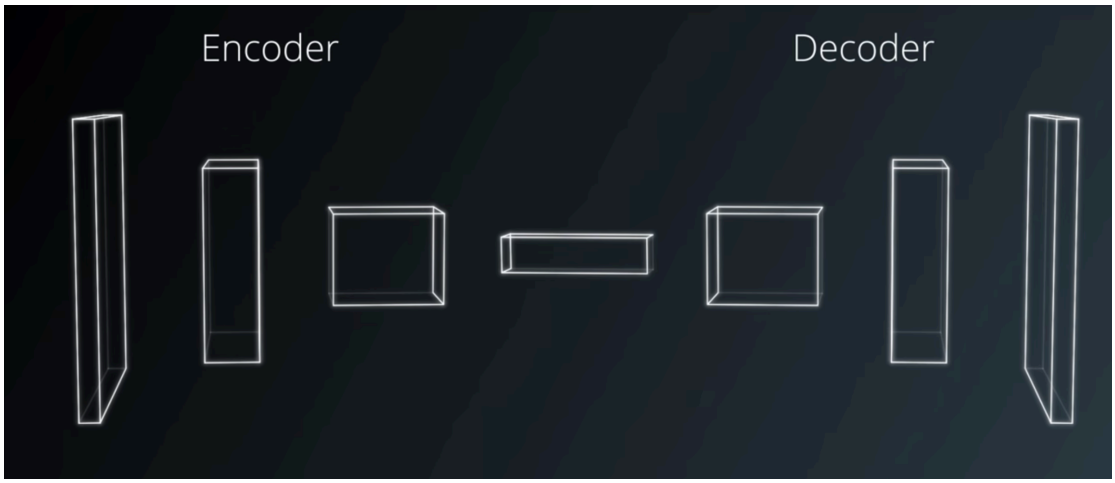


2. Up-sampling through the use of transposed convolutional layers. It is essentially a reverse convolution in which the forward and the back passes

are swapped. The property of differentiability is that retain the training is simply the same as previous neural networks.



3. Skip connections allow the network to use information from multiple resolution scales. As a result, the network is able to make more precise segmentation decisions.



4. Encoder and Decoder

The encoder is a series of convolutional layers, it can extract features from the image. The decoder up-scales the output of the encoder to the same size as the original image. The encoder is followed by the decoder and used a one by one convolutional layer at the middle. The model can add pre-trained model as encoder and only the third and the fourth polling layer used for skip connections.

5. My code for FCN'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.
    encoder1 = encoder_block(inputs, 16, 2)
    encoder2 = encoder_block(encoder1,64,2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv_layer11 = conv2d_batchnorm(encoder2, 256, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    decoder1 = decoder_block(conv_layer11, encoder1, 64)
    decoder2 = decoder_block(decoder1, inputs, 16)

    # 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')(decoder2)
```

I used encoder_block function to build two layers as encoder layers. And used con2d_batchnorm function as one by one convolutional layer. Then used decoder_block to build two skip connections layers to use information from input data and the first encoder layer.

And I try different layers architecture:

Encode1-Endoer2-OneByOne-Decoder1-Decoder2

a) 64-128-256-128-64

b) 16-32-128-32-16

c) 16-64-128-64-16

d) 16-64-256-64-16

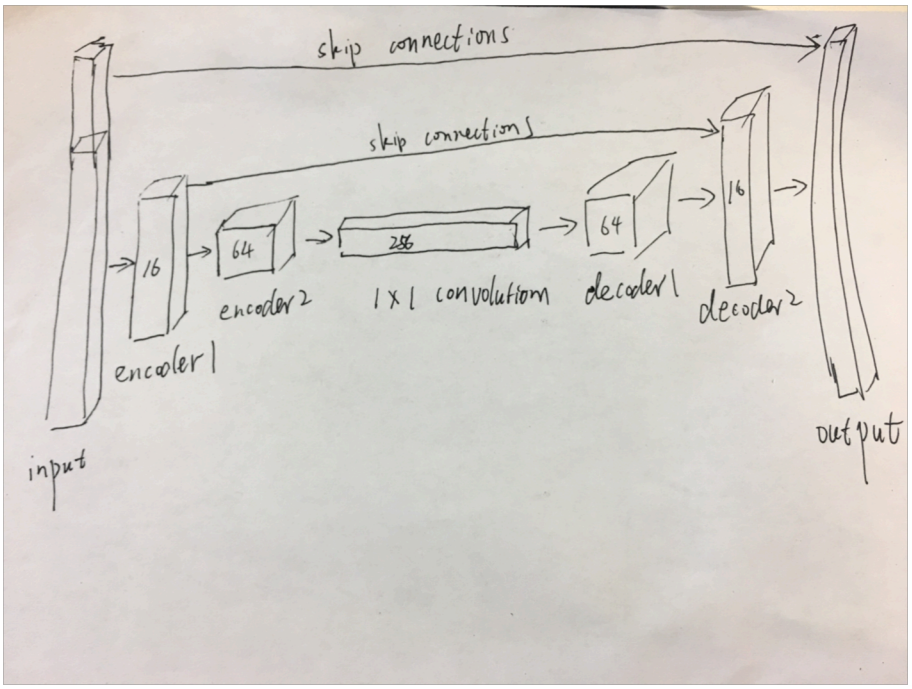
```
def encoder_block(input_layer, filters, strides):

    # TODO Create a separable convolution layer using the separable_conv2d_bat
    output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)
    return output_layer
```

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

    # TODO Upsample the small input layer using the bilinear_upsample() function.
    small_upsample = bilinear_upsample(small_ip_layer)
    # TODO Concatenate the upsampled and large input layers using layers.concatenat
    layer_concat = layers.concatenate([small_upsample,large_ip_layer])
    # TODO Add some number of separable convolution layers
    output_layer = separable_conv2d_batchnorm(layer_concat,filters)
    return output_layer
```

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



Parameter chosen

When I reduce the learning rate, the model accuracy increase when I have enough number of epochs, but when learning rate smaller than 0.01, I find it hard to train the model in 1 hour to get the best result.

When I increase the batch size, I find the validation loss decrease more than before per epochs, but it took more time to run per epochs and limited by the computer configuration. When batch size bigger than 100, it get some error when run training model.

When I increase number of epochs, the validation loss became smaller but it can not continue getting small when epochs bigger than 20, and more epochs need more time to train the model.

I try different depth of layers architecture and tuning parameter to a best result for each. There are some records:

parameters and result 1

- layers 64-128-256-128-64
- learning_rate = 0.01
- batch_size = 100
- num_epochs = 10
- steps_per_epoch = 200
- validation_steps = 50
- workers = 2

In [26]:
```
# Sum all the true positives, etc from the three datasets to get a weight for the score
true_pos = true_pos1 + true_pos2 + true_pos3
false_pos = false_pos1 + false_pos2 + false_pos3
false_neg = false_neg1 + false_neg2 + false_neg3

weight = true_pos/(true_pos+false_neg+false_pos)
print(weight)
```

```
0.7488207547169812
```

In [27]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.28304647795
```

In [28]:
```
# And the final grade score is
final_score = final_IoU * weight
print(final_score)
```

```
0.211951077238
```

parameters and result 2

- layers 16-62-128-64-16
- learning_rate = 0.01
- batch_size = 64
- num_epochs = 20
- steps_per_epoch = 200
- validation_steps = 50
- workers = 2

In [46]:
```
# Sum all the true positives, etc from the three datasets to get a weight for the score
true_pos = true_pos1 + true_pos2 + true_pos3
false_pos = false_pos1 + false_pos2 + false_pos3
false_neg = false_neg1 + false_neg2 + false_neg3

weight = true_pos/(true_pos+false_neg+false_pos)
print(weight)
```

```
0.7471131639722863
```

In [47]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.412901918469
```

In [48]:
```
# And the final grade score is
final_score = final_IoU * weight
print(final_score)
```

```
0.308484458718
```

parameters and result 3

- layers 16-64-256-64-16
- learning_rate = 0.01
- batch_size = 64
- num_epochs = 10
- steps_per_epoch = 200
- validation_steps = 50
- workers = 2

In [63]:
```
# Sum all the true positives, etc from the three datasets to get a weight for the score
true_pos = true_pos1 + true_pos2 + true_pos3
false_pos = false_pos1 + false_pos2 + false_pos3
false_neg = false_neg1 + false_neg2 + false_neg3

weight = true_pos/(true_pos+false_neg+false_pos)
print(weight)
```

```
0.718717683557394
```

In [64]:
```
# Sum all the true positives, etc from the three datasets to get a weight for the score
true_pos = true_pos1 + true_pos2 + true_pos3
false_pos = false_pos1 + false_pos2 + false_pos3
false_neg = false_neg1 + false_neg2 + false_neg3

weight = true_pos/(true_pos+false_neg+false_pos)
print(weight)
```

```
0.718717683557394
```

In [65]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.551385806941
```

In [66]:
```
weight_file_name = 'my_amazing_model.h5'
model_tools.save_network(model, weight_file_name)
```

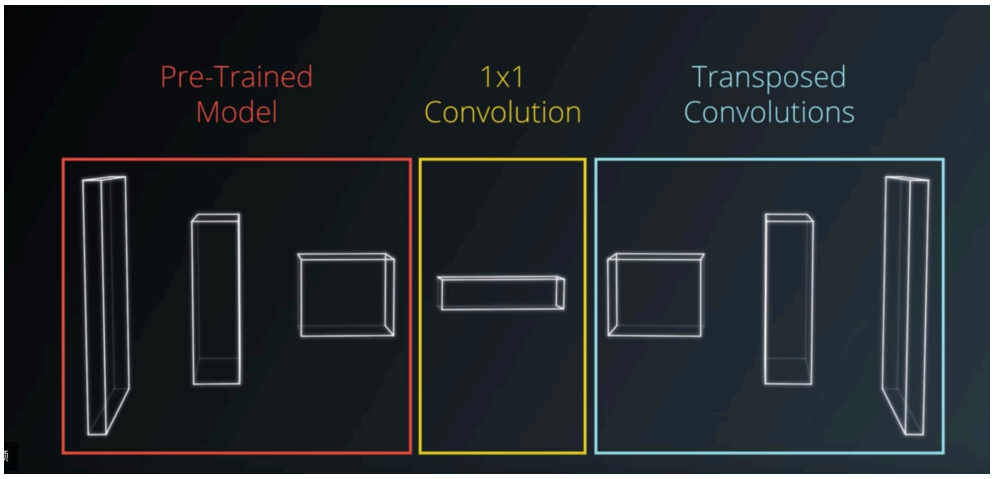
The values I selected and the best result I got 55.13%.

Future Enhancements

I think there are many things can further improve accuracy and model:

A more efficient environment that can spend less time and take more attempt to improve accuracy. I try to use another Instance Type "p3.2xlarge" for this project. But I find when I train the model, the GPU doesn't work(AMI is Udacity Robotics Deep Learning Laboratory). Maybe there are some different settings when using "p3.2xlarge" instead of "p2.xlarge ".

I think it's a good way to use pre-trained model as encoder and only the third and the fourth polling layer used for skip connections. That is similar to transfer learning, a pre-trained model can speed up training and improve the performance of the model.



Limitations to the neural network

I think the model I trained for this project can't work well for following another object (dog, cat, car, etc.).

1. The model training data only contain humans, more data contain other object would be required.
2. The network architecture and parameters that work well in this project maybe not work well when include more another object. We should distinguish different objects and find where they are. I think a more complex network architecture would be required.
3. And I think new evaluation would be required. Include the AUC of recognize different objects in addition and IoU for the dataset as before.

Environment instance

AWS - EC2

AMI: Udacity Robotics Deep Learning Laboratory

p2.xlarge instance