Udacity RoboND Deep Learning Project: Follow Me

Brett Gleason

Scope

The scope of this project is to train a fully convolutional neural network so that a drone can recognize a specific person and follow them in simulation. The rubric for this project can be found here:

https://review.udacity.com/#!/rubrics/1155/view

Intro

In previous lessons, neural networks were trained to recognize images of letters. Is this a picture of an A? B? etc. Next the idea of a convolutional neural network (CNN) was introduced. Rather than applying the model to the entire image at once, a CNN will apply the model to a small window of the full image, sliding this window across the entire image. The advantage of a CNN over a regular deep neural network is that a CNN can be trained to recognize 'Is this a picture of an A' with the A appearing anywhere in the picture. To train a deep neural network to do the same would require a much larger data set. Fully convolutional networks (FCN) are an improvement over CNNs. In a FCN the fully connected layer is replaced with a 1x1 convolutional layer, allowing spatial information to be retained. The benefit of a FCN is that it will say not just 'Is this a picture of an A?' but 'where in this picture is the A?'

Network Structure

Encoder

The encoder portion of the network is a series of convolution steps. The encoder gradually squeezes spatial dimensions while increasing the depth. More depth provides more feature maps for semantic segmentation.

1x1 Convolution

The use of a fully connected layer would change the output dimensions of the convolution tensors from 4D to 2D, causing spatial information to be lost. Replacing the fully connected layer with a 1x1 convolution maintains the spatial information.

Decoder

The decoder upscales the output of the encoder such that it is the same size as the original image. This allows the FCN to be applied to images of any size.

Skip Connections

Skip connections are connections of one layer's output to a non-adjacent layer's input. As convolutions are applied to an image the network is effectively looking closer and closer at the image. By including information from a different level of "zoom," bigger picture information is retained.

Separable Convolutions

The encoder and decoder blocks of the FCN were constructed using separable convolutions. In a separable convolution a convolution is performed on each channel of the input layer individually, and then a 1x1 convolution is performed on the output. Using separable convolutions reduces the number of parameters that are needed in the convolution step, decreasing the computational burden.

Batch Normalization

Another technique used to increase the training speed of the network is batch normalization. To do batch normalization the inputs to each layer of the neural network are normalized, instead of only the inputs to the first layer. The encoder, 1x1 convolution, and decoder layers are all implemented using batch normalization. In addition to increasing the overall training speed of the network, networks using batch normalization can use higher learning rates than other networks.

Bilinear Upsampling

The decoder must create a layer with more spatial information than the previous layer. In order to do this the decoder must use a technique to interpolate between points in the input layer. The method used for this project is called bilinear upsampling. For this process the weighted average value of the four nearest known pixels are used to estimate the value for the new pixel.

Code

The model for this project was built using Keras, an open source library containing abstractions for the constituent pieces of neural networks.

Encoder Block

The encoder block accepts three arguments: the input layer, the number of feature maps, and the stride length of the convolution. A batch-normalized separable convolution is applied to the input layer to create the output layer.

1x1 Convolution

The 1x1 convolution accepts two arguments: the input layer and the number of feature maps. A batch-normalized 1x1 convolution is applied to the input layer to create the output layer. A separable convolution is not used because it would not reduce the number of parameters for a 1x1 convolution.

Decoder Block

The decoder block accepts three arguments: the input layer, the skip connection layer, and the number of feature maps. The input layer is first upsampled using bilinear upsampling. Next the upsampled input layer is concatenated with the skip connection layer. A batch-normalized separable convolution is applied to the concatenated input layers to create the output layer.

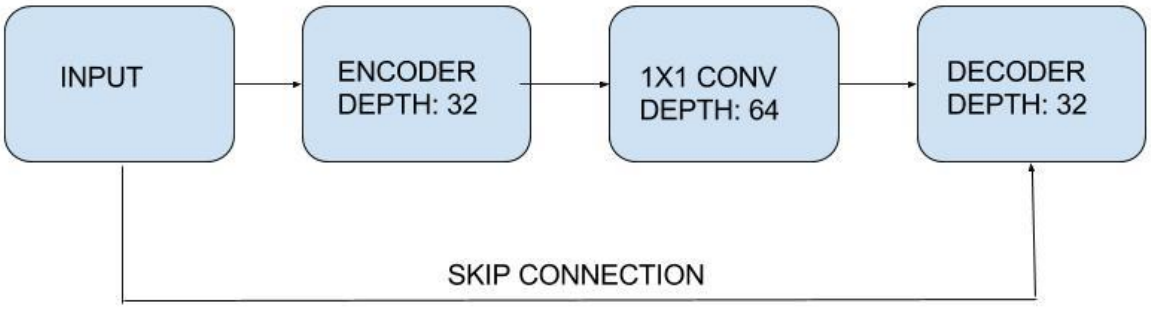
Training

All training was done using Amazon Web Services p2.xlarge instances.

Run 1

Model

For the first round of training I decided to start with a simple model to get a baseline score, using a single encoder block, a 1x1 convolution, and a single decoder block. There were two principles I used from the lessons in order to choose the depth. The first principle is that depth corresponds roughly with semantic complexity. The more feature maps in a layer, the more features the model can recognize. The second principle is that in each encoder layer the spatial information should be gradually squeezed while increasing the depth. Using these principles, I decided to start with a depth of 32 for the encoder block and 64 for the 1x1 convolution. The depth of the decoder block corresponds to the encoder block.



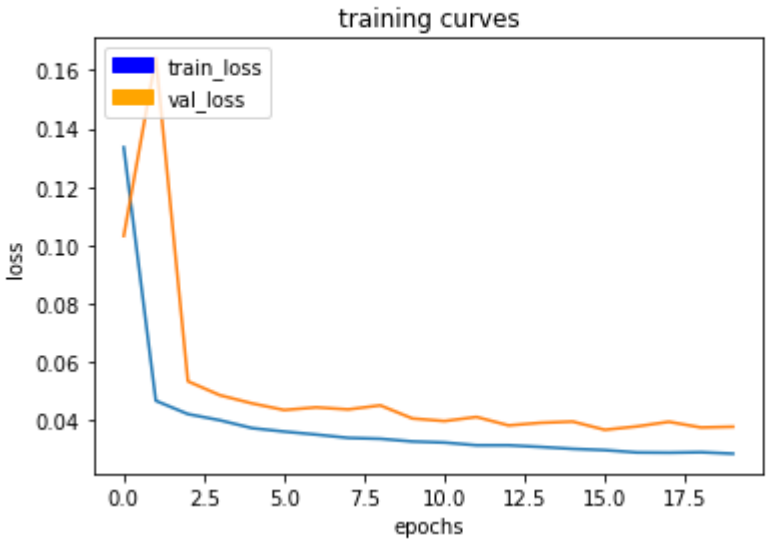
Hyperparameters

Learning Rate	0.01
Batch Size	256
Number of Epochs	20
Steps per Epoch	200
Validation Steps	50
Workers	4

I decided to start with a learning rate that had worked well in the labs. To take advantage of the computing power offered by the AWS instance I started with a large batch size of 256 and 20 epochs. The steps per epoch and validation steps were both default values in the workbook. The number of workers corresponds to the number of processes to spin up. The default value is 2, I chose 4 because the p2.xlarge instances have 4 processer cores, but I'm not sure if there is a direct correlation.

Results

```
Epoch 20/20
199/200 [============================>.] - ETA: 2s - loss: 0.0287
```

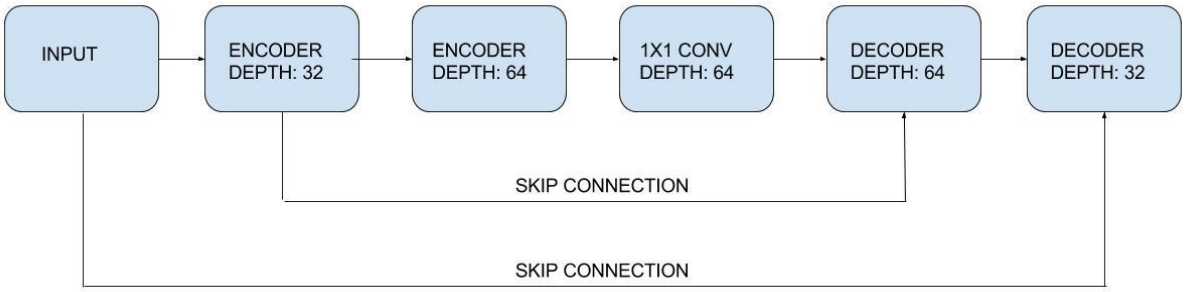


With the first model, the final intersection over union (IOU) score was 0.32.

Run 2

Model

In order to improve the model I decided to add another encoder and decoder layer. I increased the depth to 64 for the new layers so that the depth would be increased as the spatial dimensions were squeezed.

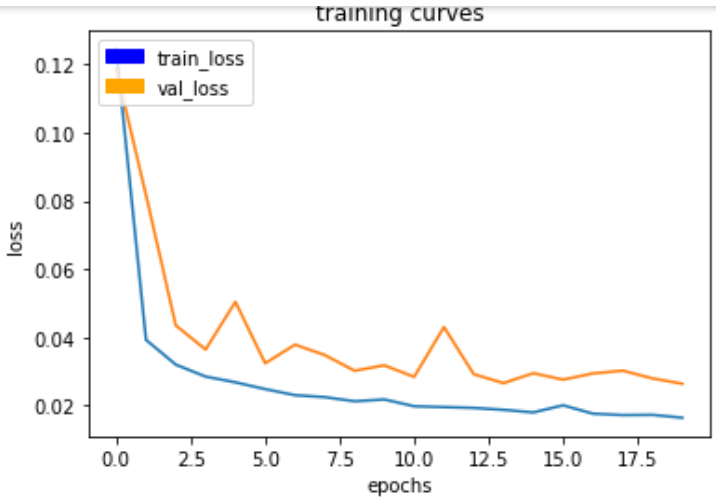


Hyperparameters

Learning Rate	0.01
Batch Size	128
Number of Epochs	20
Steps per Epoch	200
Validation Steps	50
Workers	4

For this run I originally left the hyperparameters the same as in the first one. With the more complex model, a batch size of 256 caused the AWS instance to crash. I reduced the batch size to 128 and started again.

Results

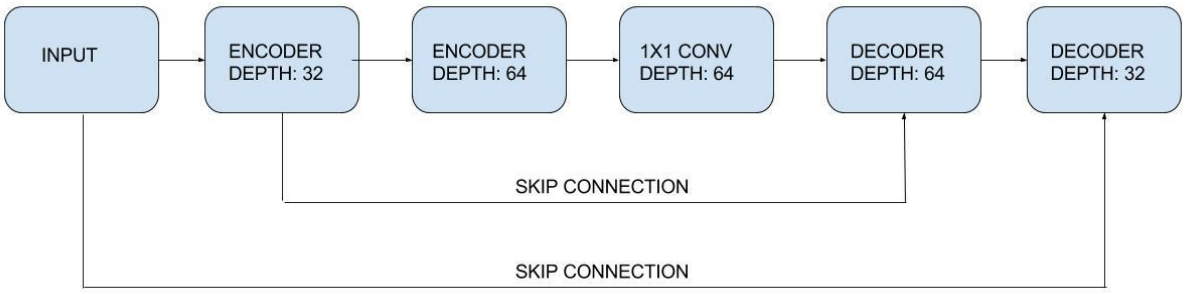



```
200/200 [==============================] - 279s - loss: 0.0164 - val_loss: 0.0264
```

The final IOU score for this model was 0.41, exceeding the goal of the project! Unfortunately I forgot to commit the changes to GitHub before terminating the AWS instance, so the entire run was lost except for the screenshot above.

Run 3

Model



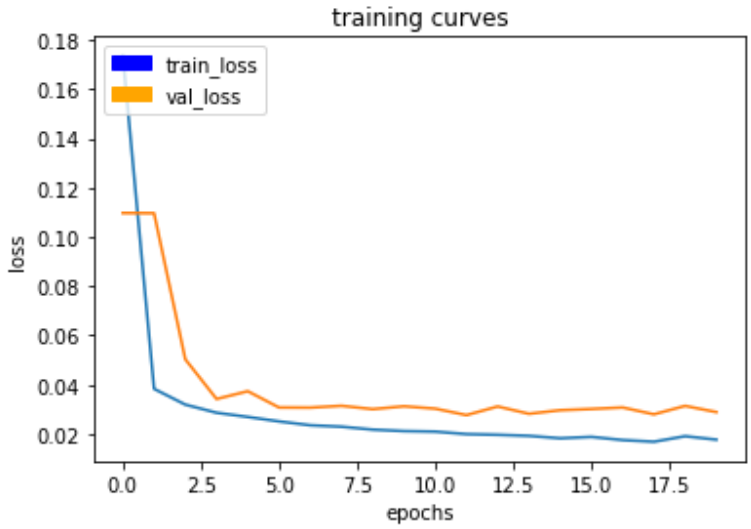
After the second run I knew the model was good enough to meet the goal so I kept it the same for the third run.

Hyperparameters

Learning Rate	0.005
Batch Size	128
Number of Epochs	20
Steps per Epoch	200
Validation Steps	50
Workers	4

I knew that my model was good enough to pass the project, but I thought I could improve it further. For the third run I dropped the learning rate to 0.005, thinking that this should cause the model to converge more slowly, but it should be better.

Results




```
200/200 [==============================] - 275s - loss: 0.0176 - val_loss: 0.0289
```

In [14]:
```
# Scores for while the quad is following behind the target.
true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, pred_following)
```

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9952254325172276
average intersection over union for other people is 0.33456564625660623
average intersection over union for the hero is 0.8684876276309296
number true positives: 538, number false positives: 0, number false negatives: 1
```

In [15]:
```
# Scores for images while the quad is on patrol and the target is not visable
true_pos2, false_pos2, false_neg2, iou2 = scoring_utils.score_run_iou(val_no_targ, pred_no_targ)
```

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.986102609872917
average intersection over union for other people is 0.7230414843193157
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 44, number false negatives: 0
```

In [16]:
```
# This score measures how well the neural network can detect the target from far away
true_pos3, false_pos3, false_neg3, iou3 = scoring_utils.score_run_iou(val_with_targ, pred_with_targ)
```

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9959032428257651
average intersection over union for other people is 0.4309227412212809
average intersection over union for the hero is 0.1879306761021616
number true positives: 117, number false positives: 1, number false negatives: 184
```

In [17]:
```
# 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)
```

```
0.7401129943502824
```

In [18]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.528209151867
```

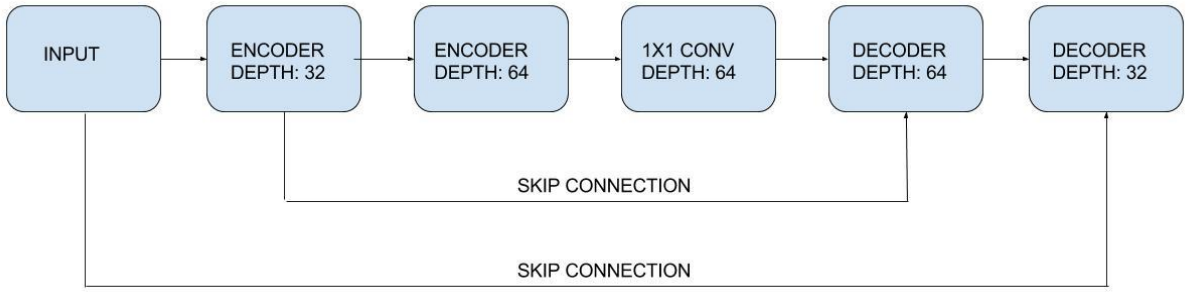
In [19]:
```
# And the final grade score is
final_score = final_IoU * weight
print(final_score)
```

```
0.390934457031
```

The final IOU score for this model was 0.39, below the goal threshold of 0.40. Looking at the training curves, it looks like this model ended up stuck at a local maxima, while the larger training rate of the model in run 2 allowed it to break through the local maxima and converge to a better result.

Run 4

Model



For run 4 I decided to repeat the second run. I knew it worked, I just needed to remember to commit my changes.

Hyperparameters

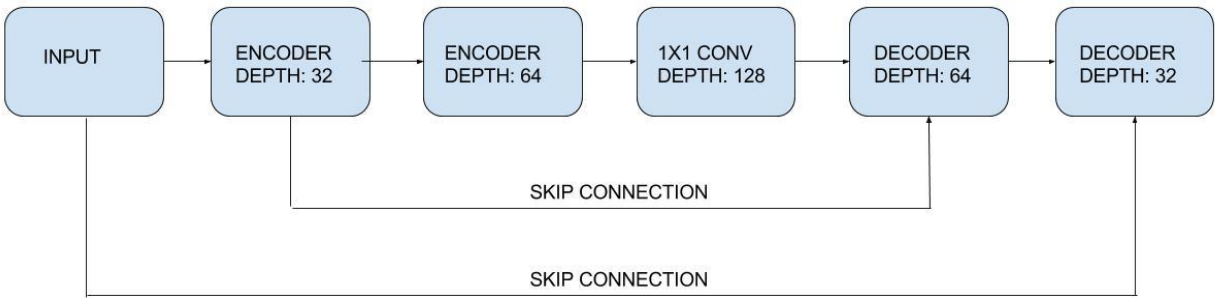
Learning Rate	0.01
Batch Size	128
Number of Epochs	20
Steps per Epoch	200
Validation Steps	50
Workers	4

Results

The final IOU score for this model was 0.38. It seems that the results in the second run depended on something outside of my control, such as the way the images were batched together or the starting weights. No matter what, the model needed further improvement.

Run 5

Model



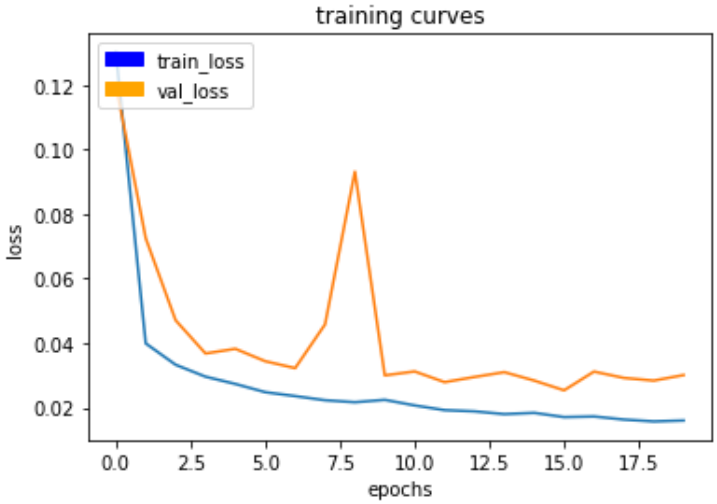
To increase the semantic complexity of the model I increased the number of feature maps of the 1x1 convolution layer to 128.

Hyperparameters

Learning Rate	0.01
Batch Size	128
Number of Epochs	20
Steps per Epoch	200
Validation Steps	50
Workers	4

I kept the hyperparameters the same as in run 4 in order to see the effects of the increased depth of the 1x1 convolution layer.

Results




```
200/200 [==============================] - 311s - loss: 0.0161 - val_loss: 0.0301
```

In [32]:
```
# Scores for while the quad is following behind the target.
true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, pred_following)
```

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9946563848415706
average intersection over union for other people is 0.33531073539282763
average intersection over union for the hero is 0.9096992352971635
number true positives: 539, number false positives: 0, number false negatives: 0
```

In [33]:
```
# Scores for images while the quad is on patrol and the target is not visable
true_pos2, false_pos2, false_neg2, iou2 = scoring_utils.score_run_iou(val_no_targ, pred_no_targ)
```

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9852624273458958
average intersection over union for other people is 0.7045863364797673
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 85, number false negatives: 0
```

In [34]:
```
# This score measures how well the neural network can detect the target from far away
true_pos3, false_pos3, false_neg3, iou3 = scoring_utils.score_run_iou(val_with_targ, pred_with_targ)
```

```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9950510551124655
average intersection over union for other people is 0.40457975159760967
average intersection over union for the hero is 0.26575954048654254
number true positives: 158, number false positives: 2, number false negatives: 143
```

In [35]:
```
# 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)
```

```
0.7518878101402373
```

In [36]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.587729387892
```

In [37]:
```
# And the final grade score is
final_score = final_IoU * weight
print(final_score)
```

```
0.441906562417
```

The final IOU score for this model was 0.44, improving on the previous models and exceeding the goal benchmark. Interestingly, the validation loss for this run was higher than previous models, but the IOU score was still significantly better.

Further Improvements

The quickest way to improve the model would be to collect more data. Looking at the training results of run 5, the highest error rate was when the target was far away. Collecting more images with the target in the distance would improve this score.

Other Applications

If this model was to be adapted for following another object, such as an animal or a car, new training images would have to be collected. Whether or not the model would work well for the new application depends on the semantic complexity of the new object and how much it differs from the non-target objects in its environment. At the very least, all new training data would be required.