Laura Curioso
12/3/2017

Project 4: Follow Me Writup

Project Overview

We are given several methods to utilize deep neural networks to train the model to find a "Hero" character human spawn and using the drone to follow that person around in the Unity quadcopter environment.

Data Collection

Due to time constraints, the provided data set was used to test the deep neural network written and several hyperperameters and convolution layers instead to optimize the model training.

Convolutional Neural Networks (Covnets)

Since the RGB pixel data is in multiple dimensions, we need to manipulate the data by matrix multiplication rather than simple multiplication. Since we know the data structure, we can manipulate it ahead of time to reduce the number of parameters needed to learn instead of starting from scratch. Convolutional neural networks, (CNNs) also known as covnets, are neural networks that share their parameters across space. CNNs add color channels (k amount) to the RGB data which add a greater "depth" to the data. The output also has a different height and width. This is called the patch size and since it is smaller we have less weights that are shared across space.

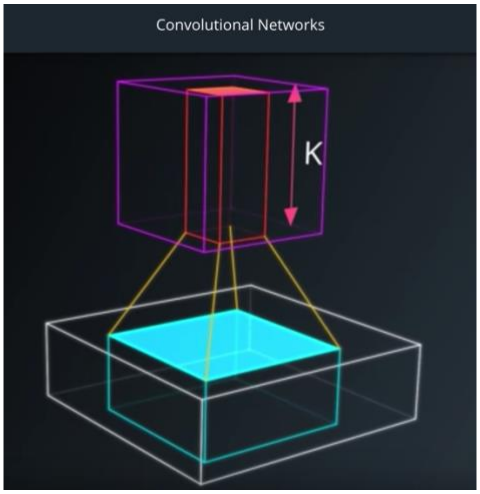


Figure 1: Convolutional Network Patch Creation Diagram

The key difference is instead of stacked matrix multiplied layers we instead get stacks of convolutions. These convolutions form a pyramid which squeeze the data from 256x256xRGB down eventually to 32x32x256.

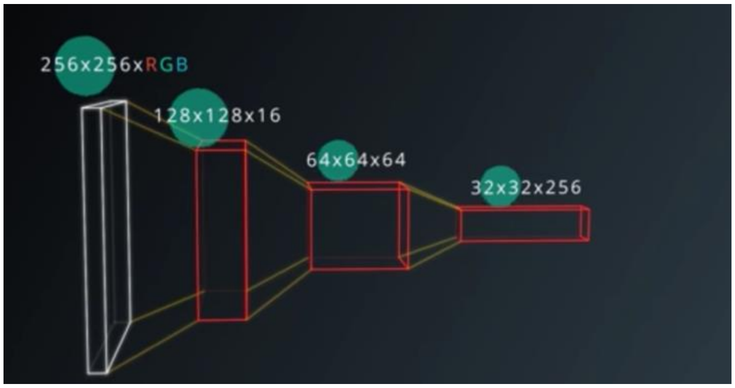


Figure 2: Convolution Data Pyramid

Encoder and Decoder layers

The encoder and decoder code blocks are where we extract and manipulate the image by using convolution layers. Each encoder layer is where we extract data from the image. For example, the first layer is able to define basic characteristics in the image such as lines, hues and brightness. The next layer is able to identify shapes like squares and circles. The caveat to having more layers to identify the detail, the more computing time is required. The decoder block scales-up the input of the encoded layer image to the same size as the original input image crating pixelwise element segmentation. For the decoder layer, it up-scales the output of the encoder layer so that it is the same size as the original image. This performs a segmentation and a prediction of each individual pixel in the original image.

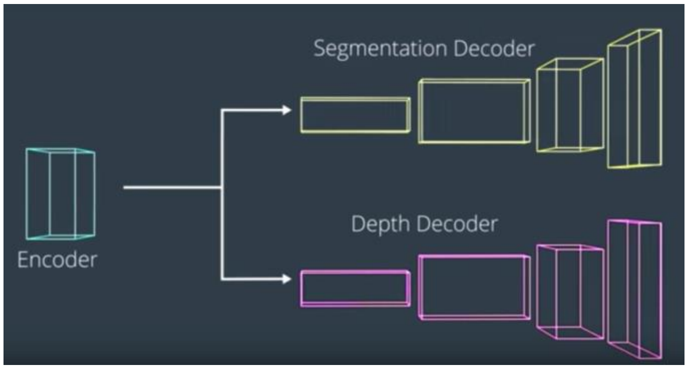


Figure 3: Decoder Diagram

Fully Convolutional Networks (FCNs)

By implementing a fully convolutional network, we can turn the pre-trained model data into transposed convolutions for further processing. In TensorFlow, the output of a convolutional layer is a 4D tensor. When an output of a convolution al layer is transferred into a fully connected layer, it flattens into a 2D tensor which results in the loss of spatial information. To resolve this data loss, the encoder and decoder blocks are connected by a 1x1 convolutional layer which turns the data back into a 4D tensor. The added advantage to doing this is that images of any size can be fed into the trained network.

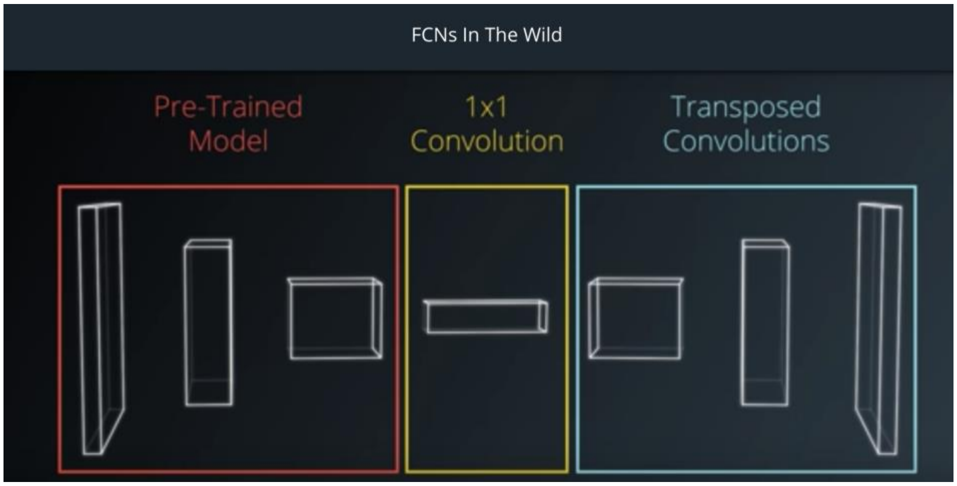


Figure 4: FCN Diagram Data Manipulation between Encoder and Decoder

Bilinear Upsampling

Bilinear upsampling uses the weighted average of the four nearest known pixels from that given pixel, and estimates the new pixel intensity value.

Skip Connections

Skip connections provide a means to keep information from the previous layers that were lost from convolution manipulation. Skip layers connect the output of one layer to the input of another layer. This allows the model to take information from multiple image sizes and make more intelligent decisions.

Testing

For the encoder and decoder blocks, I used two encoder and decoder layers utilizing two convolutions connected by one 1x1 convolution layer after trying a deeper model of 3 convolutions with an added 128 filter layer with the 1x1 convolution set to 256 filer size. My computer died so I was unable to see the result and since the compute time was longer than 8hrs I decided to change the convolution layers to two. The first convolution layer uses a stride of 2 and a filter size of 32 and the 2nd convolution layer uses a stride of 2 and a filter size of 64. The 1x1 convolution uses a filter size of 128 and the decoder filters filter the image down to 64 and then 32. The second decoder block also has a softmax function applied to it. The model of the FCN is shown below:

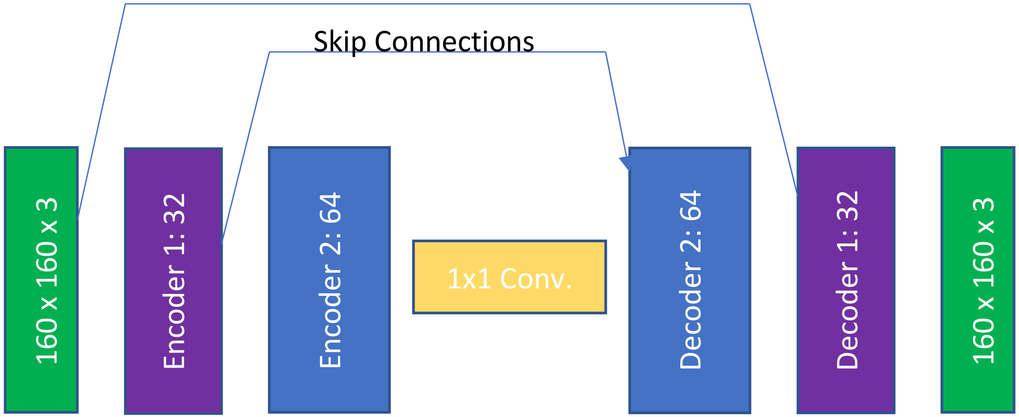
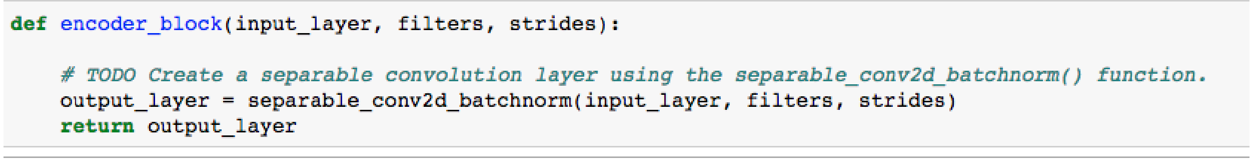


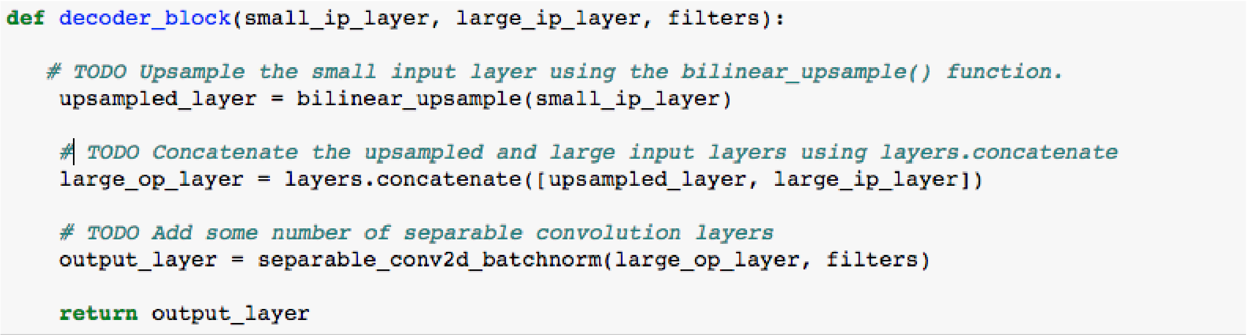
Figure 5: Training Model
Encoder/Decoder FCN Model w/ Skip Connections

For the encoder block:



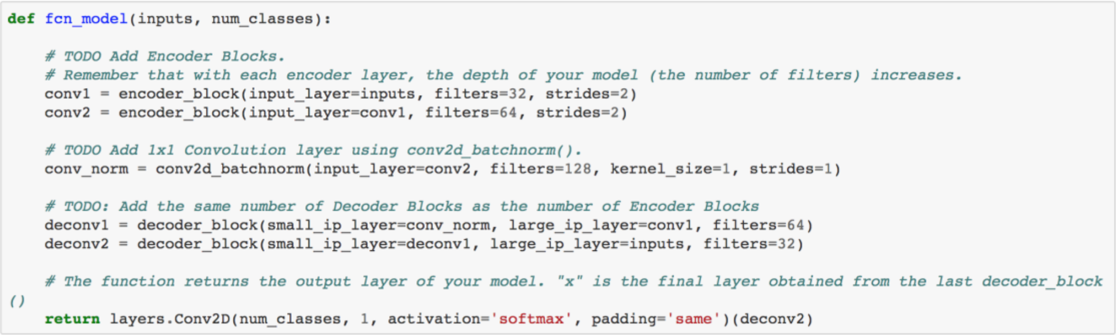
```
def encoder_block(input_layer, filters, strides):

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

For the decoder block:



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

    # TODO Upsample the small input layer using the bilinear_upsample() function.
    upsampled_layer = bilinear_upsample(small_ip_layer)

    # TODO Concatenate the upsampled and large input layers using layers.concatenate
    large_op_layer = layers.concatenate([upsampled_layer, large_ip_layer])

    # TODO Add some number of separable convolution layers
    output_layer = separable_conv2d_batchnorm(large_op_layer, filters)

    return output_layer
```

For the FCN block:



```
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.
    conv1 = encoder_block(input_layer=inputs, filters=32, strides=2)
    conv2 = encoder_block(input_layer=conv1, filters=64, strides=2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    conv_norm = conv2d_batchnorm(input_layer=conv2, filters=128, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    deconv1 = decoder_block(small_ip_layer=conv_norm, large_ip_layer=conv1, filters=64)
    deconv2 = decoder_block(small_ip_layer=deconv1, large_ip_layer=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')(deconv2)
```

Results

After deciding how many convolutions I wanted to use and what filters, I then played with the hyperparameters. From the segmentation lab, I used the inputs from that run which were:

learning_rate = .01
batch_size = 40
num_epochs = 20
steps_per_epoch = 400
validation_steps = 100
workers = 4

This gave a value of 0.4348 which was enough to pass the project. Since the run took the entire night (~8hrs) I decided to setup the AWS p2.xlarge instance to run the rest of the trials. Out of curiosity, for the 2nd trial, I added a second convolution step to the decoder block output layer to see what it did to the performance. That score was 0.408. After that I removed that modification and ran trials 3-5 under the following hyperparameters:

Parameters	Trial 3	Trial 4	Trial 5
Learning Rate	.005	.01	.02
Batch Size	40	100	100
Number of Epochs	20	20	50
Steps per Epoch	400	400	400
Validation Steps	200	100	100
Workers	4	4	8
IOU Score	0.3489	0.4171	0.4323

Table 1: Trials 2-5 Hyperparameters

Below are pictures of the training model from the best score, 0.4348, on trial 1:

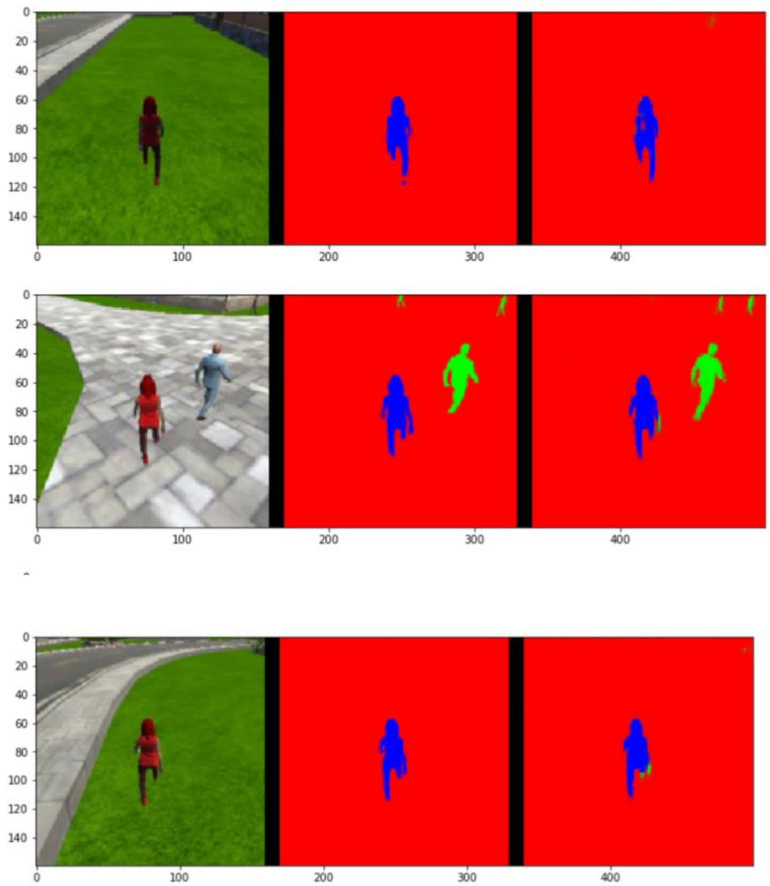


Figure 6: Images while following the target (hero)

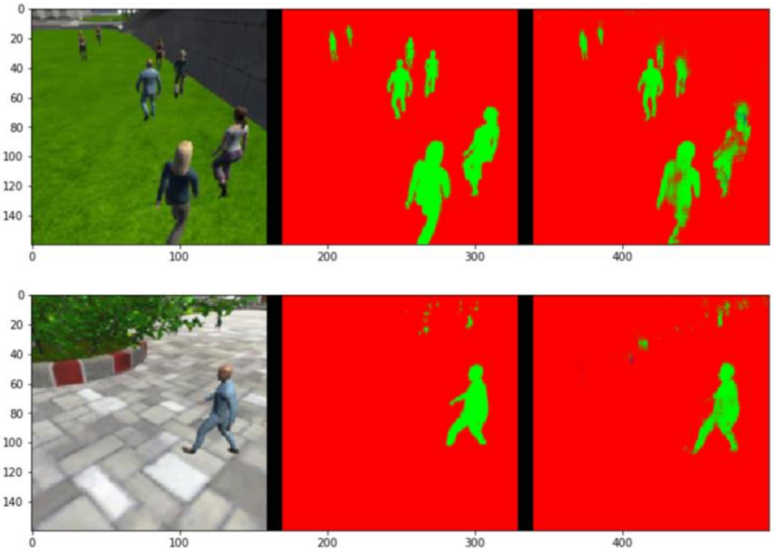


Figure 7: Images While at Patrol Without Target (snippet)

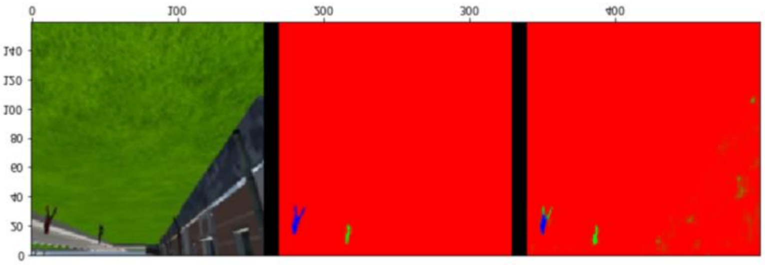
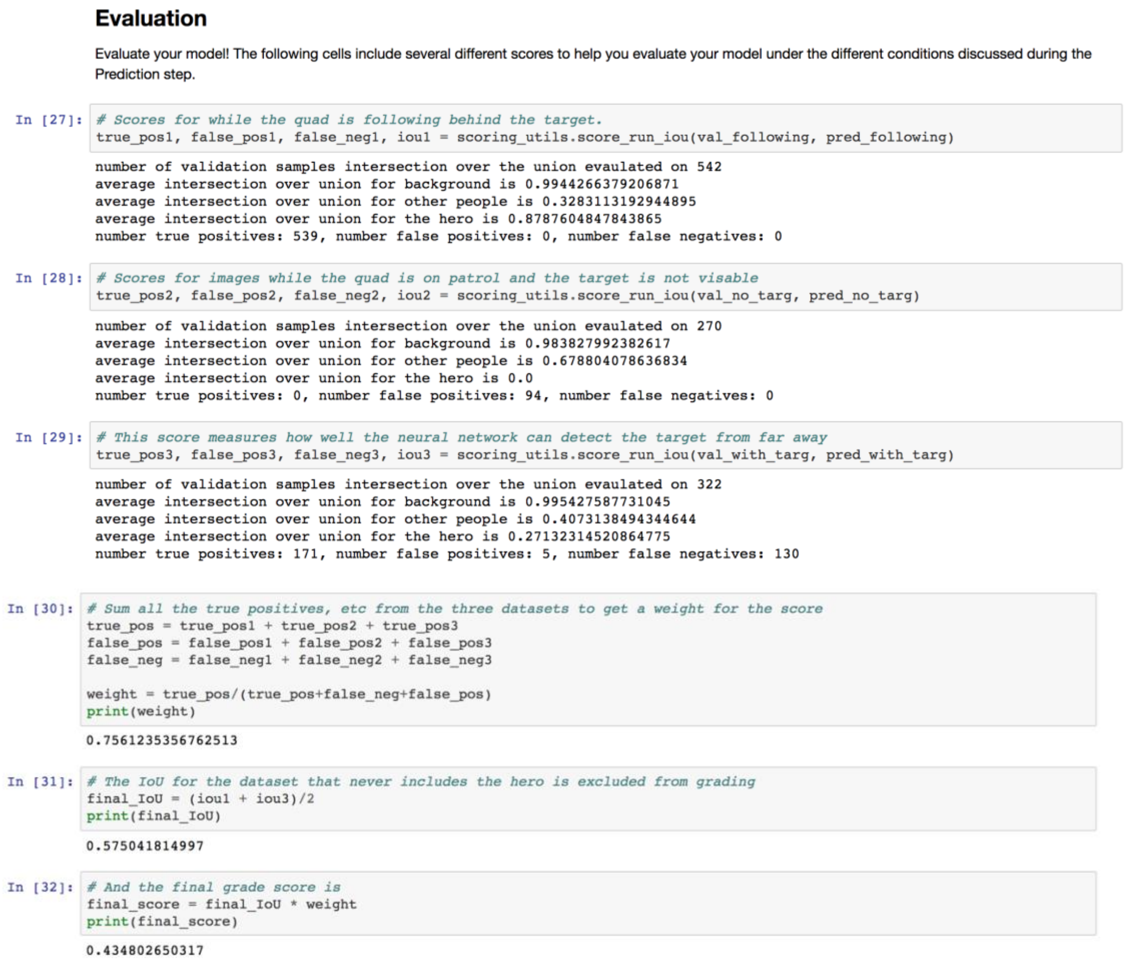


Figure 8: Images while at patrol with target(snippet)

Evaluation calculations



Evaluation

Evaluate your model! The following cells include several different scores to help you evaluate your model under the different conditions discussed during the Prediction step.

```
In [27]: # 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 0.9944266379206871
         average intersection over union for other people is 0.3283113192944895
         average intersection over union for the hero is 0.8787604847843865
         number true positives: 539, number false positives: 0, number false negatives: 0

In [28]: # 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 0.983827992382617
         average intersection over union for other people is 0.678804078636834
         average intersection over union for the hero is 0.0
         number true positives: 0, number false positives: 94, number false negatives: 0

In [29]: # 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 0.9954275877310 45
         average intersection over union for other people is 0.4073138494344644
         average intersection over union for the hero is 0.27132314520864775
         number true positives: 171, number false positives: 5, number false negatives: 130

In [30]: # 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)

         0.7561235356762513

In [31]: # The IoU for the dataset that never includes the hero is excluded from grading
         final_IoU = (iou1 + iou3)/2
         print(final_IoU)

         0.575041814997

In [32]: # And the final grade score is
         final_score = final_IoU * weight
         print(final_score)

         0.434802650317
```

Future Enhancements

For the current trained model, it would not work well identifying other objects such as dogs, cats, cars, street signs, etc. as the training data only focused on a specific “hero”. To be able to recognize other objects such as dogs, cats, cars, street signs, etc. those additional object image datasets would be needed to be added to the training data and the model would need to be re-trained from scratch again. Those new features would then be extracted, segmented, and with more logic, be able to (depending on intent) lock on and follow/avoid those specific objects.

Aside from adding different object data so the model can distinguish different objects such as pets, signs, etc. improvements to the current model training would be to use more training data of the hero at different angles so the drone can recognize them if they make a sharp turn. From an engineering approach, I would also setup a design of experiments (DOE) to help tweak the optimal hyperparameters. From brief research online, another technique that is popular to use is adaptive moment estimation (Adam). As with most neural networks, the more training data, the better the predictive model can be.