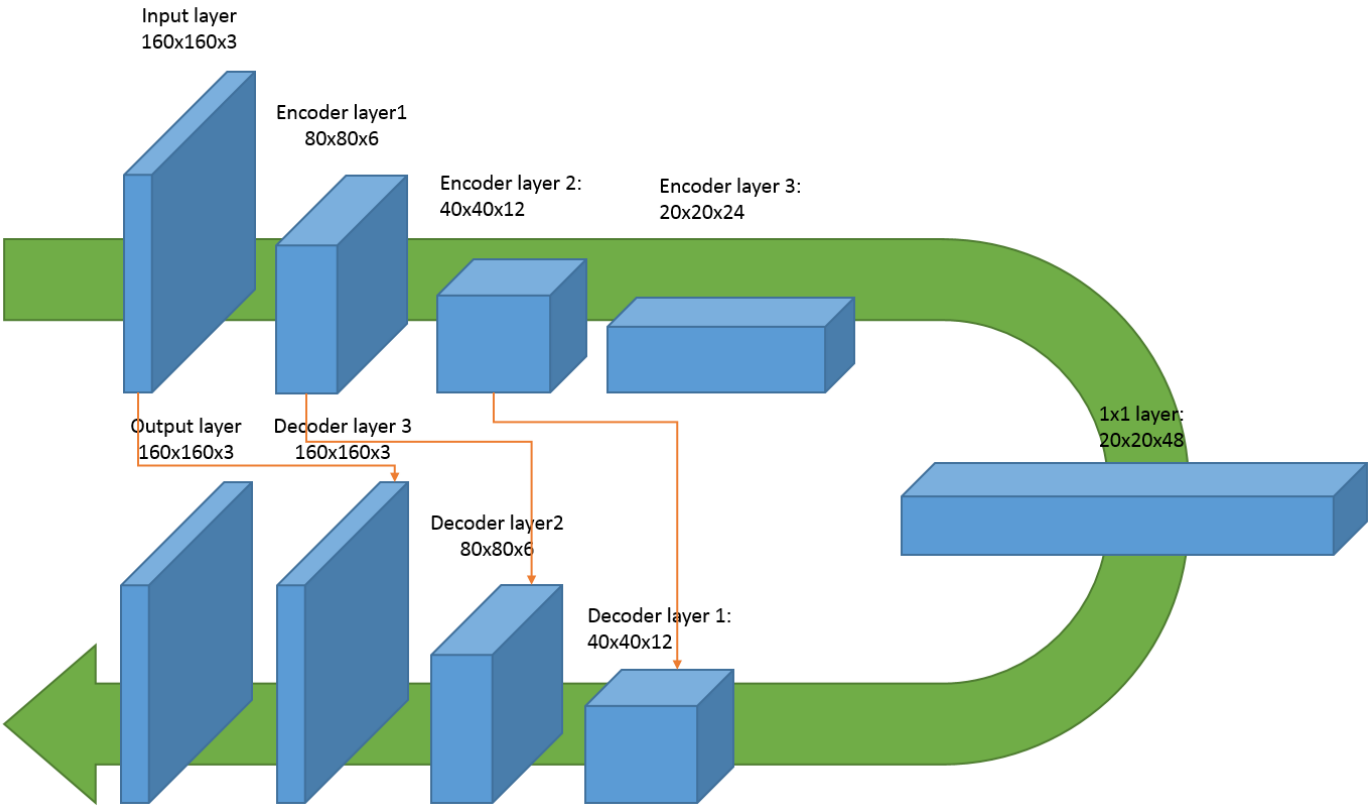
Project: 3D Perception

The student clearly explains each layer of the network architecture and the role that it plays in the overall network. The student can demonstrate the benefits and/or drawbacks of different network architectures pertaining to this project and can justify the current network with factual data. Any choice of configurable parameters should also be explained in the network architecture.

Code

https://github.com/nakata0705/RoboND-DeepLearning-Project

Neural network diagram



Explanation

For this project, I used a Fully Convolutional Network (FCN) with three encoder layers, one 1x1 convolutional layer and three decoder layers. FCN fits with this project since the objective of the image processing is identifying the location of the target object in the image. If I use a classic Convolutional Network, I can only identify if the input image is "the target object" or something else. The classic Convolutional Network will not allow me to identify where in the image the target object is.

I used three encoder layers and three decoder layers since I observed the tendency that the more encoder and decoder layers I use, the better the accuracy becomes. Also I observed tendency that the more filters I used in encoder layers, the better the accuracy becomes. Based on these two tendencies, I decided the width, height and filters in the diagram. Each encoder layer and decoder layer uses batch normalization to accelerate the

learning speed. The layer concatenation (skip connection) is used as in the diagram. The purpose is to give more detailed special information in encoder layer outputs to the upsampled decoder layer networks.

I experimented with different number of filters for 1x1 convolutional layer. General tendency seems that there is a sweet spot for the number. For filters=6, I got IoU 0.303. Filters=48, I got IoU 0.453 and for Filters=72, I got IoU 0.436. I believe the IoU value depends on the other hyperparameters such as num_epochs. But within my try and errors, filters=48 showed the best result.

The student explains their neural network parameters including the values selected and how these values were obtained (i.e. how was hyper tuning performed? Brute force, etc.)

The hyperparameters I used is as follows.
learning_rate = 0.002
I tried learning_rate 0.01 and 0.001. Lower learning rate results in the better IoU but the learning speed becomes slower. Within my try and error, the value 0.002 delivers the enough learning speed and accuracy.

batch_size = 57
I used 5635 training images and 5635 flipped training images. In total 11270 training images. The steps_per_epoch set to 200. And I used rounded up value of 11270 / 200=56.35 as batch_size which is the number of images processed in one step.

num_epochs = 32
I used 32 based on my try and error experience. Originally I used 16 but the number is too small. I tried 64 a few times but it seems the loss for training data continuously goes down while the loss for validation data doesn't improve after certain point. The processing speed on AWS was also I didn't choose large num_epoch.

steps_per_epoch = 200
This value is a constant I picked up. As far as it isn't too small and too big, that will not impact learning process.

validation_steps = 50
This value is a constant I picked up. As far as it isn't too small and too big, that will not impact learning process.

workers = 2
The value is given by Udacity as a recommended value.

The student demonstrates a clear understanding of 1 by 1 convolutions and where/when/how it should be used. The student demonstrates a clear understanding of a fully connected layer and where/when/how it should be used.

Fully connected layer is used to extract the features (specified by filters parameter) from the previous

convolutional layers. Thus it is placed after a convolutional network. Since it is connected to all outputs of the previous layer, the feature extracted by a fully connected layer only tells the feature of the overall input values.

On the contrary, 1x1 convolution layer performs a convolution with 1x1 kernel and 1 stride value. In this way, 1x1 convolution output the features (specified by filters parameter) with the same height and width value. This means 1x1 convolution extracts features for each input value. In other words, 1x1 convolution can keep the special information. Usually 1x1 convolution reduces the depth of the output, because the number of features extracted by 1x1 convolution is smaller than the output channels of the previous layer. Also 1x1 convolution itself is a linear manipulation, by putting non-linear activation function such as ReLU, 1x1 convolution can be used to add more non-linear process in the network.

I used 48 as a depth of 1x1 convolution output. My interpretation of the reason why adding more depth by 1x1 convolution produced the better result is the network needed more non-linear functions in it. And the reason why 72 as a depth of 1x1 convolution output didn't produce a better result should be the number of learning data and epochs became insufficient due to increased parameters.

The student is able to identify the use of various reasons for encoding / decoding images, when it should be used, why it is useful, and any problems that may arise.

Encoder extracts the features of the image. The width and height of the extracted feature is usually smaller than the input image. Decoder upsamples the features from the encoder and outputs the features in the same resolution as the input image. With this structure, the combination of encoder and decoder an extract the features of the input image for each pixel regardless the size of the picture.
So encoder and decoder should be used #1 in case the image size would have multiple resolutions, and #2 the one wants to extract per pixel features.

A potential problem is that the upsampling based on reduced spatial information would produce inaccurate per-pixel classification result. This issue can be addressed by layer concatenation (skip layer) by giving higher resolution information to decoder from encoders.

Another potential problem is the deeper output. As one wants to classify many types of features, the final layer will have deeper output. To avoid this issue, it might be useful to combining multiple simple features to describe the complete feature of a certain pixel. (Such as giving three network outputs brown, glass, bottle to another network to get a bottle of beer classification)

The student is able to clearly articulate whether this model **and** data would work well for following another object (dog, cat, car, etc.) instead of a human and if not, what changes would be required.

This model and data won't work for tracking cars, dogs and other objects. Since this model is trained to detect the "red person". (to be more accurate, it only has 3 classes, the target person, non-target person and everything else) To use the model for tracking other objects, the model must have more than three classes.

Then the model must be trained and validated with images that include other objects.

I expect each object requires about 10000 training images to achieve the same accuracy. So for example, if I want to classify the target person, a dog, a cat and a car, I would need in total 40000 training images.

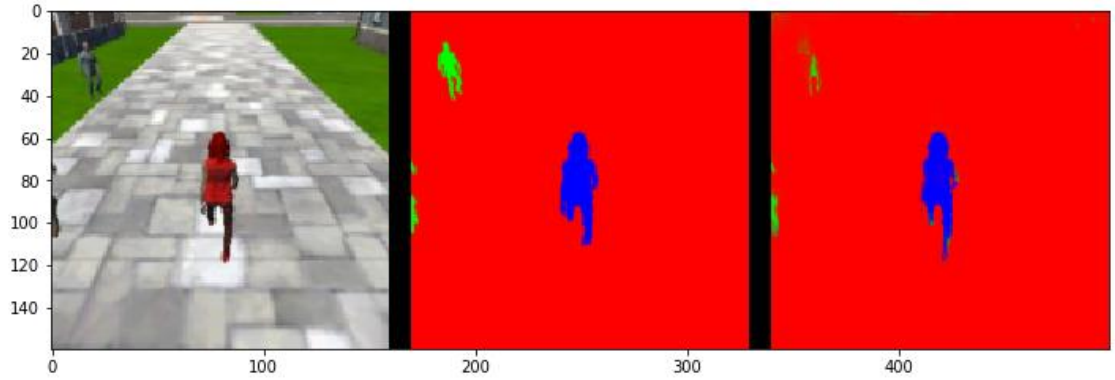
The file is in the correct format (.h5) and runs without errors.

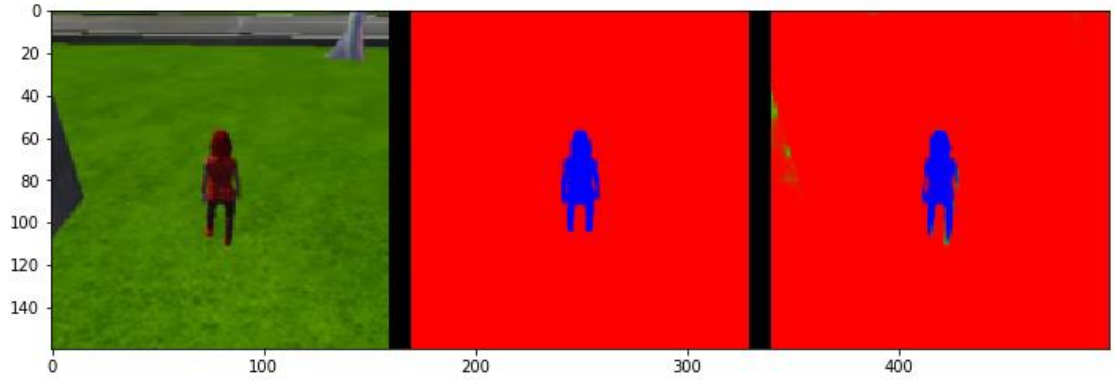
The model is data/weights/model_weights

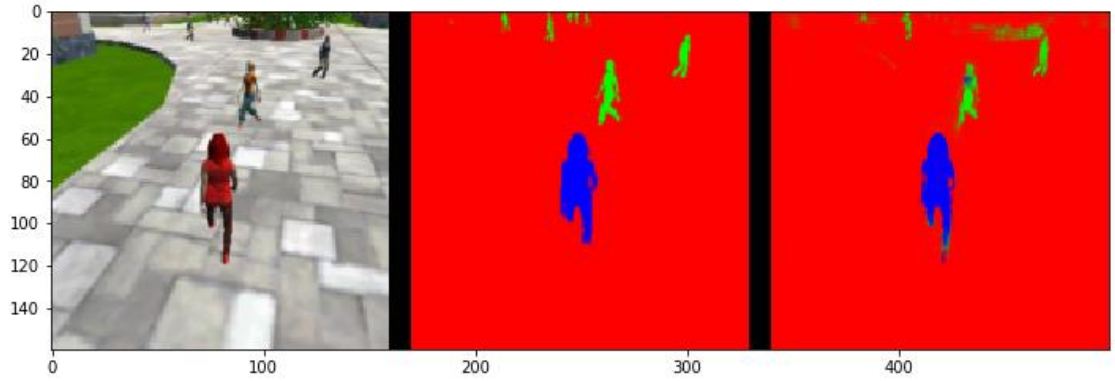
The neural network should obtain an accuracy greater than or equal to 40% (0.40) using the Intersection over Union (IoU) metric.

The final IoU is 0.452895133867 after being trained 11270 training images. (5635 original and 5635 flipped)
Here are some screen captures from the notebook.

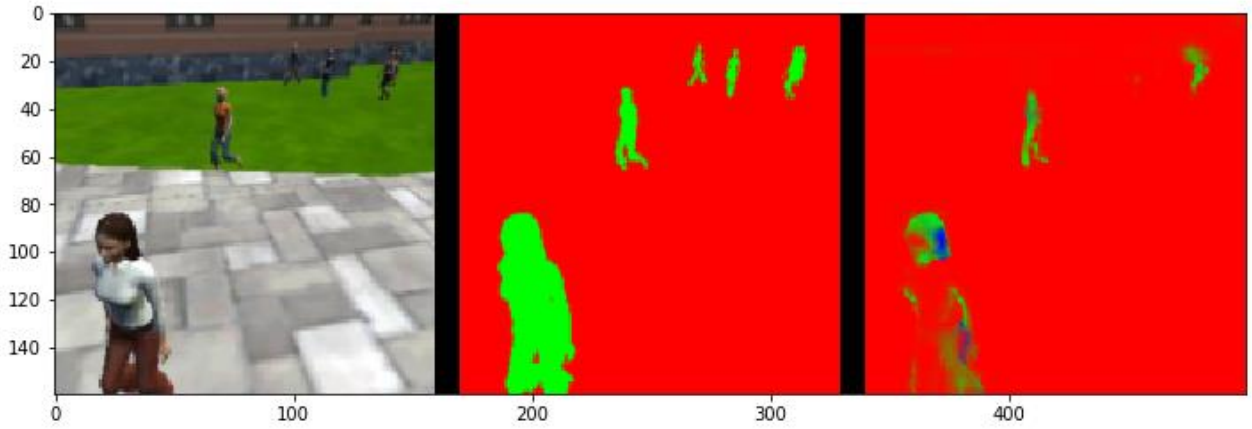
In [10]:
```
# images while following the target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','following_images', run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

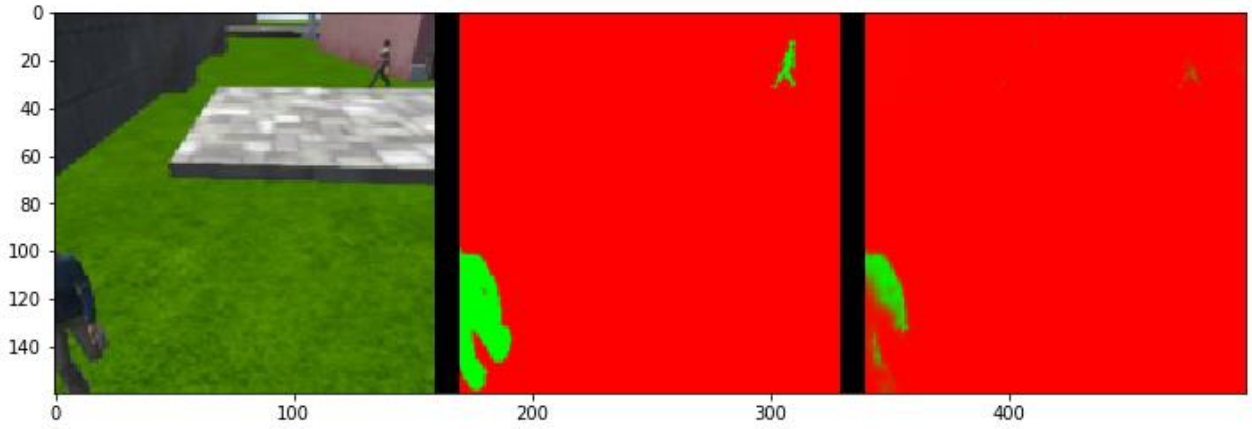


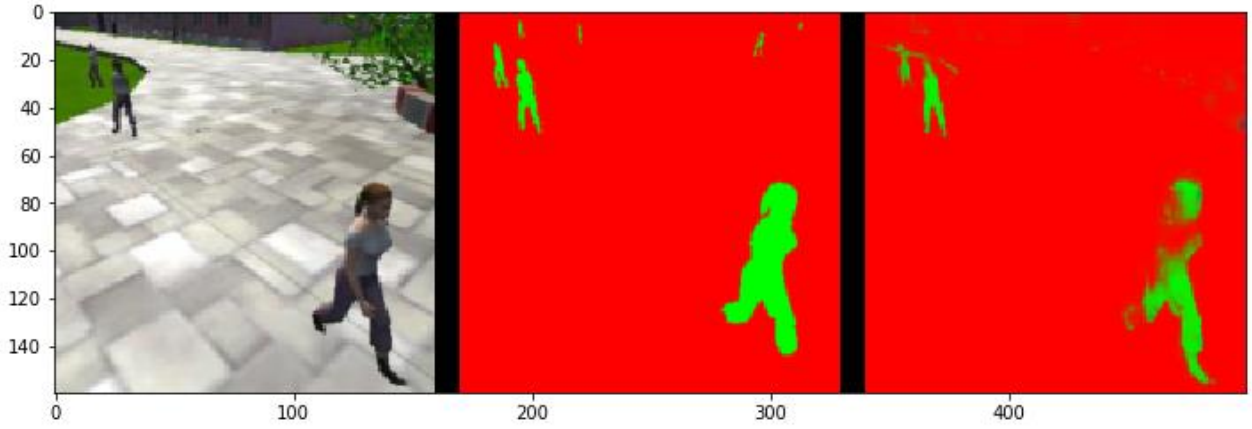




In [11]:
```
# images while at patrol without target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_non_targ', run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```

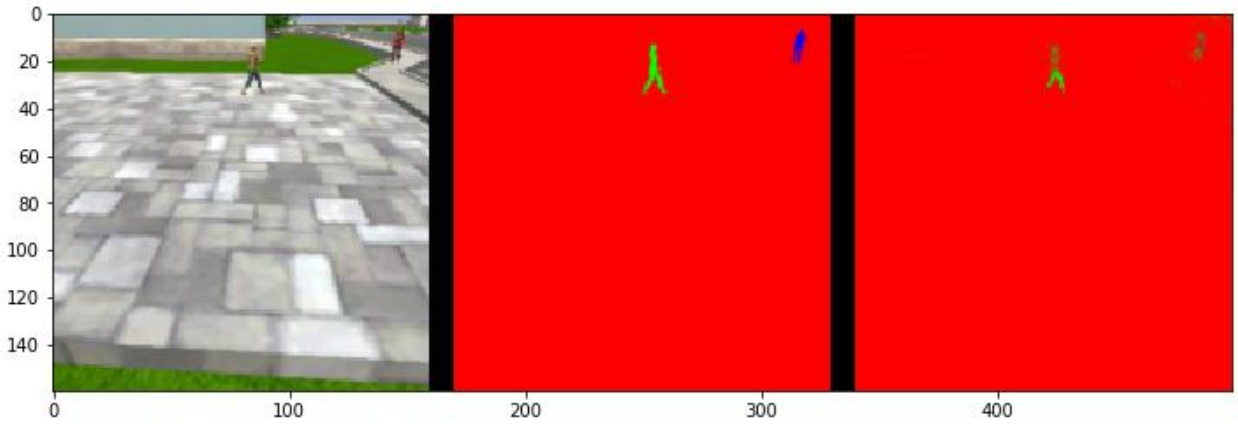


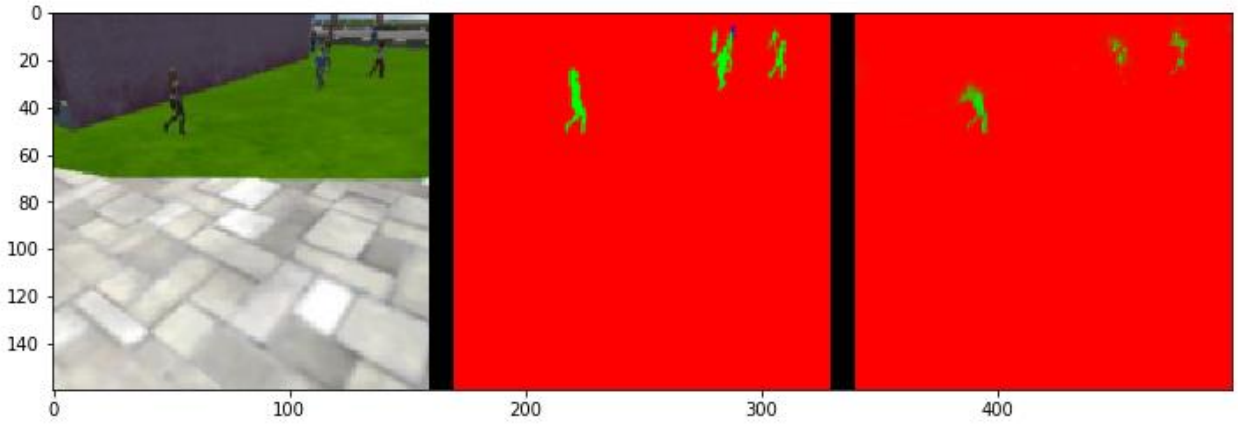


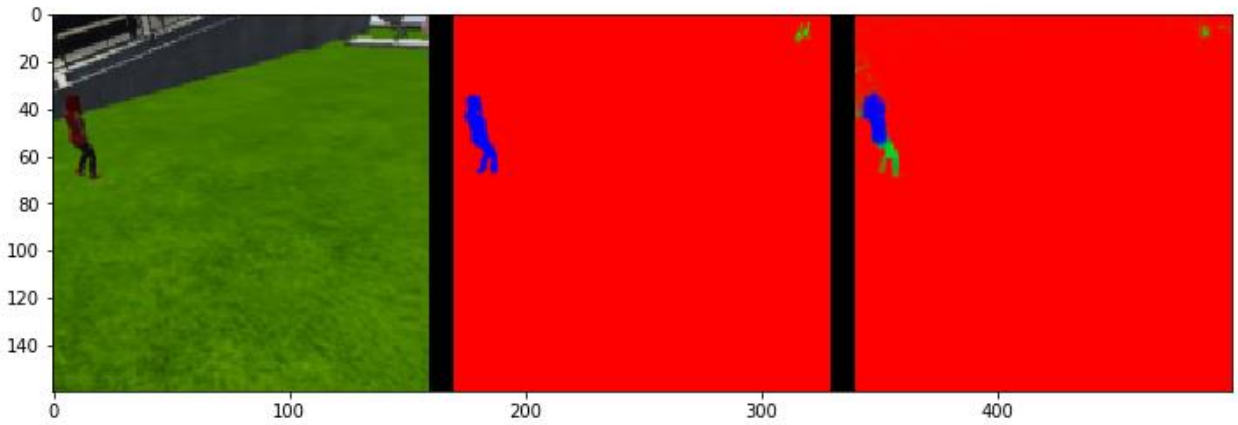


In [12]:

```
# images while at patrol with target
im_files = plotting_tools.get_im_file_sample('sample_evaluation_data','patrol_with_targ', run_num)
for i in range(3):
    im_tuple = plotting_tools.load_images(im_files[i])
    plotting_tools.show_images(im_tuple)
```







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 [13]:
```
# 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 0.9924082848284197
average intersection over union for other people is 0.16958874082369582
average intersection over union for the hero is 0.7845841913792875
number true positives: 539, number false positives: 0, number false negatives: 0
```

In [14]:
```
# 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 0.977702695289554
average intersection over union for other people is 0.5196549259812173
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 76, number false negatives: 0
```

In [15]:
```
# 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 0.9941934953216939
average intersection over union for other people is 0.2527811231234035
average intersection over union for the hero is 0.12120607635489268
number true positives: 116, number false positives: 2, number false negatives: 185
```

In [16]:
```
# 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)
```

```
0.7135076252723311
```

In [17]:
```
# The IoU for the dataset that never includes the hero is excluded from grading
final_IoU = (iou1 + iou3)/2
print(final_IoU)
```

```
0.452895133867
```

In [20]:
```
# And the final grade score is
final_score = final_IoU * weight
print(final_score)
```

```
0.3066399648
```