Project: Follow Me

By Lingzhi Kong

Udacity Robotics Nano Degree

21 Sep., 2018

Abstract

This project aims to build and train a fully convolutional neural network (FCN) and then utilize it to identify and track a moving target in a drone simulation environment. The first step is to build a network that takes images coming from front-facing camera on the drone as input, and then classifies each pixel of each image to locate the moving target. The FCN consists of 7 layers, 3 for encoder, 3 for decoder, and 1 for convolutional layer. The last decoding layer serves as the output layer which returns the result. After the FCN has been trained, we evaluated the results and tested it in the simulator. These perception techniques are key to solve a wide range of robotics problems.

2 Details

2.1 Introduction

2.1.1 Overview

Computer vision is a subset of robotic perception. A simple example of a task that is routinely performed by a human visual system, is to answer the question: Is there a dog in the picture? and where is the dog in the picture?

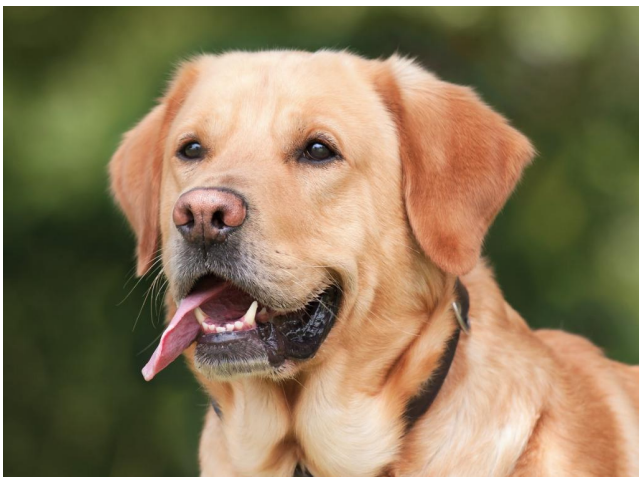


Fig. 2.1 Computer vision problem

This project will focus on classification, using an approach called semantic segmentation. Semantic segmentation is defined as a method of inference which is able to categorize fine image details. In this project, it is achieved by a fully convolutional neural network (FCN), which classifies each pixel in the image, and labels it with the class of its enclosing object of region. The 3D simulated environment built by Unity is a small city consisting of buildings, roads, elevated highways, vegetation, and pedestrians. The robotic agent is a drone, as shown in the following figure. The drone agent roams in the environment until it is able to locate the hero using the trained FCN, at which point the UAV will track the hero.

Fig. 2.2 Simulated environment

2.2 The architecture of FCN

2.1.1 Overview of FCN

Before introducing what is FCN, we first have a look at deep neural networks (DNN). Fully connected feed-forward DNNs have proven effective in a wide range of computer vision classification problems. One successful application is its use in MNIST dataset. However, DNNs performs poorly when it comes to image variation in the spatial domain. The problem can be better understood by considering the following figures. Suppose we create a DNN to classify whether an image has a puppy in it or not, and we train this DNN with lots of images like the one on the left hand side. If the trained model is used to classify the images like the one on the right-hand side, it would perform poorly. This is because our model would have only learned to classify pixel features on the left side of the picture with puppies, which say nothing about classifying a puppy on the right hand side of an image. In other words, there is no translational invariance in the model.

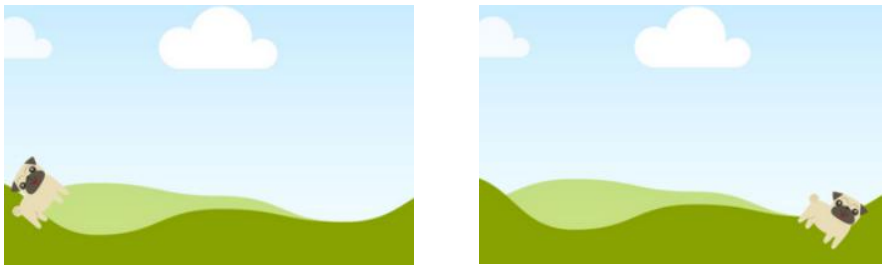
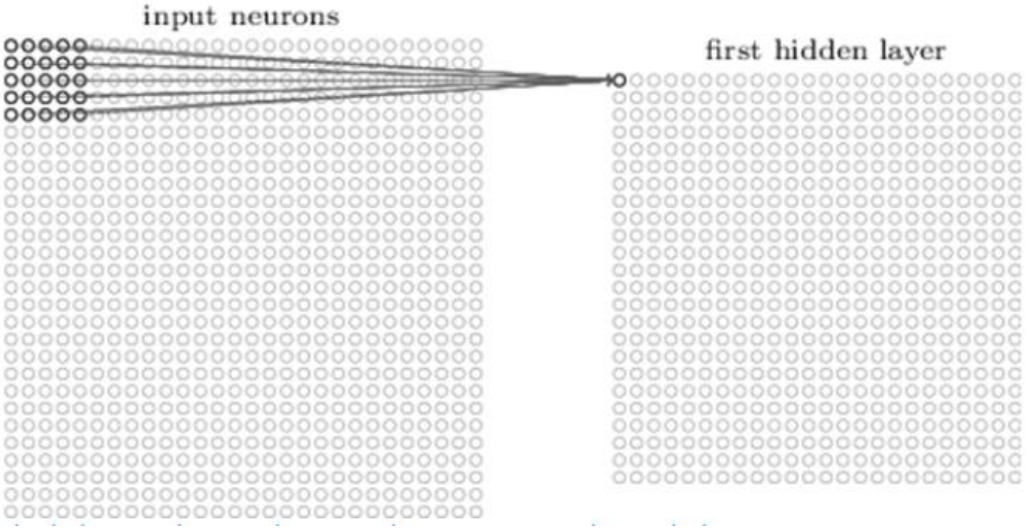


Fig. 2.3 The case that DNN performs poorly

To address this issue, a convolutional Neural Network (CNN) comes into play.



CNN layers with a 3D Kernal size will look like this:

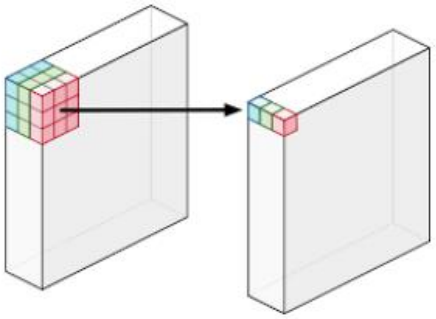


Fig. 2.3 CNN layer

Based on the idea introduced above, we can build a more complicated CNN consisting of a couple of layers, which can capture a different level in the hierarchy of objects. The first layer generally classifies small parts of the image into simple shapes. The subsequent layers tend to be higher levels in the hierarchy and generally classify more complex ideas like shapes and eventually full objects like cars. The following figure shows the typical architecture of CNNs.

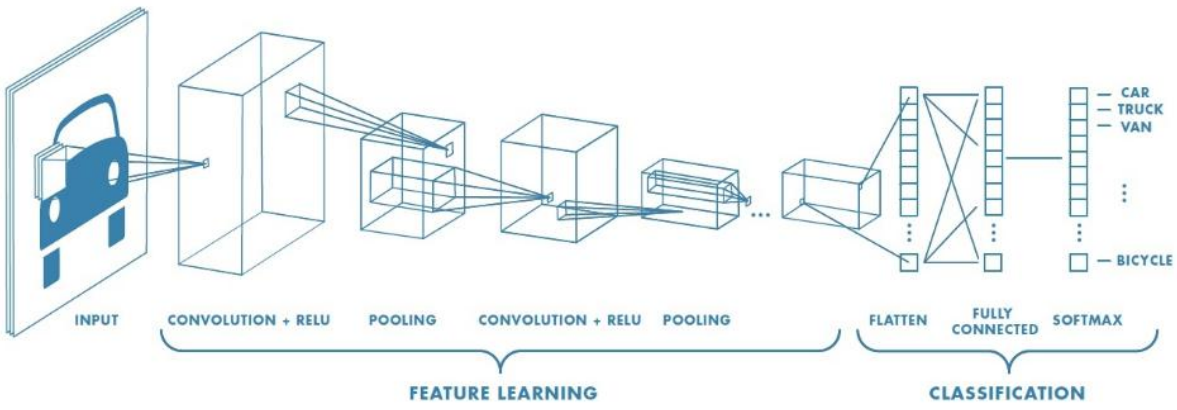


Fig. 2.4 Typical CNN

CNNs have proven better than DNNs in the image classification domain. While CNNS can achieve noteworthy performance in the task of image classification, there is a notable performance loss when they are re-tasked with pixel by pixel classification known as semantic segmentation. This is due to the fact that fully connected layers do not preserve spatial information.

To tackle the problem of pixel by pixel classification, FCNs were coined by researchers in recent years. A fully Convolutional neural network (FCN) is a normal CNN, where the last fully connected layer is substituted by a 1x1 convolution layer with a large “receptive field”. The idea here is to enable us to know what are the objects and where are they in the scene. The typical architecture of FCN is shown below.

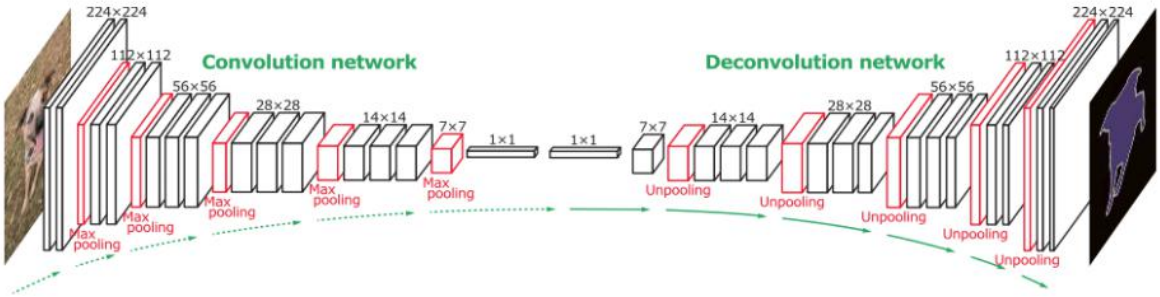


Fig. 2.5 The architecture of FCNs

FCNs preserve spatial information, and represent a state-of-the-art approach to semantic segmentation. They can be thought of as two distinct parts: encoders and decoders, as shown in the above figure. The encoder is

comprised of several convolutional layers, which are typically arranged to prograssived concentrate the spatial domain, and increase the number of channels in the image. These different layers of spatial compression are useful for training the model on different image resolutions. One problem with this approach is that we lose some information when we do convolution. That is, we keep the smaller picture and lose the bigger picture. To solve this problem, we use a technique called skip connections. To be specific, we get some activation from previous layers and sum them together with the up-sampled outputs when decoding from the previous layer as shown in below figure. The decoder, in contrast, upsamples encoder compressions, restoring the spatial dimensions, before being passed to a convolutional layer with a softmax activation function.

FCNs are often make use of 1 x 1 convolutions, which are just a 2D convolution of patch size 1 x 1, and stride of 1. The spatial dimensions of the convolved image are preserved in the output volume, but the desired number of filters changes the output volume depth, making them computationally cheap and the subsequent operations less expensive.

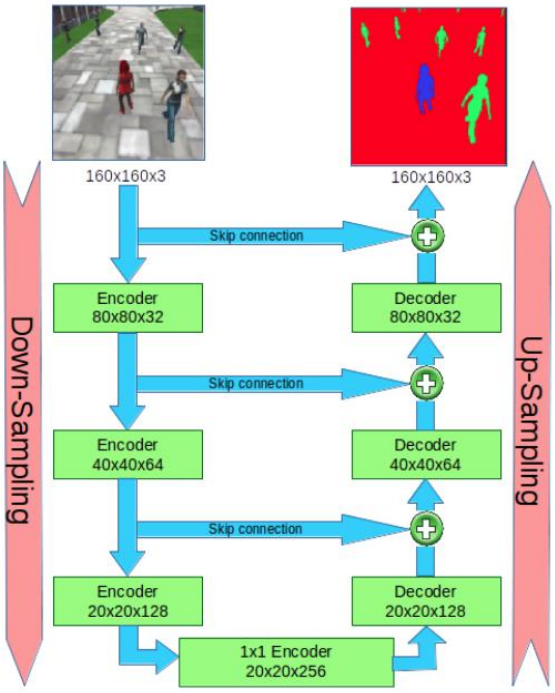


Fig. 2.6 FCN in this project

To sum up, FCN consists of the following components:

1. Encoder blocks
2. 1x1 Convolution block
3. Decoder blocks
4. Softmax activation

2.1.2 FCN in this project

Below is a illustration of the network architecture used in this project. As shown in the figure, the network consists of 7 layers: 3 encoder layers, 1 1x1 convolutional layer, and 3 decoder layers, with the last decoding layer serving as the output layer which returns the result.

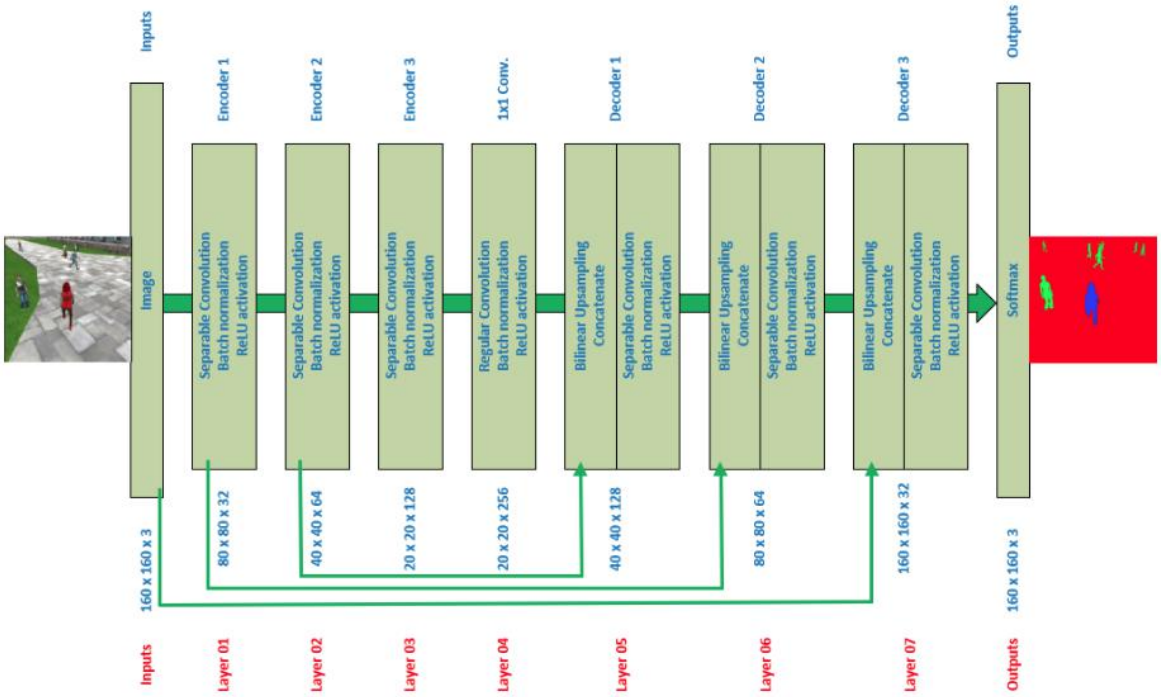


Fig. 2.7 FCN architecture in this project

The selection of network parameters:

1. Learning rate
   a) Value: 0.001
2. Number of epochs
   a) Value: 100

3. Batch size
    a) Value: 100
4. Steps per epoch
    a) Value: 41
5. Validation steps
    a) Value: 12
6. Workers
    a) Value: 4

Explaination of Network Parameters

Training the model requires specifying several hyperparameters. This section will attempt to discuss these parameters, indicate the values used, and clarify why such values were chosen.

1) Batch size

Value used: 100

What is the batch size?: The batch size is the number of images that are processed together as a batch for every step of the epoch or the training period. So in this instance, for every step of training, 100 images are taken from the training set and are used for the gradient descent step to minimize the error or loss value.

What makes for a good batch size?: A good batch size would be one that is large enough such that it can still be handled by available computing resources, but at the same time not too large to cause overfitting. It should also be small enough but not too small. If the batches are too small, the neural network may have difficulty forming its generalizations.

Justification for the batch size used: The default value provided by the project was 20 images. I used 100 here because the computing resource can handle such number of images per epoch.

2) Learning Rate

Value used: 0.001

What is the learning rate?: The learning rate is a value that sets how quickly (or slowly) a neural network makes adjustments to what it has learned while it is being trained. In more concrete terms, it is a numerical value (usually less than 1) that we multiply to either the error minimization rate we get from logistic regression, or multiply to the gradient we get from gradient descent. A value of 1 means that we take the gradient or error minimization rate as is and make adjustments at that value. A value of 0.10 would mean that we make learning adjustments at 10% the rate of the computed gradient or error minimization rate. As an example, if the error minimization rate is +10 and the learning rate is 1, then we make an adjustment with the value of +10. However, if the learning rate is 0.1 then we only make an adjustment of +1.

What makes for a good learning rate?: The ideal learning rate would allow a neural network to reach the gradient descent and error minimization minimums at the least amount of time, without failing to converge.

How to obtain the best learning rate?: Every neural network and model has its own optimal learning rate. Determining this amount normally involves testing different values to see which value would be most effective.

What happens if we use the wrong learning rate?: Using learning rates that are too large would cause error rates to increase instead of decrease.

Justification for the learning rate used: This is the default learning rate provided in the project. Learning rates of 0.003, 0.002, and 0.001 were also used. To determine the optimal learning rate, a small experiment was conducted, looking at model performance for different learning rate, holding other hyper-parameters constant. The result shows that 0.001 is better.

3) Epochs

Value used: 100

What is an epoch?: Ideally, an epoch is a single run that will let the model use or "see" all the available training data to minimize its errors. Basically, an epoch is a single complete pass over all (or most) of the available

training data.

What makes for a good value for an epoch?: The ideal number of epochs would be the lowest possible value that would still be able to fully minimize the error rates of the neural network being trained, and avoid overfitting at the same time.

What happens if we use the wrong number of epochs?: A too low number of epochs would be insufficient in training the neural network to reach an acceptable error or loss rate, rendering it incapable of properly performing its intended function (i.e. the neural network won't be good enough to actually be useful for what it was designed, in our case, detecting an identified object). A too high number of epochs may cause overfitting, meaning the neural network may become very good at performing its tasks while using the training set. However, once it is fed non-training set data, it would not be able to perform in an acceptable manner. Furthermore, using more epochs than that which is necessary would be a waste of compute resources.

Justification for the number of epochs used: The number of epochs used started off at a low value of 5, and was increased to 10, 25, 50 then 100 when it seemed like the neural network was still capable of pushing loss and error rates downward, and that a low number of epochs was prematurely interrupting the neural network from reaching the minimum error plateau. In other words, instead of continuing to minimize the loss or error rates, the neural network stops computing because the number of epochs were already exhausted. This is even before the error rates have reached a plateau. Increasing the number of epochs was also done as the network's layers increased in "thickness", or when additional layers were added. When network "depth" increased, error rates tended to reach its minimum plateau at a slower rate, necessitating more epochs to reach these plateau values.

4) Steps Per Epoch

Value used: 41

What are the steps per epoch?: A training step is basically one gradient update or one error loss update. This is the number of image batches the neural network has to go though during training. In our instance, the batch size is 100, meaning the neural network uses 100 images every time it computes and updates the gradient and error. Since there are 41 steps per epoch, we end up processing 4100 images (41 x 100) for each epoch.

5) Validation Steps

Value used: 12

What are validation steps?: Similar to steps per epoch, but instead of using training data, we use validation data.

6) Workers

Value used: 4

What are workers?: Workers are the number of instances to spin up. The project notes imply that this value is ultimately limited by available compute resources. Here we used the default value 4.

2.2 Software & Hardware & Dataset used for Project training

2.2.1 AWS

p2.xlarge instance

AMI: Udacity Robotics Deep Learning Laboratory

4 x Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz

NVIDIA GPU Tesla K80 (1xGK210GL with 12GB Memory & 2496 CUDA cores)

40GB Memory

2.2.2 Frameworks and packages

Python 3.x

Tensorflow GPU 1.2.1

NumPy 1.11

SciPy 0.17.0

eventlet

Flask

h5py

PIL

python-socketio

scikit-image

transforms3d

PyQt4/Pyqt5

2.2.3 Dataset

The initial data set provided with the problem consists of 4142 images. There are three rough categories that the images can be placed in:

1. Images while the drone agent is following the hero;
2. Images while the drone agent is patrolling, but cannot see the hero;
3. Images while the drone agent is patrolling and can see the hero.

When an image is captured, pixel label data is also provided in the form of image masks.

2.3 Training results

2.2.1 Prediction

The network is trained using an optimizer Adam, which is similar to Stochastic Gradient Descent (SGD). To evaluate how well the FCN model fits for different conditions, we will test the performance using three different predictions available from the helper code.

Firstly, we compare our predictions to the ground truth labels and original images.

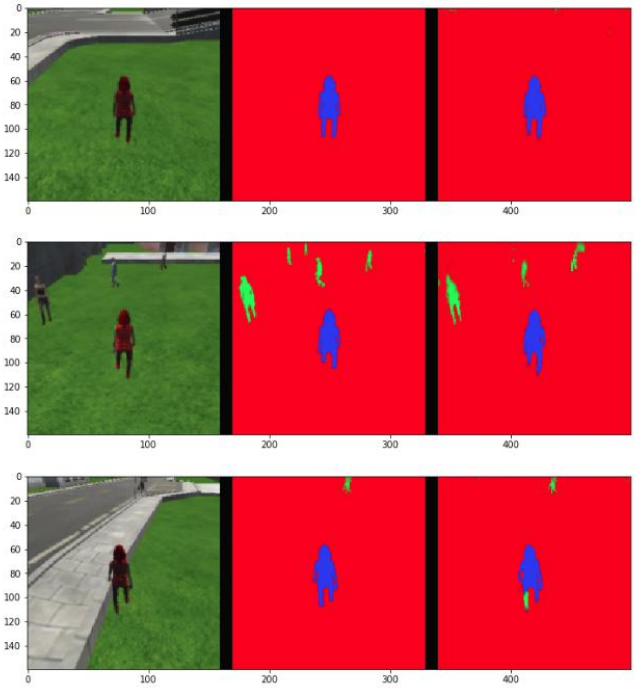


Fig. 2.8 Comparision between ground truth labels and predictions(images with hero)

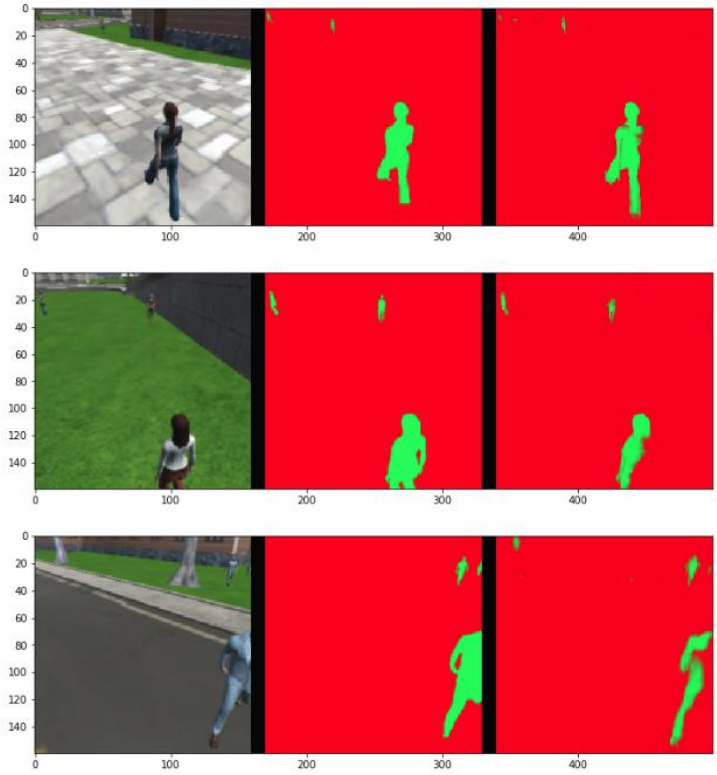


Fig. 2.9 Comparision between ground truth labels and predictions(images without hero)

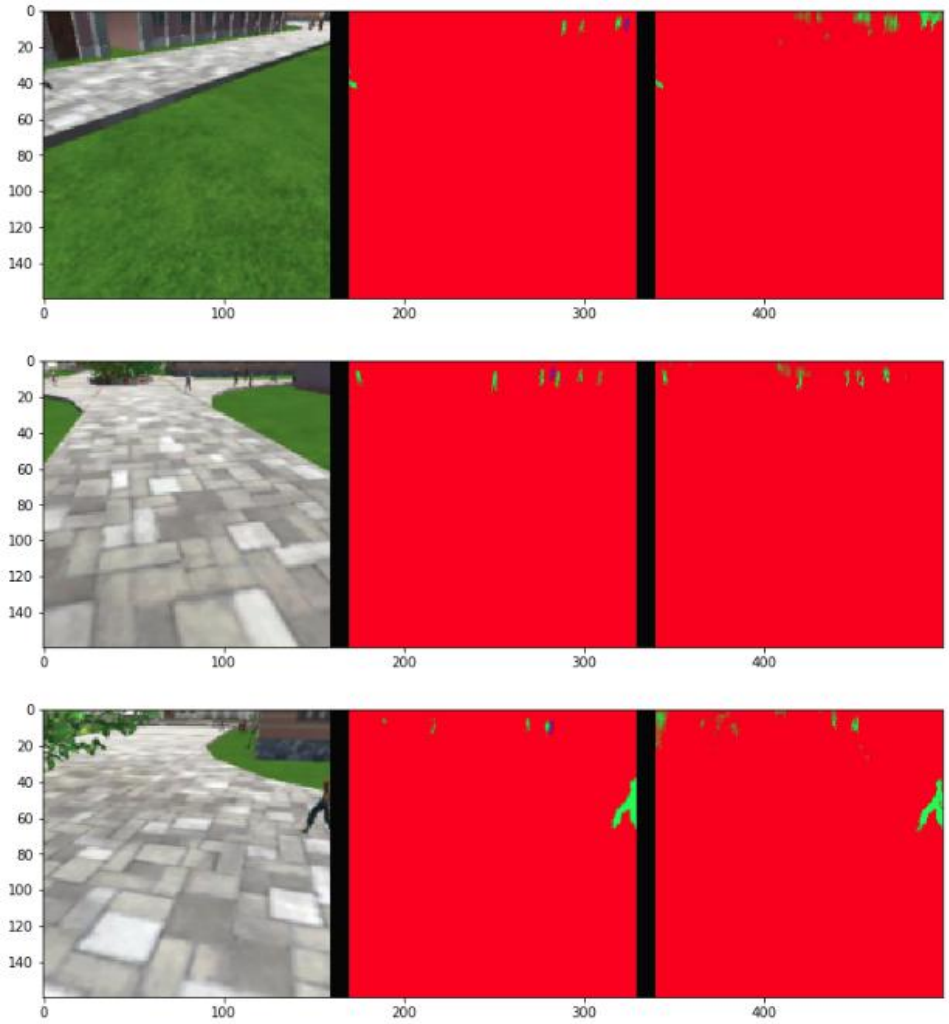


Fig. 2.10 Comparision between ground truth labels and predictions(images while at patrol with hero)

2.2.2 Evaluation

We will use the IoU to evaluate the final score. The basic idea of IoU can be illustrated by the below figure. In particular, IoU is the intersection over union, where the intersection set is an AND operation and the union is an OR operation.

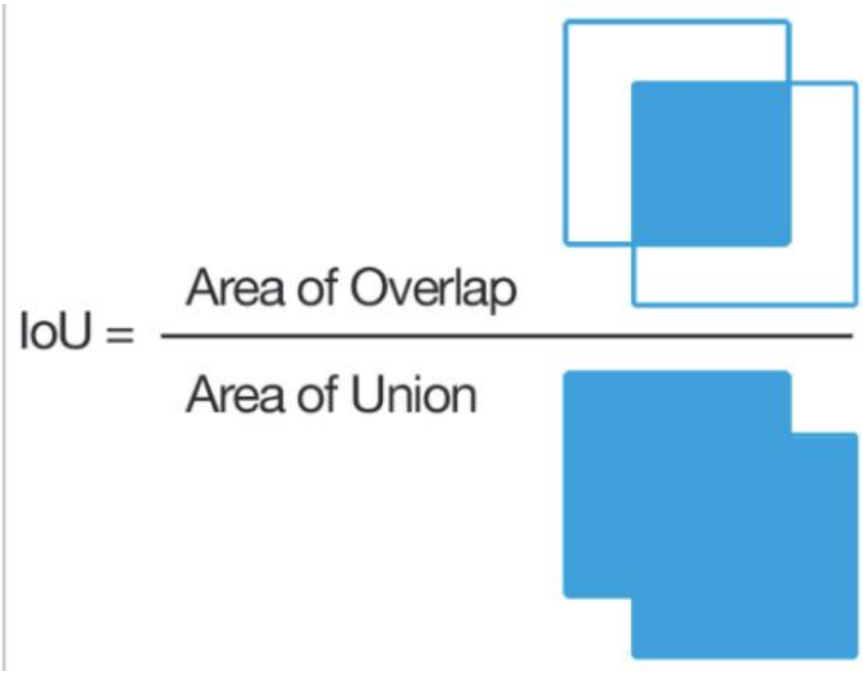


Fig. 2.3 IOU

The IoU score is 0.55074.

3 Conclusion & Discussion

3.1 Conclusion

In this project, we built and trained a FCN and utilized it for moving targets identifying and tracking tasks in a drone simulation environment. The FCN consists of 7 layers, 3 for encoder, 3 for decoder, and 1 for convolutional layer. The last decoding layer serves as the output layer which returns the result. After the FCN has been trained, we evaluated the results and tested it in the simulator.

3.2 Future improvements

1. Recording a bigger set of images;
2. Adding more layers;