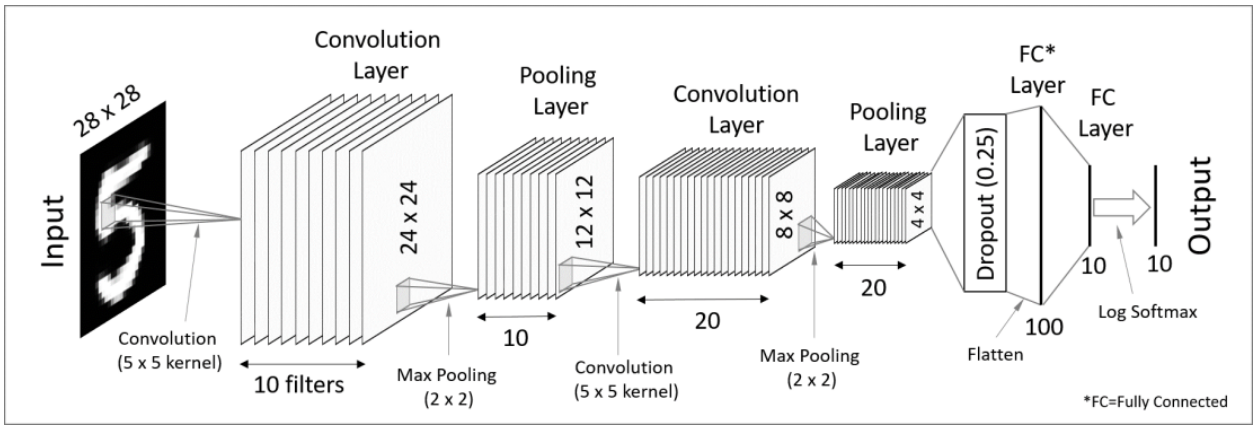
1 Introduction

This report is for “Follow Me” project for Robotics Nano-degree from Udacity. The main objective is to **Design** a Fully Convolutional Network (**FCN**) for semantic segmentation and **Train** the net for a given/recorded dataset from a quad-simulator, which contains mainly of target (woman in red), people, objects (buildings, trees, etc.) and the quad copter, to define the special information of the target and people from quad’ front camera. Then **Test** the model on the simulator.

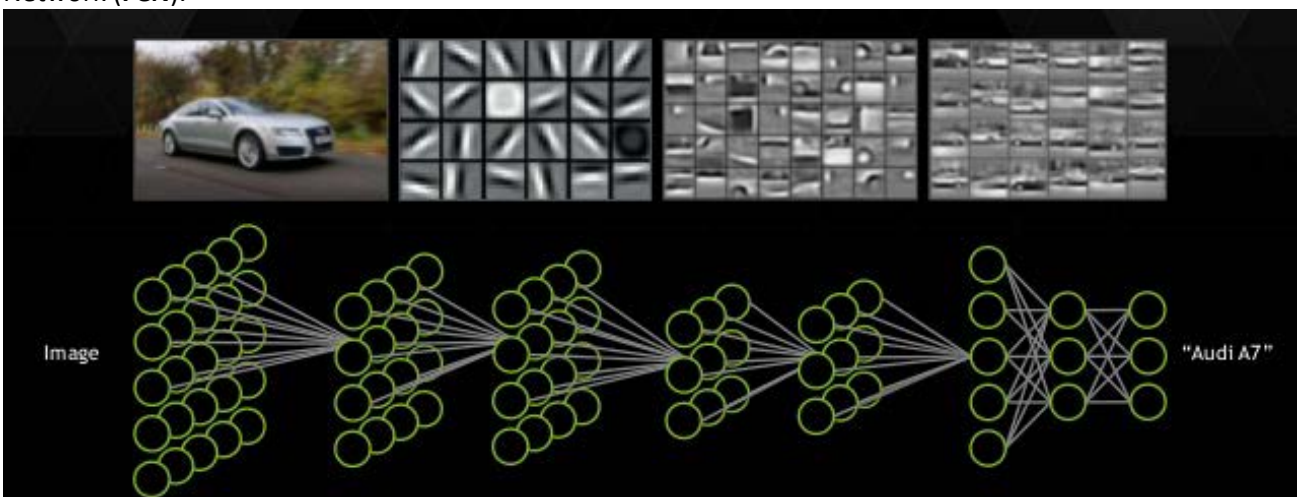
2 Overview on CNN and FCN

Convolutional neural network (**CNN**) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other, For example, the MNIST set.

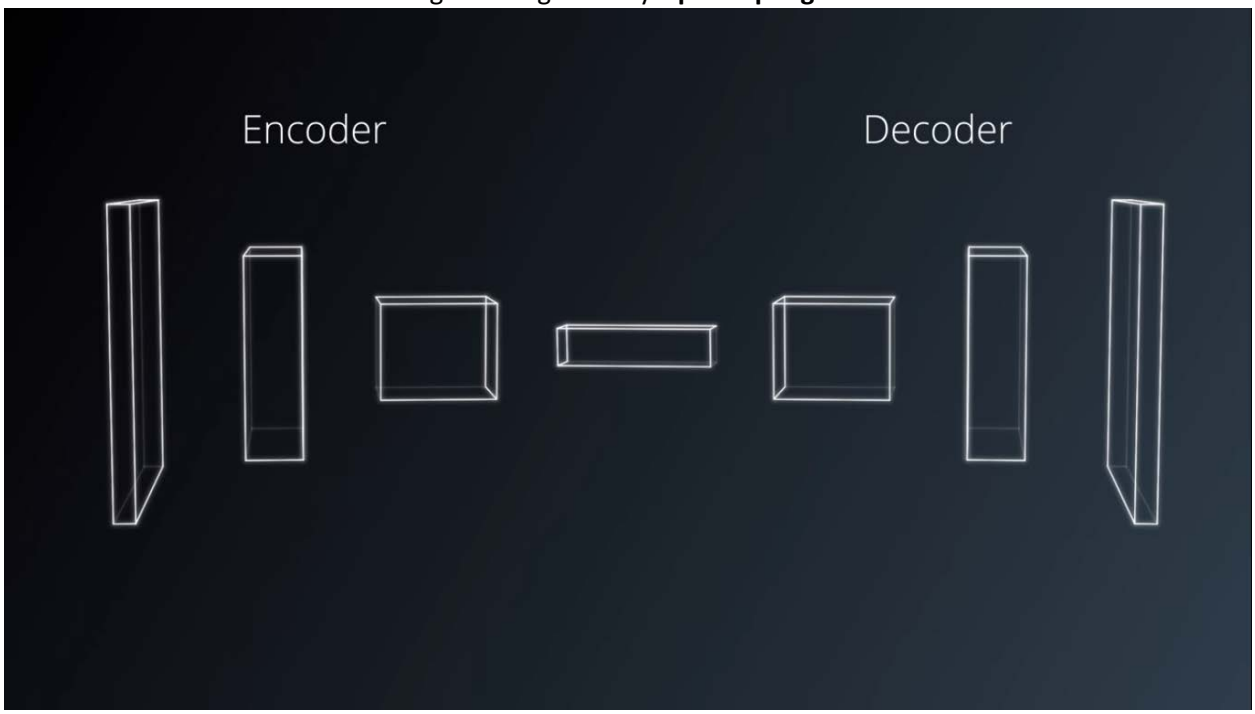


as shown, the net consists of alternating ConvNet and Pooling layers then ending the output by Fully Connected layer (FC), which is used for the classification.

Every ConvNet is responsible for detecting small parts/shapes from the image where the complexity of this parts increases with each following layer. So, the output of the Net is a classification of the input (Car, Dog, Cat, etc.), and Does not answer the question Where is that object in the image. Here comes the role of Fully Convolutional Network (**FCN**).

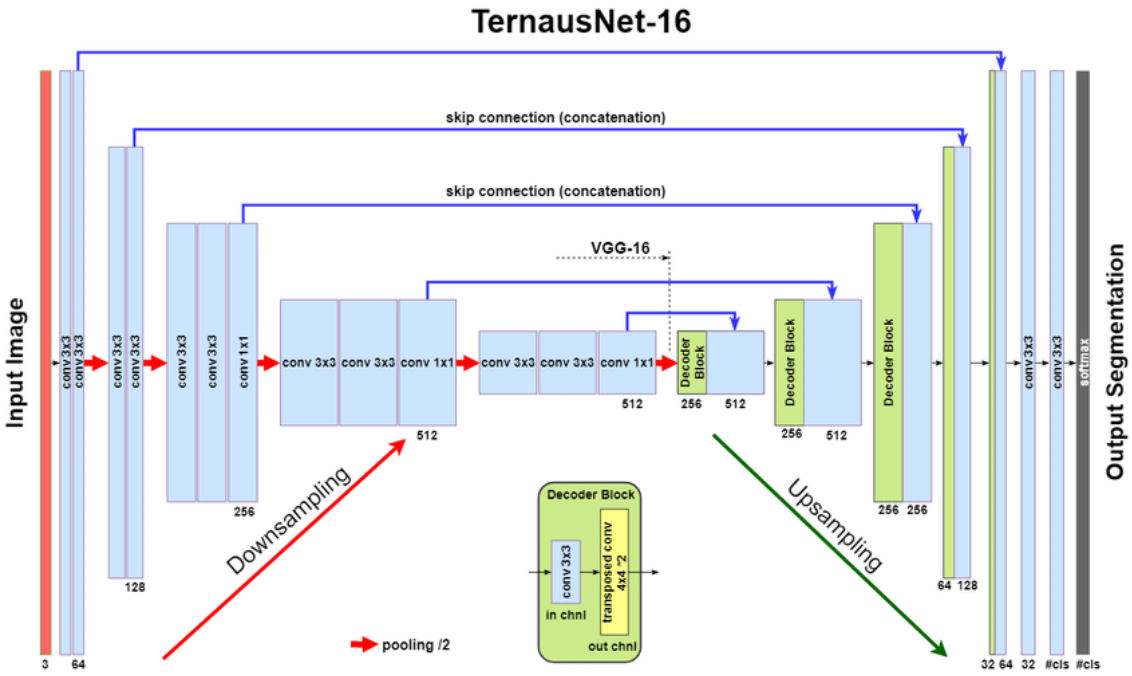


Fully Convolutional Network (**FCN**) is a normal CNN where it has layers ending with an classified layer, this part is called **Encoder**, Following by a Decoder part , which the responsible to obtain the spatial information from the classifier and maintain the original image size by **Up-Sampling** methods.



so, the FCN take advantage of 3 techniques:

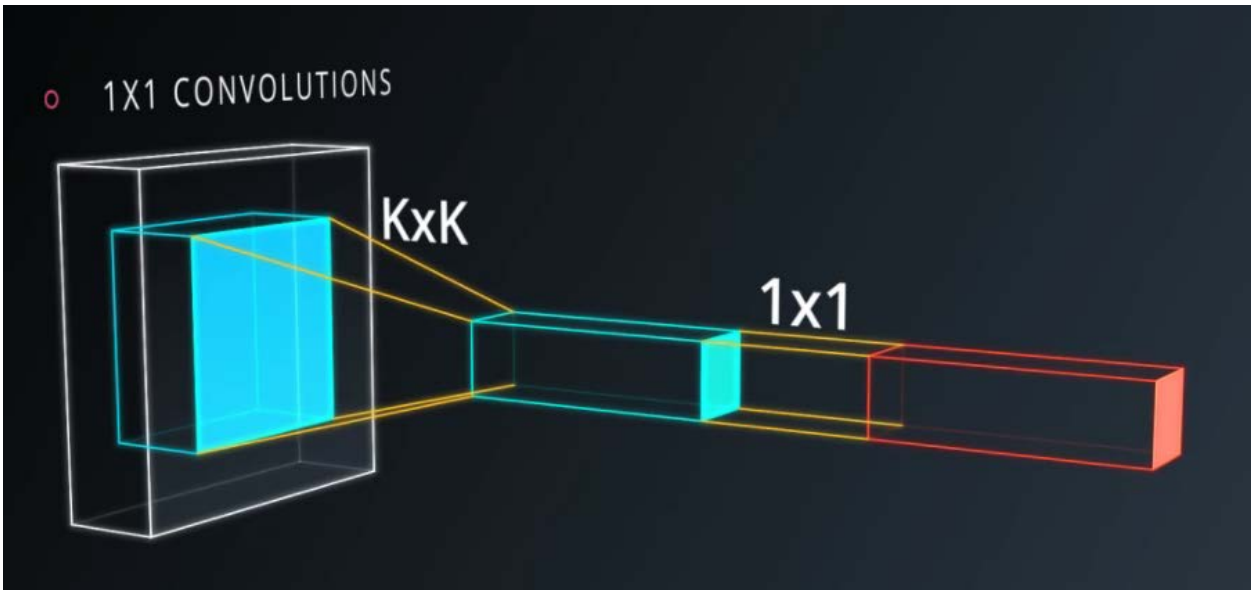
- Replacing FC by 1x1 ConvNet
- Up-sampling
- Skip Connection



We will briefly introduce every technique.

1x1 Convolutions

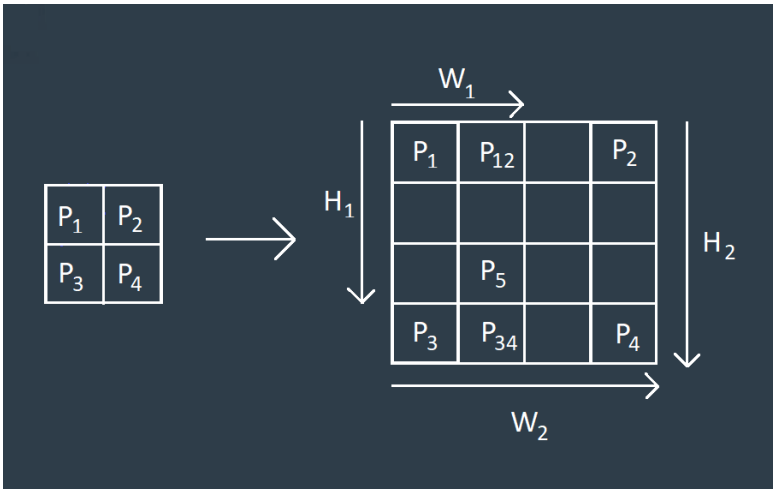
The problem with FC is that their input must be flattened from 4D tensor to 2D tensor. so, this will cause a great loss of spatial information which will not be properly useful of our segmentation. So, 1x1 convolution helped in reducing the dimensionality of the layer. A fully-connected layer of the same size would result in the same number of features. However, replacement of fully-connected layers with convolutional layers presents an added advantage that during inference (testing your model), you can feed images of any size into your trained network.



Up-Sampling

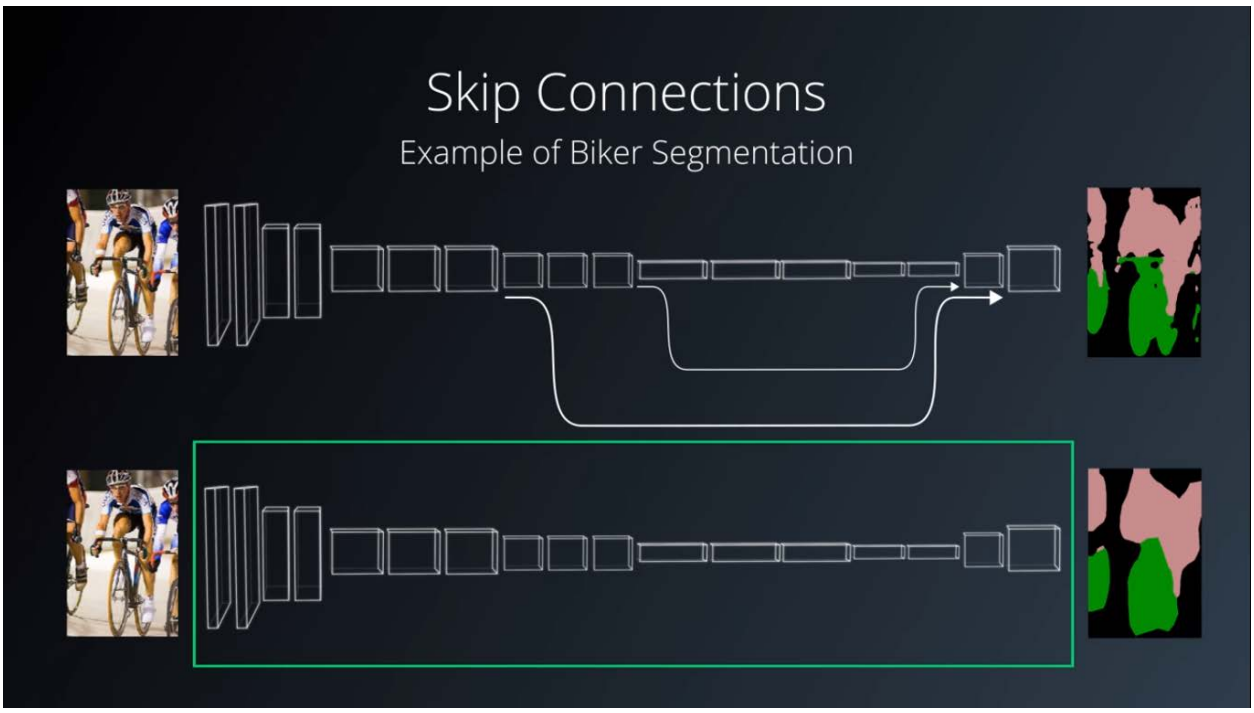
Up-Sampling is the opposite technique used in the encoder (Down-Sampling). The main objective of this layers is to obtain the original size of the image using the info. From the 1x1 ConvNet.

There are many methods for up-sampling and we will be using the Bilinear method. Bilinear up-sampling is a resampling technique that utilizes the weighted average of four nearest known pixels, located diagonally to a given pixel, to estimate a new pixel intensity value. The weighted average is usually distance dependent. Let's consider the scenario where you have 4 known pixel values, so essentially a 2×2 grayscale image. This image is required to be up-sampled to a 4×4 image. The following image gives a better idea of this process.



Skip Connection

This problem here is that the Net is always looking for small features through the layers (Max pooling, down-sampling) which will affect the main overview of the input picture. So, Skip connection helps the Net to be in the main picture.



Another concept used is

Batch Normalization

Batch normalization is based on the idea that, instead of just normalizing the inputs to the network, we normalize the inputs to layers within the network. It's called "batch" normalization because during training, we normalize each layer's inputs by using the mean and variance of the values in the current mini-batch.
Introducing the batch normalization layer presents us with quite a few advantages.
Some of them are:

- Networks train faster – Each training iteration will actually be slower because of the extra calculations during the forward pass. However, it should converge much more quickly, so training should be faster overall.
- Allows higher learning rates – Gradient descent usually requires small learning rates for the network to converge. And as networks get deeper, their gradients get smaller during back propagation so they require even more iterations. Using batch normalization allows us to use much higher learning rates, which further increases the speed at which networks train.
- Simplifies the creation of deeper networks – Because of the above reasons, it is easier to build and faster to train deeper neural networks when using batch normalization.
- Provides a bit of regularization – Batch normalization adds a little noise to your network. In some cases, such as in Inception modules, batch normalization has been shown to work as well as dropout.

2 Project

2.1 Code

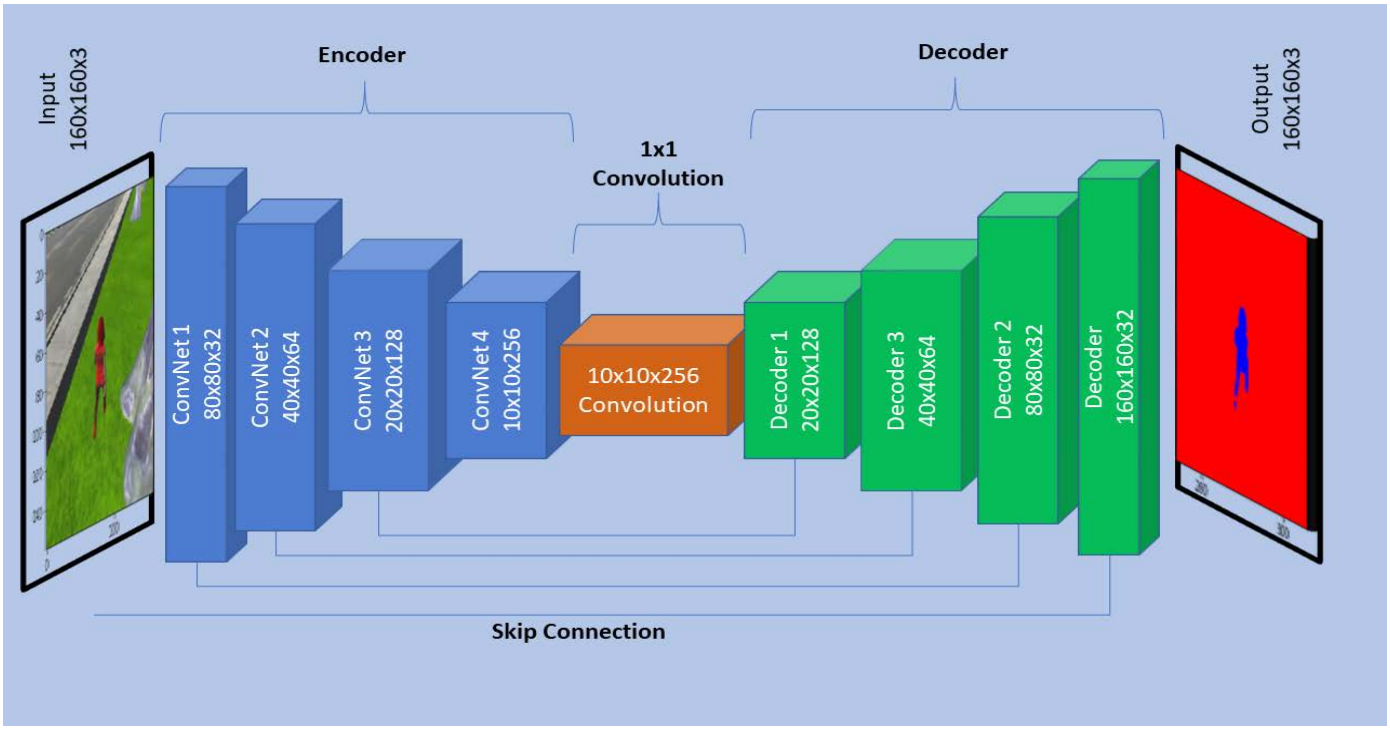
The design of our FCN is mainly consists of 3 elements

1- **Encoder block** : using the `separable_conv2d_batchnorm()` function
2- **Decoder Block :** The decoder block is comprised of three parts:
   - A bilinear up-sampling layer using the upsample_bilinear() function. The current recommended factor for up-sampling is set to 2.
   - A layer concatenation step. This step is similar to skip connections. You will concatenate the up-sampled small_ip_layer and the large_ip_layer.
   - Some (one or two) additional separable convolution layers to extract some more spatial information from prior layers.
3- **Model: using** the encoder and decoder blocks ready, go ahead and build your FCN architecture!
   There are three blocks:
   - encoder blocks to build the encoder layers.
   - a 1x1 Convolution layer using the conv2d_batchnorm() function. Remember that 1x1 Convolutions require a kernel and stride of 1.
   - decoder blocks for the decoder layers.

2.2 Design

Here is an image illustrating my architecture of the network

First, I used only 3 ConvNets and for improving the model I used 4 layers



2.3 Hyper parameters

For tuning the training model, we use these parameters as high periptery to change:

- **learning_rate :**
  - **the value wich define the percentage of wights / bias that would be changed**
  - **mostly I am using = 0.001**
- **batch_size**: number of training samples/images that get propagated through the network in a single pass.
  - For the most of the training sessions the value is set **64** and usually **128**
- **num_epochs**: number of times the entire training dataset gets propagated through the network.
  - Was set usually as 50 ,100 ,200
- **steps_per_epoch**: number of batches of training images that go through the network in 1 epoch.
  - **Since I am working on different dataset (3000 : 4300) images so the value is usually between 50 : 100 (batch_size /train dataset samples)**
- **validation_steps:** number of batches of validation images that go through the network in 1 epoch.
  - **Since I am working on different dataset (1000 : 1500) images so the value is usually between 20 : 50 (batch_size / dataset samples)**
- **workers**: maximum number of processes to spin up. This can affect your training speed and is dependent on your hardware.
  - **Since I am working on the GPU provided in Classroom. This parameter is not effective**

2.4 Dataset

I worked on 2 different datasets:

- **Udacity_data:** Given ready
- **Recorded:** from the simulator, it was first about 2000 training images and for improving the model I was adding more features and samples to reach up to 4300 training images and 1300 validation images

Training and Results:

I have trained the model for more than 15 times with different datasets and Hyperparameters Here, I will mention only the Highest Results obtained.

Udacity dataset

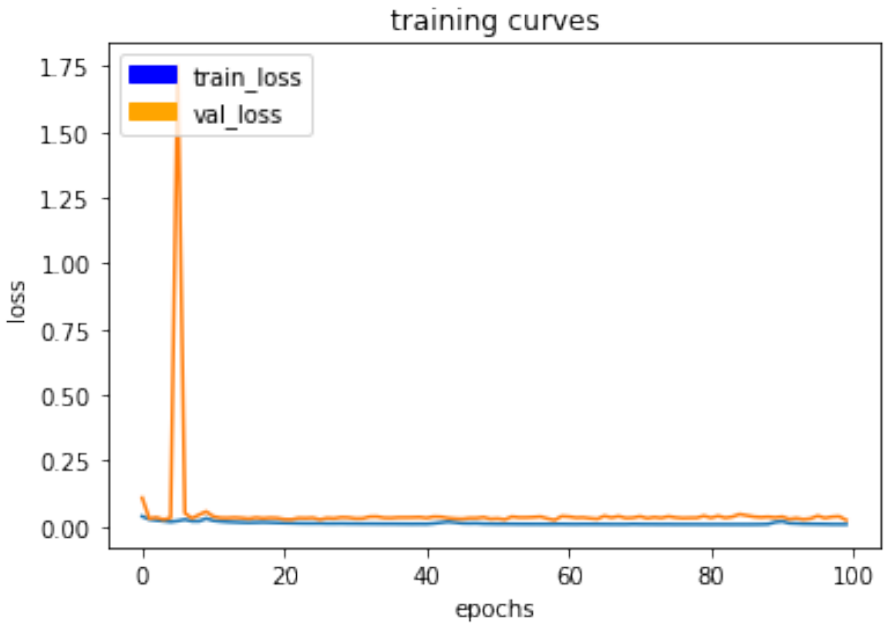
Parameters:

```
learning_rate = 0.001
batch_size = 64
num_epochs = 100
steps_per_epoch = 66
validation_steps = 25
workers = 20
```

Training Curve:

Losses:

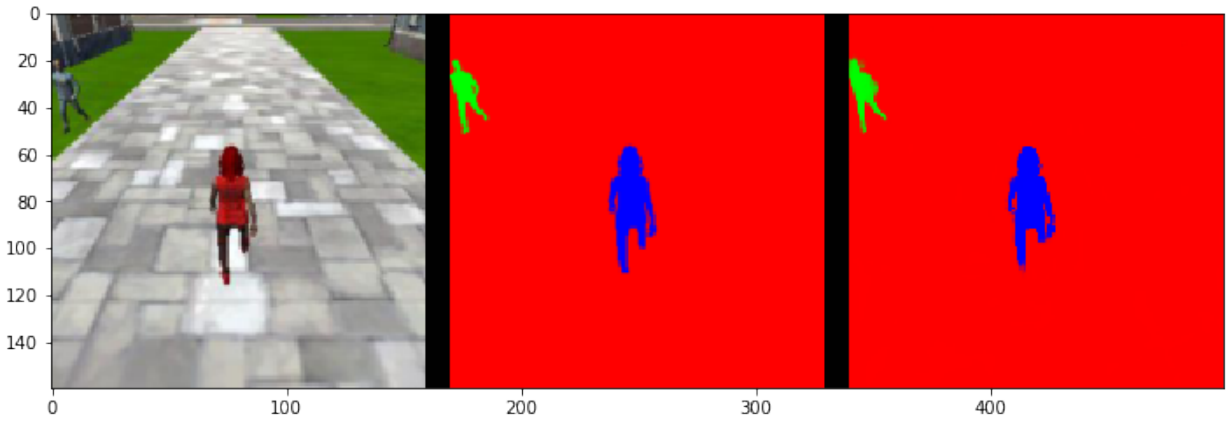
```
- loss: 0.0080
- val_loss: 0.0240
```



Prediction and Evaluation

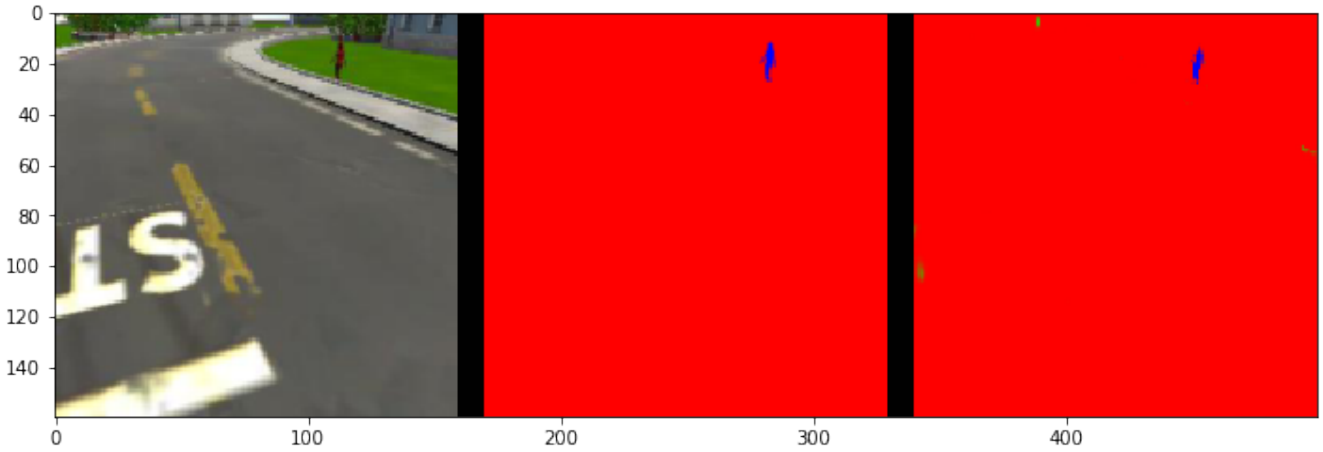
The evaluation is performed on 3 Different datasets

- **following_images:** : Test how well the network can identify the target while following them.



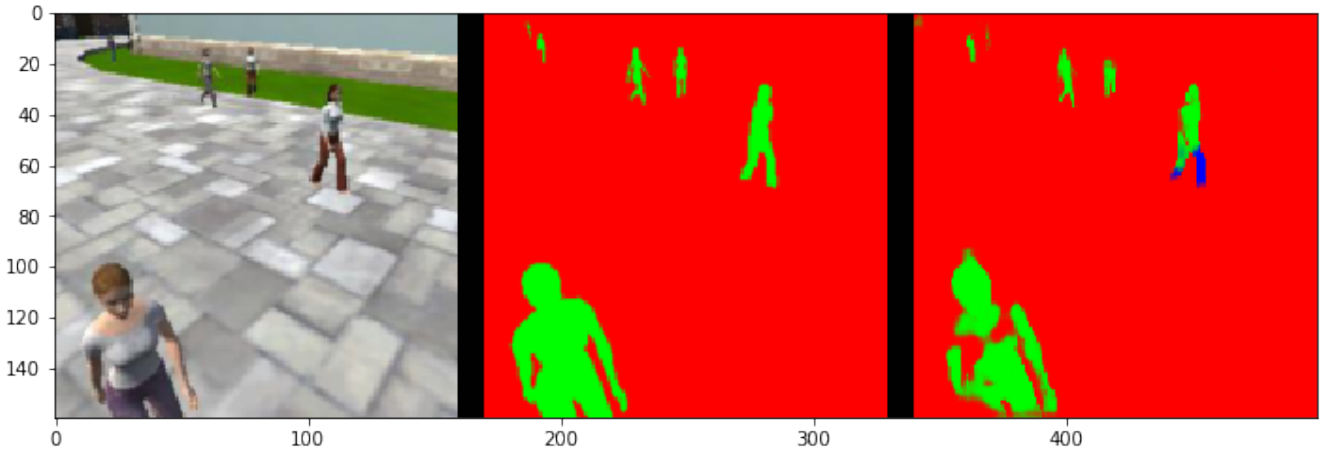

```
number of validation samples intersection over the union evaluated on 542
average intersection over union for background is 0.9961068516336401
average intersection over union for other people is 0.3699227214933152
average intersection over union for the hero is 0.9298835730957277
number true positives: 539, number false positives: 0, number false negatives:0
```

- **patrol_with_target**: Test how well the network can detect the hero from a distance.




```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9966528441048096
average intersection over union for other people is 0.44073008834931615
average intersection over union for the hero is 0.18122780965230473
number true positives: 118, number false positives: 1, number false negatives:
183
```

- **patrol_non_targ**: Test how often the network makes a mistake and identifies the wrong person as t he target.




```
number of validation samples intersection over the union evaluated on 270
average intersection over union for background is 0.9889786487635716
average intersection over union for other people is 0.7742028697941806
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 41, number false negatives: 0
```

Score:

Using the concept of IoU the Final score is `0.413832300717`

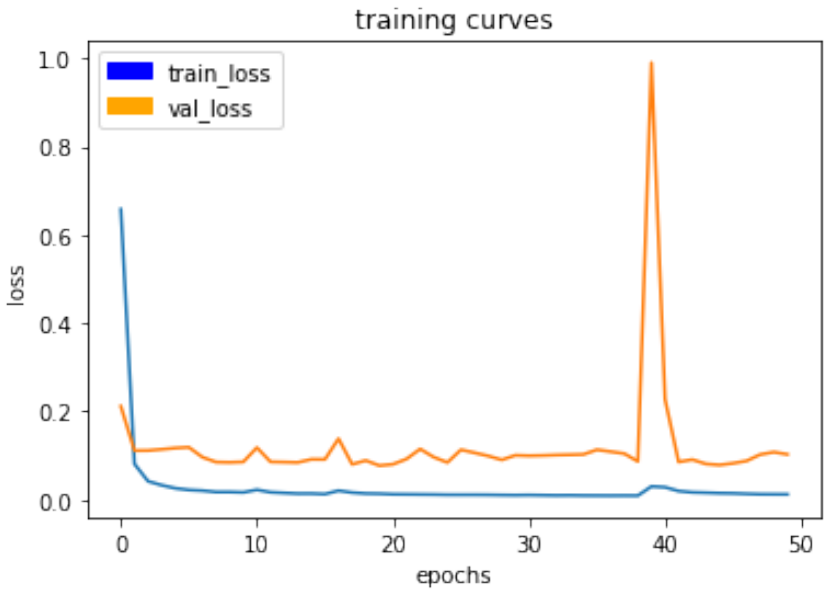
****HTML version of `model_training.ipynb` notebook called 'model_training_Udacity'**

Recorded Dataset

Parameters:

```
learning_rate = 0.001
batch_size = 64
num_epochs = 50
steps_per_epoch = 80
validation_steps = 25
workers = 12
```

Training Curve:



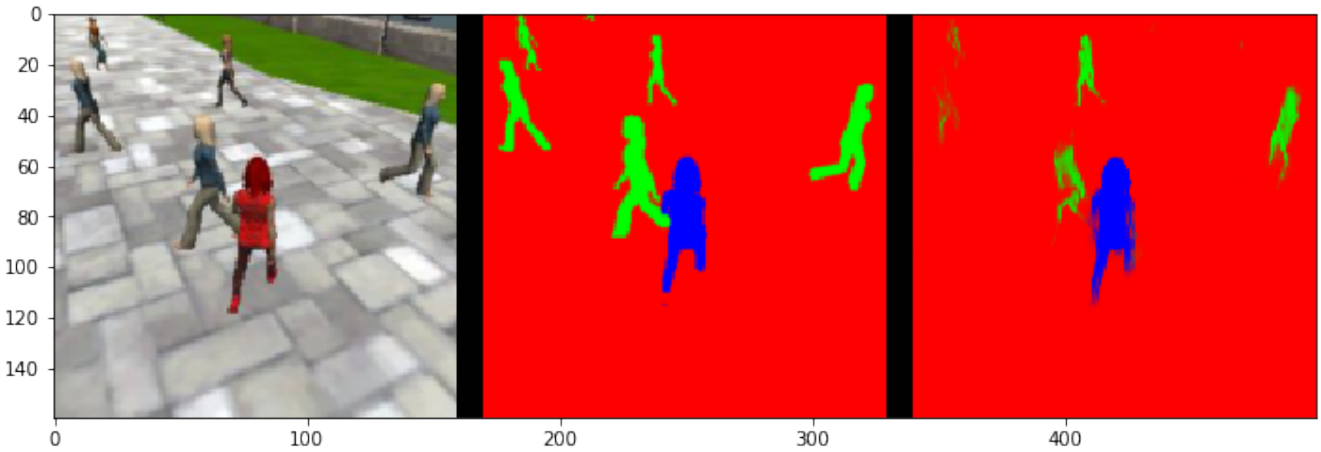
Losses:

```
- loss: 0.0119
- val_loss: 0.1026
```

Prediction and Evaluation

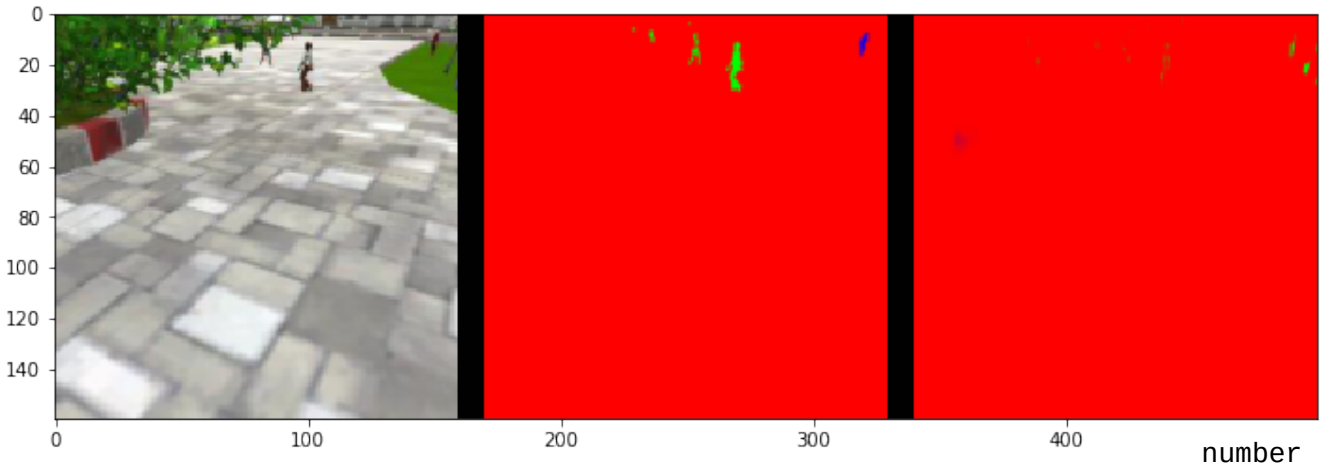
The evaluation is performed on 3 Different datasets

- **following_images:**



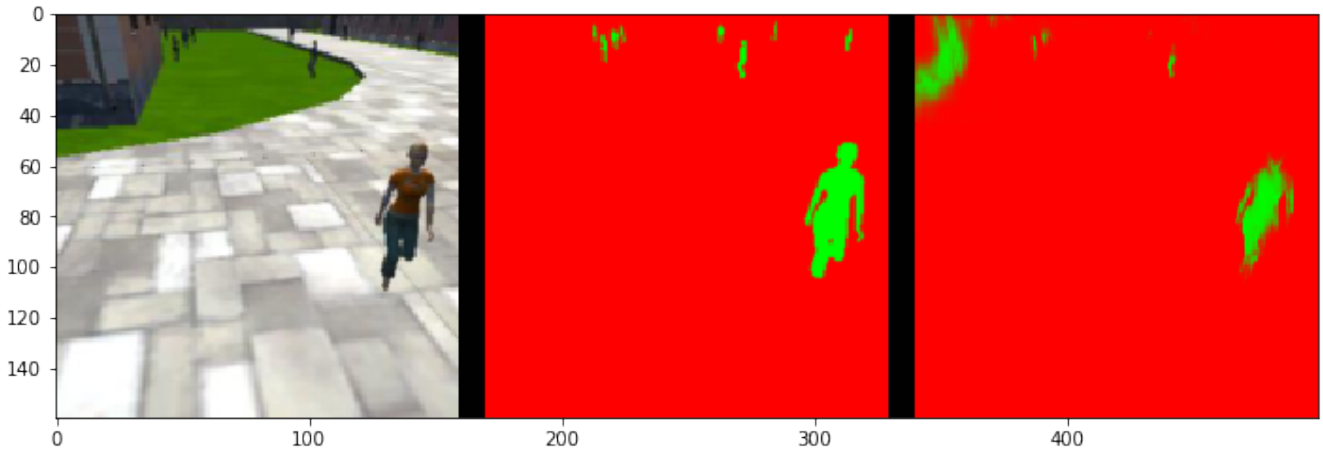
```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.9865965981954056
average intersection over union for other people is 0.13038546190738
average intersection over union for the hero is 0.8521054743658092
number true positives: 539, number false positives: 0, number false negatives
: 0
```

- **patrol_with_target**:




```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.9883641503479935
average intersection over union for other people is 0.17131394659455274
average intersection over union for the hero is 0.1147582733196874
number true positives: 79, number false positives: 1, number false negatives: 2
22
```

- **patrol_non_targ**:



```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.9738482023986998
average intersection over union for other people is 0.4716841799471474
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 17, number false negatives: 0
```

Score:
Using the concept of IoU the Final score is `0.348206174866`

***HTML version of** `model_training.ipynb` **notebook called 'model_training_my_dataset'**

4 Improvements

- **An Important note**:
  The FCN architecture used here can be used to classify other objects such as Cars, Cats, Dogs, etc. but th ere is fear overfitting could be reduced by using regularization techniques like Dropout.
- Further improvement for the segmentation is adding more and more of datasets for the objects that the quad might encounter in the environment. Since, the network don't try to classify them.
- Adding more layers to the net would be very useful.
- Adding more convNets layers with Decoders to find more attractive spatial features for segmentation.
- Tuning the learning rate and the number of epochs will also be very effective.