Project FollowMe Writeup

Introduction

This writeup means to explain all the necessary technical steps to complete the Follow Me project. It is separated into parts : neural network architectures, training process, limitations of the final solution.

Neural Network Architecture

In the big picture, the network is divided into two parts encoder and decoder. Each part would have multiple layer of convolutional networks. The encoder would simply scan through the network and extract all the pixels details information at different levels and pass it on to the decoder to actually combine those spatial information to classify different type of objects in the pictures.

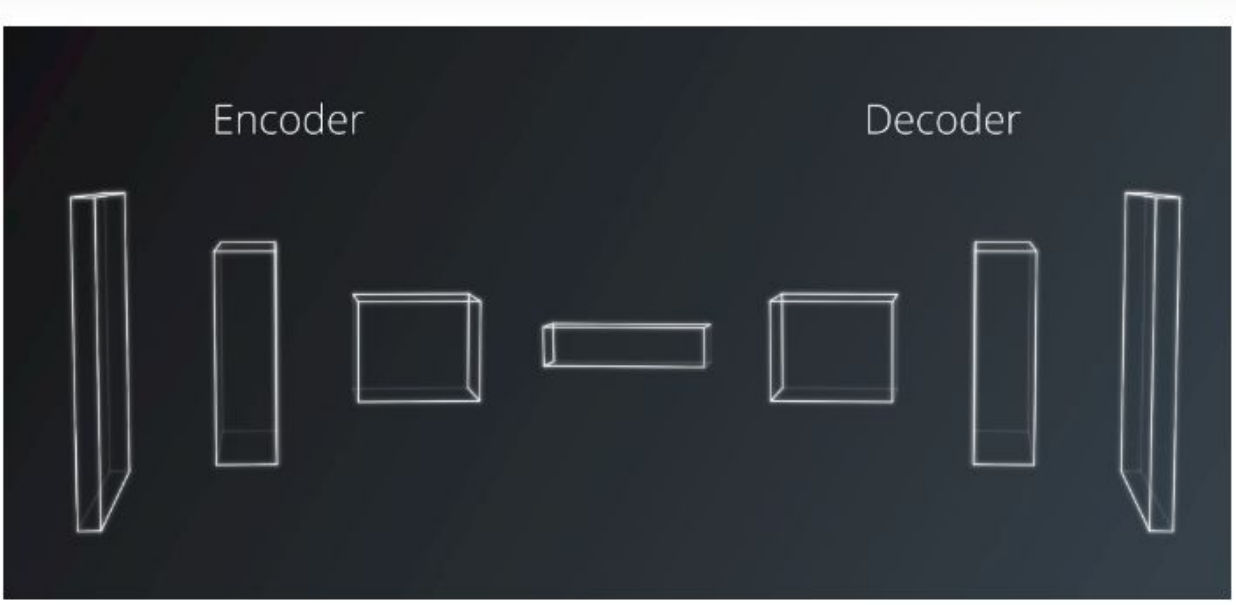

Figure 1 : Overview of Fully Convolutional Neural Network

In order to train the drone simulator to follow the hero object, we use the modern neural network architecture called Fully Convolutional Networks. The reason for using this architecture instead of the regular convolutional network is that fully convolutional network can help preserve the spatial information of the image. Regular convolutional neural network with the last layer as fully connected layer is only good for classification task rather than locating different objects in the same image. We actually replace the fully connected layer with a deep 1x1 convolutions layer which helps to reserve the spatial information.

In order to train the network more efficiently, we apply the technique called Separable Convolutions for the encoder. It means instead of having a specific n number of kernel of size mxm traverse each layer of the image, we will only have one kernel to traverse each of the layer

to produce a set of feature maps and those feature maps would be run through by a set of n 1x1 convolutions. This helps to greatly reduce the number of train parameters which helps the network to be trained more efficiently. It also adds benefit of reducing overfitting due to the smaller number of parameters.

On the other hand, we can see in Figure 1, that decoder will upsampling the information pass through from the encoder layer. In order to fulfil that we apply the technique all bilinear upsampling. This technique simply make use the values of four nearest known pixels in order to calculate new pixel value. Lastly, we also add one to two separable convolution layers to extract some more spatial information. Also, in the decoder block, we apply layer concatenation technique which allow information from a specific encoder flow straight through to a non-adjacent encoder layer in order to keep some of the original information which would get lost while the encoder layers are extracting specific features.

With all that information, I come up with the final architectures for my network after quite some experimenting and fine tuning :

- **Encoder:**
  - 5 layers of separable convolutions with depths of 24, 48, 96, 192, 384 respectively.
- **A 1x1 convolution layer of depth 768.**
- **Decoder:**
  - 5 layers of separable convolutions with depths 384, 192, 96, 48, 24 and concatenate with the encoder layer of same size.
  - 2 layers of of separable convolutions.

The network is implemented using Keras Separable Convolution and Bilinear Upsampling APIs with Tensorflow backend . Also, each Separable Convolution is connected to a Batch Normalization layer to improve training speed and accuracy.

Training Process

It was an empirical process of data collection and fine tuning parameters to be able to come up with the final architecture in previous section.

I first start with a simple network of only 3 layers for both encoder and decoder. Also, I start with only one layer of separable convolutions for the decoder.

I start with the hyper parameters as below:

- Learning rate : 0.0005
- Batch Size : 200
- Steps per Epoch: 100
- Number of Epochs : 20
- Validation Steps : 50

After each training process, I examine the training graph and the trends of loss and validation accuracy. I keeps increasing the epochs if I see the network is in the good trend of decreasing validation loss. Otherwise, if the training data show that the training loss and validation loss is not decreasing for a large period of epochs, I will go back and change the architecture to make the network deeper or increase the number of filters.

Also, the comparison between the network prediction and sample also gave me some good information about the current weaknesses of the network such as detecting hero from far away, recognizing random people is not the hero. Therefore, I started manually collect a lot of additional data with the hero from far away at different angles to enhance the detection of the hero at a long distance. Also, I collected a lot of data with no hero and just random people walking in different area of the city.

Aside from that, I also use the open-source library Augmentor to generate another 10000 samples of training by flipping and distorting images from the original set.

Finally, after many training iterations and fine tuning the network, I am able achieve the final grade score at 0.455 with the following hyper parameters:

- Learning rate : 0.0005
- Batch Size : 100
- Steps per Epoch: 100
- Number of Epochs : 150
- Validation Steps : 50

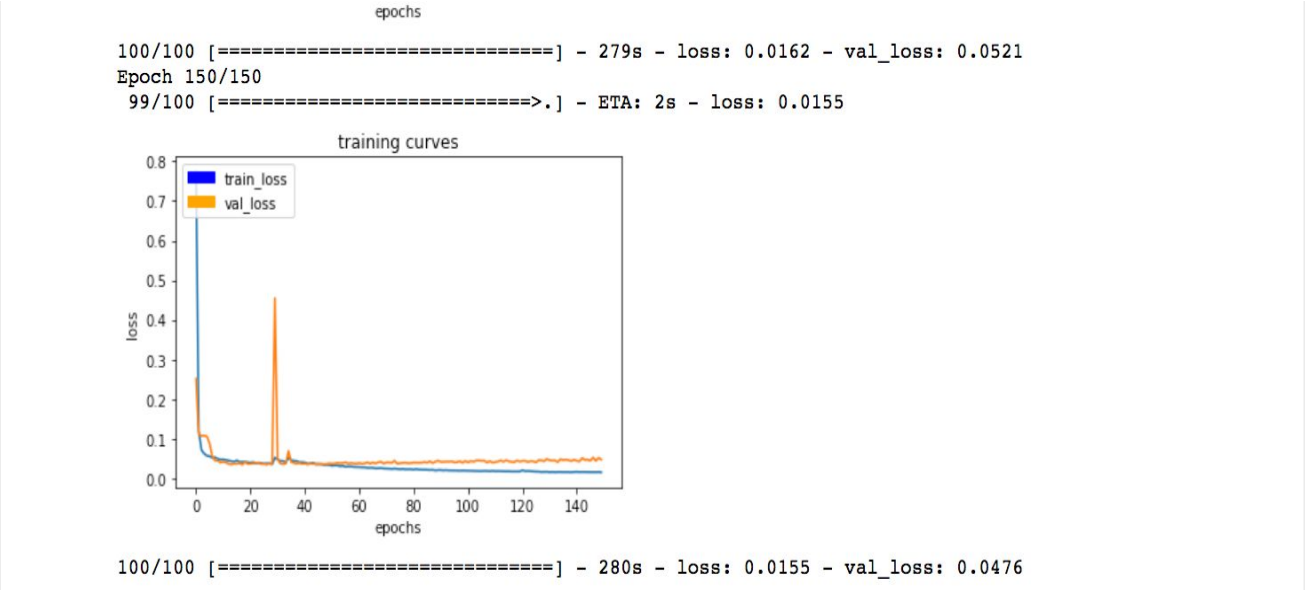

Figure 2 : Final training results

Limitations

The final network still has a lot of limitations. Although I had added a lot of images contained hero from far away with various angles, the results are not tremendously improved as the number of false positives detection is still high. I think a deeper network might help to combine small pixels to segment better. To be able to work well with other object rather than human, the network will need to be retrained with a dataset that contains the desired object to be detected. A good idea is to use the encoder from a well-known pretrained convolutional

network like VGG16 and removed the fully connected layer, then we can use this to train the encoder which might help to improve the training speed as well as accuracy.