Udacity's Software Robotic Program
Deep Learning – Follow me Project
Carlos Sanoja

Introduction-.

For this project, the core idea is the design and tunning of neural network's architectures. Specially, Fully convolutional networks (FCN) to detect and track an target objective. Fully convolutional indicates that the neural network is composed of convolutional layers and tries to learn representations and make decisions based on local spatial inputs.

Network Architecture-.

Rubric Point :

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.

The network architecture chosen is Fully Convolutional Network (FCN for short) because the idea is to implement Semantic Segmentation to find a target. This architecture works better handling tasks like images classification. And specially, allow us to answer the questions: What and Where a certain object is in space. That's the main difference between FCN and Fully connected layers who remove any spatial information in the image and that's why we are not going to use it (*Rubric point-. The student demonstrates a clear understanding of a fully connected layer and where/when/how it should be used*).

The FCN used for this project can be represented in three parts: Encoding, connection stage, and Decoding.

- **Encoding:** is a convolution network that reduces to a deeper 1x1 convolution layer. Basically, this stage extract features via multiple layers that find simple patterns from the first layer and goes deeper learning more complex pattern preserving spacial information from the image. This is done by separable convolutions. A convolution filter is called separable if it can be broken down into the convolution of two or more filters. This operation suggest a cheaper way to perform it. The main task of separable convolution is to independently look into spatial dimensions and channel dimension. For my architecture, I have to layers in this stage. Both of them perform separable convolution, the output size is same as a regular convolution layer but, this method has more filters (64 and 128 respectively). This is why, separable convolution give us a better performance. It split up the original image into 64 sub-batches with an stride of 2 and thus, split it up again applying 128 filters given as result, the input for and 1x1 convolution. The procedure is to traverse every channel with 1 kernel and then apply 1x1 convolution for every feature map.

  Another important sub stage is normalization. For this project was implemented separable convolution with batch normalization. This algorithm is based on the idea that we can normalize the inputs to layers during training by using the mean and variance of the values in the current mini-batch. Accordingly with theory, that will allow us some advantages: train faster the network,regularization, high learning rates options and create deeper networks.
- **Decoder:** The idea with this stage is up-scale the encoder output back to the same dimensions as the input image through multiple layers. This part uses three techniques that allow us

accomplish that objective: bilinear up-sampling, concatenation layers and separable convolutions. Following with classroom definition "Bilinear upsampling 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." Basically, it get back to an older layer dimension but we are "estimating" an pixel value and cause an information missing (finer details). However, this algorithm helps to speed up performance. Here, is important to note that the stride parameter is who will determine the size of the new layer. Concatenating two layers, the upsampled layer and a layer with more spatial information than the upsampled one, works like "skip connections". Skip connections are a great way to retain some of the finer details from the previous layers as we decode or upsample the layers to the original size. Then, basically we are taken some information from the output of our two layer of the encoder and passing them as input for the reconstructing one on the decoder to obtain some extra real information for the upsampling. As was said in video lessons, "One added advantage of concatenating the layers is that it offers a bit of flexibility because the depth of the input layers need not match up unlike when you have to add them. Which helps simplify the implementation as well". Finally just as before, we apply separable convolution to obtain the upsampled layer reconstructed. This time we apply it because it can often be better to add separable convolution layers after concatenated ones for your model to be able to learn those finer spatial details from the previous layers better.
(*Rubric point-. 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.*)

- **1x1 Convolution:** This layer behaves in a similar way of fully connected layer but it keeps the spatial information and it's the connection between encoder and decoder. As was suggested in video's class, 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. Also, the main advantage is, ***with 1x1 convolution we don't loss spatial information of the input images' features.*** *(Rubric point-. The student demonstrates a clear understanding of 1 by 1 convolutions and where/when/how it should be used)*

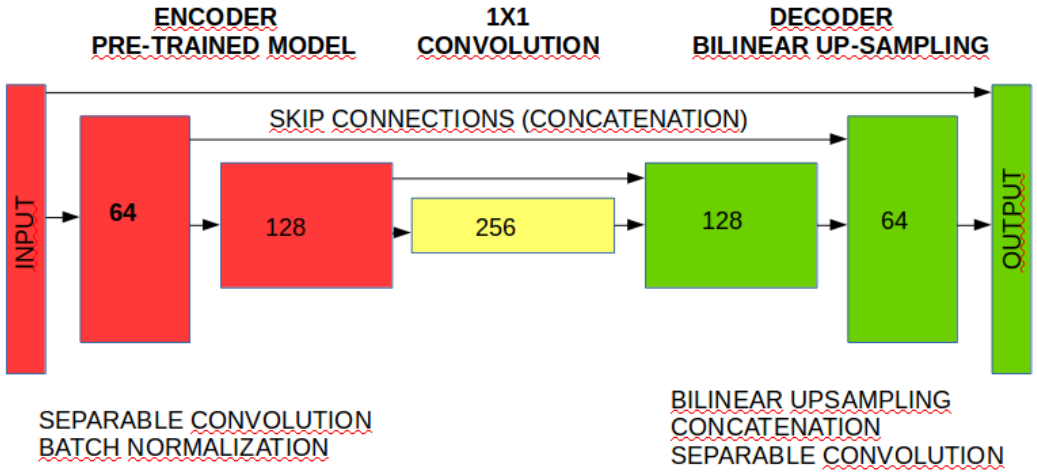

Figure 1.- FCN Architecture

Neural Network's Parameters-.

The hyperparameters are:

batch_size: number of training samples/images that get propagated through the network in a single pass.
num_epochs: number of times the entire training dataset gets propagated through the network.
steps_per_epoch: number of batches of training images that go through the network in 1 epoch. We have provided you with a default value. One recommended value to try would be based on the total number of images in training dataset divided by the batch_size.
validation_steps: number of batches of validation images that go through the network in 1 epoch. This is similar to steps_per_epoch, except validation_steps is for the validation dataset. We have provided you with a default value for this as well.
workers: maximum number of processes to spin up. This can affect your training speed and is dependent on your hardware. We have provided a recommended value to work with.

I followed this logic:

- try to kept the learning rate low as possible (However, I am using batch normalization and allow to set higher values)
- To set the steps per epoch, I took the total number of images for training and divided it for the batch size. I tried to keep this value close as possible. IF this values get lower, the number of epochs should increase.
- For validation steps, I tried to keep as close as possible to batch size.
- workers, dependent of CPU capacity. For my computer I took 2 but, for AWS I took 4.
- I kept the number of epochs as my free variable.

Note: The results are explained in the Discussion part.

Discussion-.

Training Process

FCN are very computational expensive to train. That was showed in my firsts attempts to train the network for the Semantic segmentation laboratory. I have a CPU with i5-560M 2.66G and 4GB RAM. I attempted to run 10 epochs with 100 steps per epoch and a batch size of 32. It tooks over 14 hours and I just got 46% as performance measure. When I came to the project I felt terrible because the dataset had been increase from 1400 images to 12000 and the response from amazon for AWS didn't arrive.

My first option was take the output model from the lab (knowing that its performance was terrible) and try to make a prediction, the result was not too bad, 34%. For my second attempt I try to train the network on my CPU. I took this time 250 steps per epoch (following my logic) y try to stay closely as possible of 400) I ran it and that was suicide. I left him train over 18 hours and it just train 97/250 steps of 1/8 epochs. By the time I was frustrated, the response from AWS arrived I was capable of train my network over an Amazon's GPU.

Quickly than immediately, I could see the difference in power. The same configuration I have tried took over 25 minutes to be perform. After training I just download the models into my computer.

Results

My first attempt with GPU support, as I said before, was not to bad. I was decided to keep my configuration architecture and tune my hyperparameters first. The “a model” improve the early pre trained model and the IOU score was 0.39 (Yes, to close but not enough).

For my second attempt (b_model), I keep all my older parameters (learning_rate=0.001, steps_per_epoch=250) and decide to increase the number of epochs to 12 (my free variable) and the validation steps to 39. The results of IOU were 0.43, by this time I felt better. The final result of training can be seeing in figure 2. When I saw those peaks, I though that my data was over-fitting but after a while the validation loss function get down again.

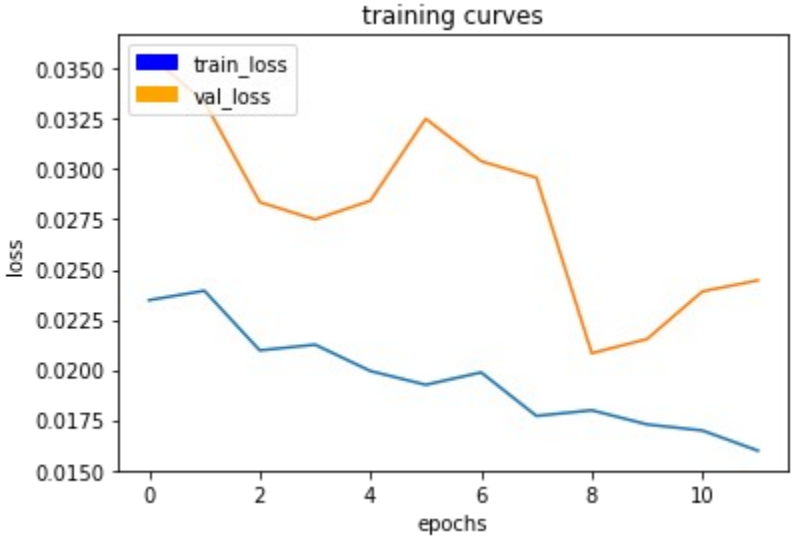

Figure 2-. Final training curve of model b. Loss: 0.0160 - Val_loss: 0.0245

Also, the measure of performance: “average intersection over union for the hero is 0.913896550944634” for Scores for while the quad is following behind the target, was too high which means a lot of increase of performance for my model.

Finally, i decide to make another tune operation (model c), his time, getting close (following my logic) to 400 on steps_per_epoch. This time, the parameters were: epochs = 4, steps_per_epoch=450. The results can be observed in figure 3. This time, the IOU score was just 0.37 which show a worst behavior.

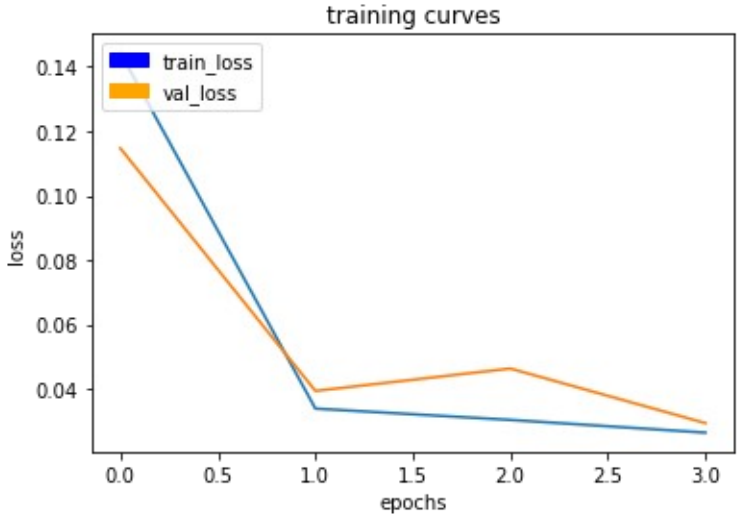

Figure 3-. Final training curve of model c. Loss: 0.0264 - Val_loss: 0.0294

Future Enhancement-.

1. As I was not capable to prove my model on simulation because the simulation doesn't work in my computer I I didn't have much options, the first future enhancement its assist the performance and behavior of the drone following its target.
2. Improve the tune parameters maybe trying some search algorithm to find a better relationship between them.
3. Increase the data for training specially for "other people" and "hero" because as I was not capable to run the simulator I just took the data that was given for us. That could prevent over-fitting and improve my final score.

Conclusion-.

I have completely done this project obtaining an accuracy of 0.43 on IOU. I really enjoy this project as an introduction to deep learning because I have learned the basics structured and how can I switch them to obtain the desired goal. Also, I liked the GPU adventure because I have never used one of them. Its capabilities are huge and for training an network are really important to improve your work time and use it in performing analysis of the network behavior than wasting time waiting for results.

Furthermore, even though I was not capable to see my model in action I am going to find a way to get back to this project an certainly improve the score.

Finally, it's important to note that this network could be used to track other objects, such as dogs or cats. However the training data must change and use images for those objects. It could be interesting if your drone could choose what object to track, that would require extra training and complex architectures. In addition too, for me its really interesting the architecture where we can add additional information like depth to create 3d model adding to another piece for the decoder stage.