UDACITY

‹ Return to "Self-Driving Car Engineer" in the classroom

DISCUSS ON STUDENT HUB

Semantic Segmentation

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Excellent work on the project! Your results look great! 😄

Play around more if you'd like and see how you could improve your results, or try them out on a different or more difficult dataset (maybe https://www.cityscapes-dataset.com/).

I hope the review helped you. If you feel there's something more that you would have preferred from this review please leave a comment. That would immensely help me to improve feedback for any future reviews I conduct including for further projects. Would appreciate your input too. Thanks!

Congratulations on finishing the project! 😄

Build the Neural Network

The function `load_vgg` is implemented correctly.

Great job correctly implementing `load_vgg`!

The function `layers` is implemented correctly.

Nicely done! Well structured code here. Especially good for implementing kernel initializer *and* regularization. Did you also try the Xavier Initializer? Try it out and see if it helps with your results or not.

You might already be aware of these but here are couple of additional resources that I have found helpful for weight initialization and regularization -

- http://cs231n.github.io/neural-networks-2/#init
- http://cs231n.github.io/neural-networks-2/#reg

Question for you to think about -

Do you think you need the Kernel/Weight Initializer for your conv2d_transpose layers? Since it's just upsampling an existing layer, is weight initialization important? Think about it! :)

The function `optimize` is implemented correctly.

Good work! Especially for adding the regularization loss (many students forget to do that) !

Try to experiment more with your regularization constant. Try out values greater than 1, far greater than 1 etc. Usually, you should expect better results with a value closer to 0.01 in this case but that's not necessary for all cases.

Once you are done with the project, I would highly recommend trying to play around with the beta1 parameter for the Adam Optimizer as well!

If you'd like to read more on Adam Optimizer - https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

The function `train_nn` is implemented correctly. The loss of the network should be printed while the network is training.

A good, clean implementation. Good choice on the keep_prob and the learning rate.

Couple of questions for you to think about -

- Is it better to call `sess.run(tf.global_variables_initializer())` in this function (like you have), or to call it in `run()` before calling `train_nn()`? This is more of a TF or general python related question, so not too much to worry about, but sort of helps understand how TF works under the hood at times. So, try to think about this if you'd like!
- How do you think your loss or your results will change if you decreased your learning rate by a factor over the epochs? Like after every 5 epochs you reduced it by a factor of 1.5, for example? Would it help given you are working with the Adam optimizer?
- Do you think you really need to have a dropout/keep_prob here considering you are also trying to apply L2 Regularization? Think about what both of them aim to do for your model, and the scope of this project and see if you think you need both or just one, and if the latter then which one.

Neural Network Training

On average, the model decreases loss over time.

Nice work!

The number of epoch and batch size are set to a reasonable number.

Nice job with the hyperparameters here!

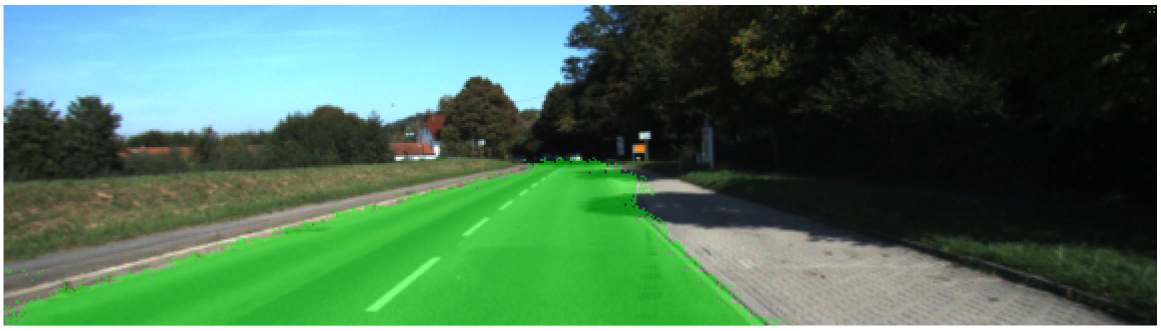
As you might already know, usually, FCNs require smaller batch sizes (and even epochs) because of memory limitations. This is also dependent on the kind of system you have as well (like GPU memory). But you have selected good values nevertheless.

The project labels most pixels of roads close to the best solution. The model doesn't have to predict correctly all the images, just most of them.

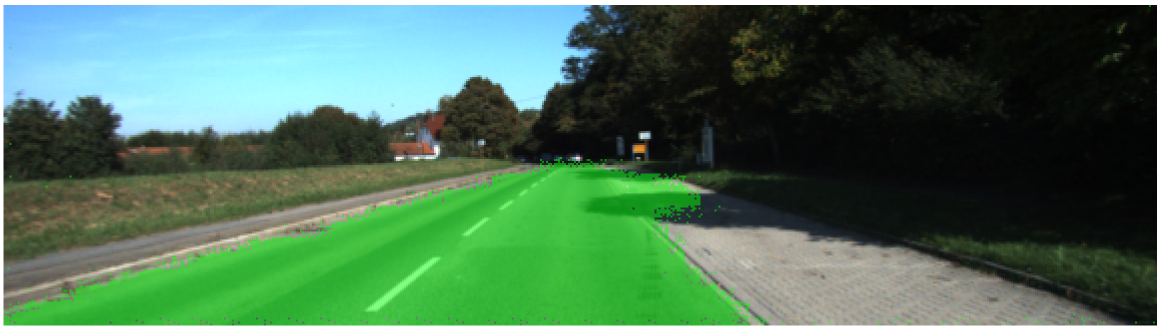
A solution that is close to best would label at least 80% of the road and label no more than 20% of non-road pixels as road.

Excellent work! You could try some more experimentation around hyperparameters and your Kernel Initialiazation/Regularization, but your results look quite good to me!

Your result -



Minimum Expected result -



Questions for you to think about -

1. If you check out the data, you will notice that there are 3 possible classes/labels. How can you modify the code to incorporate 3 classes instead of 2?
2. In your results, you will notice that the model doesn't perform that well when there are shadows on the road. What kind of data augmentation or preprocessing techniques can you use here to improve

that? Or would that require a completely different model than the VGG one we are using? Can you implement preprocessing with different colorspaces? Think about it :)

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review