Object Classification

A part of the Deep Learning Nanodegree Foundation Program

* [**PROJECT REVIEW**](https://review.udacity.com/)
* [**CODE REVIEW**](https://review.udacity.com/)
* [**NOTES**](https://review.udacity.com/)

**SHARE YOUR ACCOMPLISHMENT**

Requires Changes

**1 SPECIFICATION REQUIRES CHANGES**

Great work on this project. You have made big progress, and you are on the finish line. Please address issues mentioned in my comments.  
Please do not be disappointed. You can resubmit as many time as needed without any penalties. Take it as a feedback and opportunity to learn more.  
Good luck with the nanodegree!

One more tip: You might want to try using batch normalization techniques, which normalize the output of every conv layer, very much for the same reason that you normalized the input image before feeding it to the net. You can add tensor = tf.contrib.layers.batch\_norm(tensor, center=True, scale=True) before relu in conv2d\_maxpool.  
Read more on Batch Normalisation layers and Residual Nets.  
<http://cs231n.github.io/neural-networks-2/#batchnorm>  
<https://arxiv.org/abs/1502.03167>

Re your question: it's old fashioned trial and error approach :)

**Required Files and Tests**

**The project submission contains the project notebook, called “dlnd\_image\_classification.ipynb”.**

**All the unit tests in project have passed.**

**Preprocessing**

**The normalize function normalizes image data in the range of 0 to 1, inclusive.**

Your normalize function works fine.

**The one\_hot\_encode function encodes labels to one-hot encodings.**

Your implementation works, but it's not very efficient. If you wanted to implement it from scratch, you could have written it as:

def one\_hot\_encode(x):

result = np.zeros((len(x), 10))

result[np.arange(len(x)), x] = 1

return result

You could have used LabelBinarizer or OneHotEncoder from scikit-learn in order to save time developing and running.

**Neural Network Layers**

**The neural net inputs functions have all returned the correct TF Placeholder.**

You have correctly implemented all neural network input functions!

**The conv2d\_maxpool function applies convolution and max pooling to a layer.**

**The convolutional layer should use a nonlinear activation.**

**This function shouldn’t use any of the tensorflow functions in the tf.contrib or tf.layers namespace.**

Your implementation of conv2d\_maxpool is great!

**The flatten function flattens a tensor without affecting the batch size.**

Good

**The fully\_conn function creates a fully connected layer with a nonlinear activation.**

Very good. Impressive that you decided not to use Layers and Layers (contrib) packages.  
You can try using relu instead of tanh, since it's non-linear and can reproduce more complex relationships.

**The output function creates an output layer with a linear activation.**

Perfect

**Neural Network Architecture**

**The conv\_net function creates a convolutional model and returns the logits. Dropout should be applied to alt least one layer.**

Great! Your implementation of conv\_net meets all the requirements.

**Neural Network Training**

**The train\_neural\_network function optimizes the neural network.**

You almost had it right. You have hardcoded keep probability.

**The print\_stats function prints loss and validation accuracy.**

Yes, you have correctly set keep\_prob = 1!

**The hyperparameters have been set to reasonable numbers.**

Keep probability of 0.6 is too high. You should consider decreasing keep\_probability to around 0.5. This way your neural network will be more robust.  
Batch size of 256 is adequate. The higher you set it, the easier it will be for NN to train. But you have to consider your physical memory limitations, of course :)

**The neural network validation and test accuracy are similar. Their accuracies are greater than 50%.**

Redo this part after addressing all other comments.