UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Perfect submission!

You have implemented the convolutional network correctly and trained it such that it easily meets the required 50% accuracy. Be sure to check out my comments below they might help you to further improve your project.

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.

The `one_hot_encode` function encodes labels to one-hot encodings.

Neural Network Layers

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

Good work! You have created the placeholders with the right shapes and data types and given them proper names.

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.

Great job! You correctly implemented a convolution, activation and max pooling layer in the right order.

Also well done on the weight initialization, as you might have noticed this can speed up the learning of your network considerably. Check out http://cs231n.github.io/neural-networks-2/#init for more tips on weight initialization.

Note that you can interchange the max pooling and nonlinear layer (as long as the nonlinear activation function is an increasing function - you can try to prove this mathematically). If the max pooling is applied before the nonlinear activation, the number of nonlinear activation function calls reduces, so this should give you some small speed up!

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

The `fully_conn` function creates a fully connected layer with a nonlinear activation.

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

Neural Network Architecture

The `conv_net` function creates a convolutional model and returns the logits. Dropout should be applied to alt least one layer.

Good work on the network architecture!

Convolutional networks with multiple convolutional layers with increasing depth (e.g. 32, 64, 128), a small convolutional filter (3x3) and stride (1x1) , small max pooling size (2x2) and stride (2x2) usually work best. Be sure to check out http://cs231n.github.io/convolutional-networks/#architectures for a detailed discussion of and tips for building convolutional network architectures.

Neural Network Training

The `train_neural_network` function optimizes the neural network.

The `print_stats` function prints loss and validation accuracy.

The hyperparameters have been set to reasonable numbers.

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

The validation and test accuracy are even greater than 70%, well done!

Some suggestions to get an even better accuracy:

- See if your network performs better if you replace the nonlinear activation by `tf.nn.elu`. In my experiments a saw an increase of a few percentage points! Check out this paper for more details https://arxiv.org/abs/1511.07289.
- Add more convolutional layers to your network. Here you will need to be careful about initializing the weights as the network will have problems to train if not done properly. See http://cs231n.github.io/neural-networks-2/#init for more info on weight initialization.
- Implement batch normalization, this will be handled later in a later course - in deep convolutional GANs to be precise. See https://github.com/udacity/deep-learning/blob/master/batch-norm/Batch_Normalization_Lesson.ipynb for the notebook on this topic.

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ