UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

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.

That works! You can also just divide by 255 since we know the max value of a pixel is 255 and the min is 0. `x/255`

For more real-life applications, we would typically use sklearn's StandardScaler

George Hinton, a pioneer in neural networks, has a nice Coursera course on neural nets. This lecture explains why normalization helps.

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

Nice job! I always find the one-line solutions neat at codewars.com.
One line solution for this problem:
`np.eye(10)[x]`

or

`np.identity(10, dtype=int)[x]`

Neural Network Layers

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

For the first function you could do:
`tf.placeholder(tf.float32, shape=[None, *image_shape], name='x')`
The * is unpacking a list.

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.

Nice job! There are a few xavier initializers you might want to take a look at. They initialize the weights in a more ideal way that adapts to the network shape.

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.

Awesome work, that's a simple yet effective model! Here's more info on the architecture of conv nets. Usually we don't apply dropout to convolutional layers because they already have a lot of regularization built-in.

Every great net out there I see has the convolutional kernels increasing with network depth. For example, the VGG net designs. Here's a keras model, and here's a tensorflow model (the tf model code is quite confusing in my opinion). They start with a number (64) for number of kernels, and double it each time they descend a layer. They are using 3x3 kernels (I've always seen it recommended to use 3x3 or 5x5 kernels, although I've seen 2x2 kernels work in this project quite well). They don't use dropout until the dense layers, and then it's at 0.5. Just some food for thought on conv net design.

You can also use transfer learning, i.e. check out the 'extract features' section of this repo. This will pass the image through some pre-trained world-champion-level networks. You can then use the features vector and pass it through some fully-connected layers to get better results. The only issue here is that cifar-10 images are really small, and they would have to be up-scaled to 224x224 for the feature extraction to work.

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%.

Great job!

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ