UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

Great first submission on this project! 👍

You only need to make a few small changes to complete the project. Check out my comments below for the required change and some tips to further improve your project and the prediction accuracy.

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.

Well done! Alternatively you could use sklearn's LabelBinarizer:

```
    import sklearn.preprocessing
    label_binarizer = sklearn.preprocessing.LabelBinarizer()
    label_binarizer.fit(range(10))
    one_hot = label_binarizer.transform(x)
    return one_hot
```

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.

Python tip: consider using the more compact `[None + *image_shape]` here and in other places, see https://docs.python.org/3/tutorial/controlflow.html#unpacking-argument-lists for more information on unpacking lists.

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 a slight 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.

The default activation function of `tf.contrib.layers.fully_connected` is `tf.nn.relu`. Set the parameter `activation_fn` in `tf.contrib.layers.fully_connected` to `None` to have a linear activation in the output layer.

We want a linear activation because the output layer should return logits - these can be negative too.

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) and only a few or no fully connected layers 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 training loss calculation is correct, but you are calculating the accuracy also on the training data. You have to calculate the accuracy with the **validation** data, so you will need to use the global variables `valid_features` and `valid_labels` as explained in the instruction in the notebook.

The hyperparameters have been set to reasonable numbers.

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

The test accuracy is greater than 50% but the validation accuracy cannot be evaluated because it's not correctly calculated. Please check this item again once you correct the validation accuracy calculation.

Additionally, some tips to get a higher 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.

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Student FAQ