UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

I'm pleased to see that you've used tensorflow for applying in deep learning models very nicely. You should soon solve some interesting deep learning applications using tensorflow for example by participating on kaggle. 😀

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.

You could also have normalized the image by dividing each pixels by the maximum value of the image pixels. 0 is the minimum value of the image pixels and 255 is the maximum value of the image pixels.

```
def normalize(x):
    return x/255
```

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

Alternatively you could used numpy's eye operation as follows;

```
def one_hot_encode(x):
    return np.eye(10)[x]
```

Numpy library is based on C language and so it is quite efficient implementation.

Neural Network Layers

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

Input data placeholder should be with datatype of float32 because image pixels have been normalized and are in decimals. However class labels should be in int32.

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.

It is a correct implement. However the following order will be more efficient if you're using ReLU activation function;
1)convolution
2)max pooling
3)activation-ReLU

Since relu(max_pool(x)) == max_pool(relu(x)) we can save most of the relu-operations by max-pooling first.

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

The purpose of flatten function is to connect the fully connected layer with the convolutional/maxpool layer by changing the dimensions of x_tensor from a 4-D tensor to a 2-D tensor.

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

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

That's correct! There's no need for any non-linear activation function here. Logits will be created from the output layer. These logits will be used by the softmax which will generate the predicted values.

Neural Network Architecture

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

That's correct! It is better to apply dropout only on fully connected layer instead also on convolutional layer. Because convolutional layers are already regularised.
https://www.reddit.com/r/MachineLearning/comments/42nnpe/why_do_i_never_see_dropout_applied_in/

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 batch size can vary, depending on the performance of the computer. Very low batch-size can result in the optimization of the loss value to be noisy.

Check out this lesson video on mini-batch training and gradient descent

https://www.youtube.com/watch?v=hMLUgM6kTp8&index=20&list=PLAwxTw4SYaPn_OWPFT9ulXLuQrImzHfOV

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

Your project meets the accuracy requirements

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ