UDACITY

PROJECT

Object Classification

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

A good project demonstrating solid understanding of convolutional neural networks and application of TensorFlow library functionality.

Congratulations on passing the second 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.

Well done, nice coding.

Note: This functionality is also provided by LabelBinarizer in the sklearn library.

```
from sklearn.preprocessing import LabelBinarizer
labelBinarizer = LabelBinarizer()
labelBinarizer.fit(range(10))

def one_hot_encode(x):
return labelBinarizer.transform(x)
```

Neural Network Layers

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

Well done. All placeholders are correctly instantiated.

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.

Excellent work in building up the convolution with max pooling layer from the individual tf components.

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

Well done in implementing `flatten(...)` using base tf functionality

Note: This functionality is also provided by the `tf.contrib` method `tf.contrib.layers.flatten(x_tensor, num_outputs)`.

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

Well done in implementing `fully_conn(...)` using base tf functionality.

Note: This functionality is also provided by the `tf.contrib` method `tf.contrib.layers.fully_connected(x_tensor, num_outputs, activation_fn=tf.nn.relu)`.

Note: Bias is usually initialised to zeros rather than a normal distribution.

```
weights = tf.Variable(tf.truncated_normal(w, mean=0.0, stddev = 0.1))
bias = tf.Variable(tf.truncated_normal([num_outputs]))
```

Note: I am not sure why `elu` used rather than `relu`. `relu` is computationally cheaper.

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.

Well done. Architecture meets the specifications.

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 network testing accuracy of 66% is well above the specification minimum testing accuracy of 50%. Well done.

DOWNLOAD PROJECT

Rate this review

Student FAQ