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

Kudos ! I think you've done a perfect job of implementing a convolutional neural net fully. It's very clear that you have a good understanding of the basics. Keep improving and keep learning.

If you are keen on learning a bit more into what Computer Vision Scientists use regularly in their nets. Try reading up a bit more on

- Batch Normalisation layers
- Deconvolutional layers
- Dilated Convolutional layers
  The details of all these layers are there in the TFLearn modules.

Keep up the good work !

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.

Good job normalising the image with global maxima of intensities i.e. 255 instead of normalising it with

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

SUGGESTION

The model that you have implemented would work well only if the number of labels are limited to 10 as you are likely to face memory and time issues once labels scale to 100s or 1000s . You could use something like LabelBinariser to work for dynamic ranges.

Neural Network Layers

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

Good job in implementing all the placeholders so perfectly !

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.

Very good job in implementing the conv, maxpool layers with the appropriate shape filters and adding the bias. This was one of the tougher challenges in the entire submission. Good you could solve it so easily :)

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

Appreciate that you used basic tensor operations for the reshape layer instead of using a direct off-the-shelf implementation, Very impressive indeed !

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

Again, appreciate that you used basic tensor operations for the fully connected layer instead of using a direct off-the-shelf implementation, Very impressive, again !

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

Again , very impressive and more so because of careful implementation of 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.

Looks like a perfect conv net to start off with !

Pro Tip : Given you've already implemented the net so well, there's some extra resources for you to read on. This answer is regarding how to choose the best architecture of convolutional layers.
Couple of rules of thumb to help you going ahead

- Try and always use batch_normalisation after conv layers before maxpool so that gradients don't overflow while training
- If you're downsampling image by factor of x (say using maxpool) , then always increase the number of filters in the subsequent convolution kernel by x i.e. if you're having `conv_num_outputs` as 16 and you maxpool by 2, then in the next conv layer increase `conv_num_outputs` to 32.

Neural Network Training

The `train_neural_network` function optimizes the neural network.

The `print_stats` function prints loss and validation accuracy.

The `print_stats` function requires to print training loss and **validation_accuracy** while you've printed training loss and **training accuracy** , this is because of the wrong feed_dict you've used for calculating accuracy. Ideally it should've been

```
valid_accuracy = session.run(accuracy, feed_dict={x: valid_features, y: valid_labels, keep_prob:1.0})
```

The hyperparameters have been set to reasonable numbers.

The hyper parameter optimisation can only be determined if it's fine using the validation accuracy to check if the net is overfitting. As there was no validation accuracy printed out , it's tough to ascertain if the hyper-parameters were tuned correctly or not.

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

Test accuracy is observed above 50% . As the validation accuracy hasn't been printed it's not possible to compare the validation and test accuracy.

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)

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Rate this review

Student FAQ