Paul Voigtlaender <voigtlaender@vision.rwth-aachen.de>
Francis Engelmann <engelmann@vision.rwth-aachen.de>

Exercise 5: Convolutional Neural Networks

due **before** 2018-01-18

Important information regarding the exercises:

- The exercises are not mandatory. Still, we strongly encourage you to solve them! All submissions will be corrected. If you submit your solution, please read on:
- Use the L^{2}P system to submit your solution. You will also find your corrections there.
- Due to the large number of participants, we require you to submit your solution to L^{2}P **in groups of 3 to 4 students**. You can use the **Discussion Forum** on L^{2}P to organize groups.
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please do **not** include the data files in your submission!
- Please upload your pen & paper problems as PDF. Alternatively, you can also take pictures (.png or .jpeg) of your hand written solutions. Please make sure your handwriting is legible, the pictures are not blurred and taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.

Question 1: Classification with CNN ..... $(\Sigma = 20)$

In this exercise you will implement a CNN for classifying image patches. A basic code framework is provided in the L2P learning room. In the following tasks, you will complete the basic steps necessary for building a simple CNN that can classify images with acceptable accuracy. A dataset (500MB) is provided at `https://omnomnom.vision.rwth-aachen.de/data/cityscapesExtracted.zip` and contains examples for training, validation and testing. It consists of RGB images of three categories: pedestrians, cars and cyclists which were down-scaled to at most 100×100 pixels for memory reasons.

(a) Have a brief look on how to start TensorBoard so that you can visualize the training progress. Basic learning error outputs have already been added. **(0 pts)**
Hint: You may also look at the TensorFlow debugger. The code has a debug flag which runs the model in a debug session.

(b) The dataset contains extraction artifacts e.g. small parts of cars or far away pedestrians. In order to remove images which do not contain any reasonable content, you should complete `input_cs` and filter out these artifacts by thresholding the size of the content to at least 900 pixels. **(1 pt)**
Hint: Take a look at the `Dataset.filter` function of TensorFlow and use it with the given `input_cs._filter_size` function.
Hint: You may pass a python function using TensorFlow's `tf.py_func`.

(c) In general, further preprocessing such as normalization will be necessary before using the CNN. In our case, a normalization of the channel values from $[0 \dots 255]$ to $[0 \dots 1]$ will be sufficient and should be implemented in `input_cs`. **(1 pt)**
Hint: Take a look at `Dataset.map` and rescale the images.

(d) Finally, we also need to resize the images before passing them to the CNN. Resize the image 64 by 64 pixels using TensorFlow's `tf.resize_images` function. **(1 pt)**

(e) The main building blocks of CNNs are convolution and pooling layers, respectively. For our purpose a small CNN with 3 layers of each type will be sufficient where each convolutional layer is succeeded by a ReLU activation and a max-pooling layer. The convolutional layers should have a receptive field of 5 by 5 pixels. The layers may comprise 24, 32, and 50 filters which are applied with a stride size of 1. You may pool over blocks of 3 by 3 with a stride size of 2 for all pooling layers. Implement the proposed architecture in `model.build_model`. **(5 pts)**
Hint: For implementing the convolution in low level TensorFlow you will need to use TensorFlow's `tf.get_variable` for adding weight variables for the convolutions and biases to the computational graph.
Hint: Using these variables, you can apply the convolution with `tf.nn.conv2d` and add the biases with `tf.nn.bias_add` afterwards.
Hint: Finally, you will need `tf.nn.relu` and `tf.nn.max_pool` for non-linearity application and pooling, respectively.

(f) In order to map the representation after the last convolutional layer to a class label, you should implement three fully connected layers of size 100, 50 and number of categories in `model.build_model`. **(2 pts)**
Hint: You should not implement a softmax on top of the last fully connected layer since we will use a stable internal implementation later on.

(g) For training purposes, we need to specify a proper loss on top of a softmax which will be appropriate to predict the final class. In our case, a suitable loss function is the cross entropy loss which should be implemented in the `model.loss` function. **(1 pt)**
Hint: You may use `tf.nn.sparse_softmax_cross_entropy_with_logits`, a stable implementation of softmax and cross entropy error.
Hint: The function should return the per-sample loss for book-keeping reasons and the batch loss, respectively.

(h) Implement the training operation in `model.get_train_op_for_loss`. **(1 pt)**
Hint: A simple `tf.train.GradientDescentOptimizer` will be sufficient.

(i) Complete the basic training loop in `train.train`. **(4 pts)**

(j) One possibility to regularize the model is early stopping based on the validation set error. After training for a certain number of epochs, the error on a validation set is measured. One may save the best model or stop training if the error increases too much. Although not crucial for this simple task, you should implement a validation step which saves the best model. Complete the code in `train.train` which computes the mean cross entropy error and the accuracy on the validation set. **(3 pts)**

(k) Finally, we want to use the model to predict the images in the test set. Implement `main_cityscape.run_test` which should print the test accuracy. Furthermore, compute the accuracy for each call separately and print the mean of these accuracies. What do you observe? Finally, you should compute the confusion matrix i.e. a matrix where entry (*prediction*, *label*) corresponds to how often the model predicted *prediction* for an instance of label *label*. What do you observe and why? Is there a problem with this dataset? **(1 pt)**
Hint: You may use `tf.argmax` for predicting the output if you want to do the prediction in TensorFlow.