```
1 # Copyright 2015 Google Inc. All Rights Reserved.
2 #
3 # Licensed under the Apache License, Version 2.0 (the "License");
4 # you may not use this file except in compliance with the License.
5 # You may obtain a copy of the License at
6 #
7 #     http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implie
12 # See the License for the specific language governing permissions and
13 # limitations under the License.c
14 # ==============================================================================
15
16 """Evaluation for CIFAR-10
17 Accuracy:
18 cifar10_train.py achieves 83.0% accuracy after 100K steps (256 epochs
19 of data) as judged by cifar10_eval.py.
20 Speed:
21 On a single Tesla K40, cifar10_train.py processes a single batch of 128
22 in 0.25-0.35 sec (i.e. 350 - 600 images /sec). The model reaches ~86%
23 accuracy after 100K steps in 8 hours of training time.
24 Usage:
25 Please see the tutorial and website for how to download the CIFAR-10
26 data set, compile the program and train the model.
27 http://tensorflow.org/tutorials/deep_cnn/
28 """
29 from __future__ import absolute_import
30 from __future__ import division
31 from __future__ import print_function
32
33 from datetime import datetime
34 import math
35 import time
36
37 import numpy as np
38 import tensorflow as tf
39 import os
40 import StringIO
```

```
41 import cv
42 import cv2
43 import urllib
44
45
46 from PIL import Image
47
48 import matplotlib
49
50 import glob
51
52 import cifar10
53
54 cur_dir = os.getcwd()
55
56 FLAGS = tf.app.flags.FLAGS
57
58 tf.app.flags.DEFINE_string('eval_dir', '/tmp/cifar10_eval',
59                            """Directory where to write event logs.""")
60 tf.app.flags.DEFINE_string('eval_data', 'test',
61                            """Either 'test' or 'train_eval'.""")
62 tf.app.flags.DEFINE_string('checkpoint_dir', '/tmp/cifar10_train',
63                            """Directory where to read model checkpoints."
64 tf.app.flags.DEFINE_integer('eval_interval_secs', 60 * 5,
65                             """How often to run the eval.""")
66 tf.app.flags.DEFINE_integer('num_examples', 10000,
67                             """Number of examples to run.""")
68 tf.app.flags.DEFINE_boolean('run_once', False,
69                          """Whether to run eval only once.""")
70
71
72 def eval_once(saver, summary_writer, top_k_op, summary_op,images,labels
73   """Run Eval once.
74   Args:
75     saver: Saver.
76     summary_writer: Summary writer.
77     top_k_op: Top K op.
78     summary_op: Summary op.
79   """
80   with tf.Session() as sess:
```

```
81    ckpt = tf.train.get_checkpoint_state(FLAGS.checkpoint_dir)
82    if ckpt and ckpt.model_checkpoint_path:
83      # Restores from checkpoint
84      saver.restore(sess, ckpt.model_checkpoint_path)
85      # Assuming model_checkpoint_path looks something like:
86      #   /my-favorite-path/cifar10_train/model.ckpt-0,
87      # extract global_step from it.
88      global_step = ckpt.model_checkpoint_path.split('/')[-1].split('-')
89    else:
90      print('No checkpoint file found')
91      return
92
93    # Start the queue runners.
94    coord = tf.train.Coordinator()
95    try:
96      threads = []
97      for qr in tf.get_collection(tf.GraphKeys.QUEUE_RUNNERS):
98        threads.extend(qr.create_threads(sess, coord=coord, daemon=True
99                                         start=True))
100
101     num_iter = int(math.ceil(FLAGS.num_examples / FLAGS.batch_size))
102     true_count = 0  # Counts the number of correct predictions.
103     total_sample_count = num_iter * FLAGS.batch_size
104     step = 0
105
106
107
108
109
110     while step < num_iter and not coord.should_stop():
111       predictions = sess.run([top_k_op])
112       true_count += np.sum(predictions)
113       step += 1
114     # Compute precision @ 1.
115       precision = true_count / total_sample_count
116       print('%s: precision @ 1 = %.3f' % (datetime.now(), precision))
117       e = tf.nn.softmax(logits)
118       log = sess.run(e)
119       #print(log)
120       predict = np.zeros([FLAGS.batch_size])
```

```
121         max_logi = np.zeros([FLAGS.batch_size])
122 
123         for i in xrange(FLAGS.batch_size):
124           predict[i] = np.argmax(log[i, :])
125           max_logi[i] = log[i, :].max()
126         lab = sess.run(labels)
127         top = sess.run([top_k_op])
128         predictions = sess.run([top_k_op])
129         true_count = 0
130         true_count += np.sum(predictions)
131         # chk = sess.run(images)
132         #print(top)c
133         for i in xrange(FLAGS.batch_size):
134           #    tf.cast(images, tf.uint8)
135           img = sess.run(images)
136           save_img = img[i, :]
137 
138           save_img = ((save_img - save_img.min()) / (save_img.max() - sa
139 
140           #      save_img2 = Image.fromarray(save_img, "RGB")
141 
142           path = cur_dir + "/result/"
143 
144           if not os.path.exists(path):
145             os.mkdir(path, 0755)
146           if predictions[0][i]==True:
147             path = path + "Correct/"
148           else:
149             path = path + "Incorect/"
150 
151           if not os.path.exists(path):
152             os.mkdir(path, 0755)
153           class_fold = path + str(predict[i]) + "/"
154           # class_fold = path + str(max_logi[i]) + "/
155           if not os.path.exists(path + str(predict[i]) + "/"):
156             os.mkdir(class_fold, 0755)
157 
158           cv2.imwrite(os.path.join(class_fold, str(i) + ".jpeg"), save_i
159 
160 
```

```
161
162       summary = tf.Summary()
163       summary.ParseFromString(sess.run(summary_op))
164       summary.value.add(tag='Precision @ 1', simple_value=precision)
165       summary_writer.add_summary(summary, global_step)
166     except Exception as e:  # pylint: disable=broad-except
167       coord.request_stop(e)
168
169     coord.request_stop()
170     coord.join(threads, stop_grace_period_secs=10)
171
172
173 def evaluate():
174   """Eval CIFAR-10 for a number of steps."""
175   with tf.Graph().as_default() as g:
176     # Get images and labels for CIFAR-10.
177     eval_data = FLAGS.eval_data == 'test'
178     images, labels = cifar10.inputs(eval_data=eval_data)
179
180     # Build a Graph that computes the logits predictions from the
181     # inference model.
182     logits = cifar10.inference(images)
183     true_count = 0
184     # Calculate predictions.
185     top_k_op = tf.nn.in_top_k(logits, labels, 1)
186
187
188
189
190     # Restore the moving average version of the learned variables for ev
191     variable_averages = tf.train.ExponentialMovingAverage(
192         cifar10.MOVING_AVERAGE_DECAY)
193     variables_to_restore = variable_averages.variables_to_restore()
194     saver = tf.train.Saver(variables_to_restore)
195
196     # Build the summary operation based on the TF collection of Summarie
197     summary_op = tf.merge_all_summaries()
198
199     summary_writer = tf.train.SummaryWriter(FLAGS.eval_dir, g)
200
```

```
201     #while True:
202     eval_once(saver, summary_writer, top_k_op, summary_op,images,labels,
203     #  if False:
204     #    break
205     #  time.sleep(FLAGS.eval_interval_secs)
206
207
208 def main(argv=None):  # pylint: disable=unused-argument
209   cifar10.maybe_download_and_extract()
210   if tf.gfile.Exists(FLAGS.eval_dir):
211     tf.gfile.DeleteRecursively(FLAGS.eval_dir)
212   tf.gfile.MakeDirs(FLAGS.eval_dir)
213   evaluate()
214
215
216 if __name__ == '__main__':
217   tf.app.run()
218
```