HW10: Convolutional Neural Network

Yu Che Wang / yuchecw2

December 13, 2018

Abstract

Use convolutional neural network to build an image classifier and utilize TensorBoard to log and visualize accuracy.

1 Classify MNIST with tutorial

In this section, I refer to the MNIST tutorial `https://www.tensorflow.org/tutorials/estimators/cnn` to build a handwritten digit classifier.

1.1 Train and validation accuracy plot

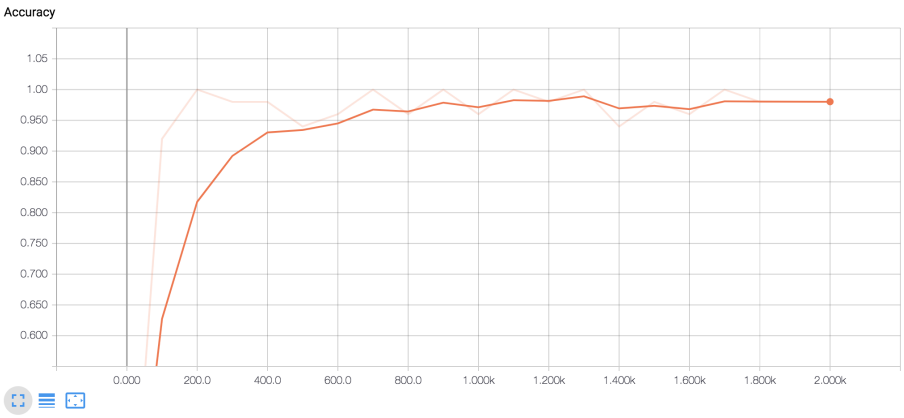

Figure 1: Train accuracy plot for MNIST.

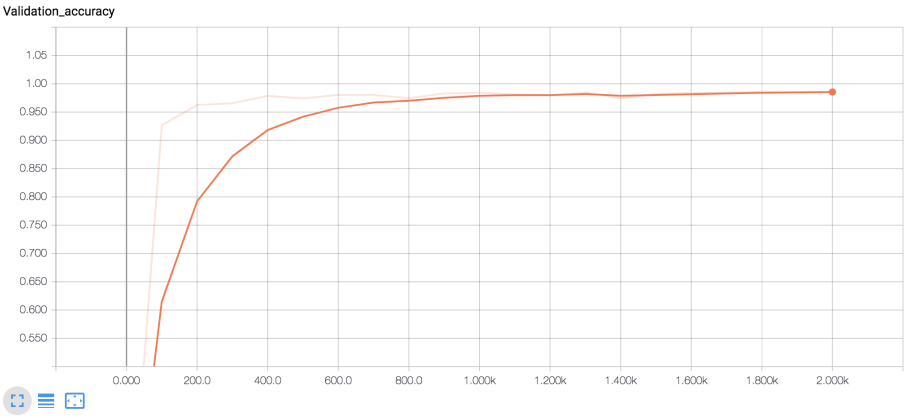

Figure 2: Validation accuracy plot for MNIST.

1.2 Accuracies from the best model

I choose the one after 800 epochs for the best model since the train accuracy becomes stable and further training does not boost the accuracy a lot.

Accuracy	Value
Train	0.975
Validation	0.968
Test	0.972

Table 1: Accuracies for MNIST.

2 Classify MNIST with modifications

In this section, I build a classifier based on Section 1 with slight modifications. I modified the output channel of the first convolutional layer and the input channel of the second convolutional layer from 32 to 8. Furthermore, I applied batch normalization after each ReLU activation.

2.1 Train and validation accuracy plot

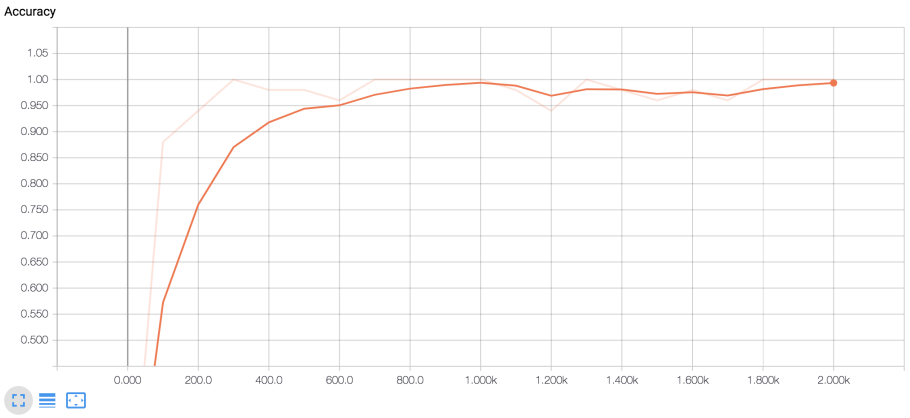

Figure 3: Train accuracy plot for MNIST.

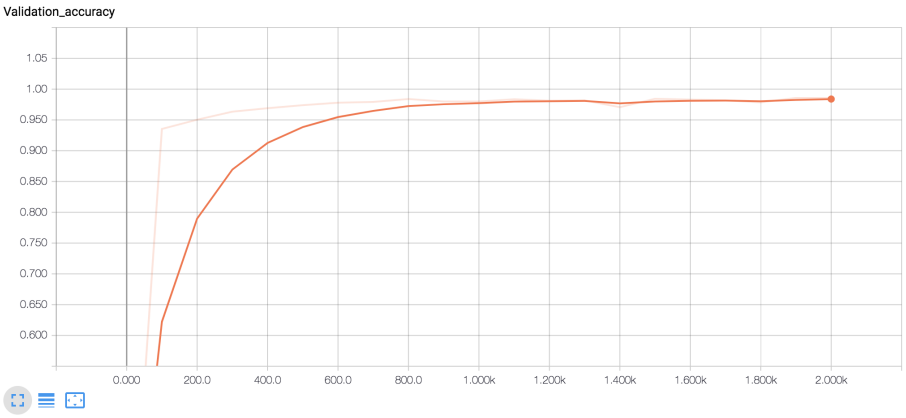

Figure 4: Validation accuracy plot for MNIST.

2.2 Accuracies from the best model

I choose the one after 800 epochs for the best model since the train accuracy becomes stable and further training does not boost the accuracy a lot.

Accuracy	Value
Train	0.982
Validation	0.972
Test	0.975

Table 2: Accuracies for modified MNIST model.

3 Classify CIFAR-10 with tutorial

In this section, I refer to the CIFAR-10 tutorial `https://www.tensorflow.org/tutorials/images/deep_cnn` and `https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10` to build a image classifier.

3.1 Train and validation accuracy plot

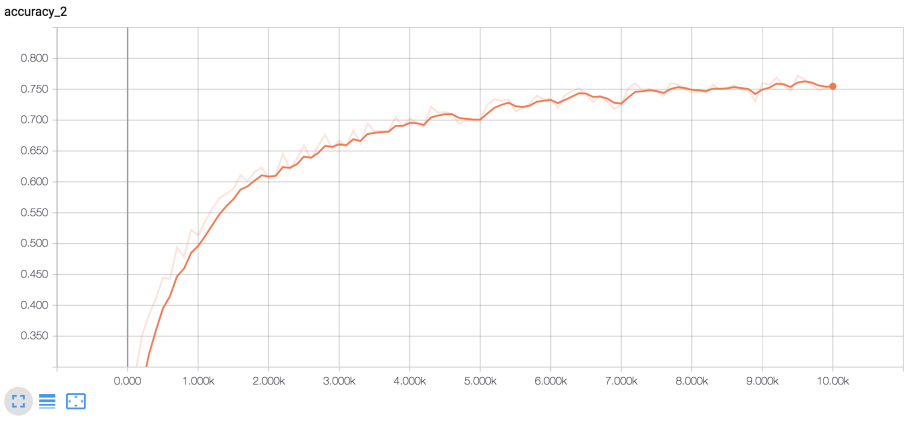

Figure 5: Train accuracy plot for CIFAR-10.

To get validation accuracy plot, run 'python cifar10_train.py' and 'python cifar10_eval.py' concurrently.

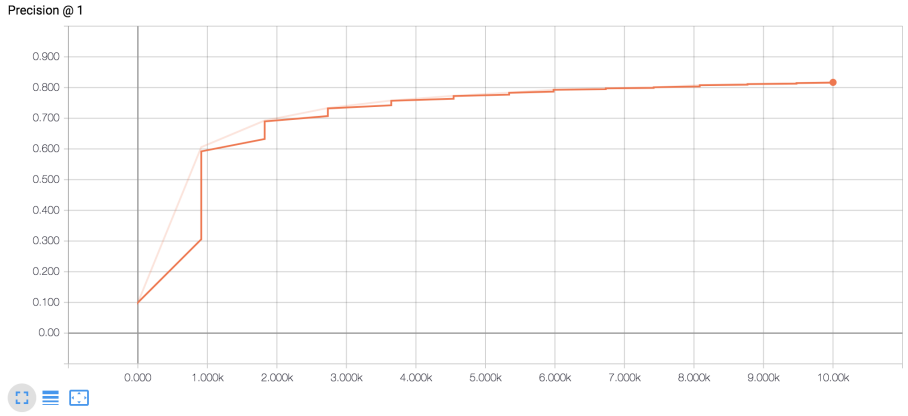

Figure 6: Validation accuracy plot for CIFAR-10.

3.2 Accuracies from the best model

Due to the accuracy and time for training trade-off, I choose the model after 8000 epochs as the best model. Although the accuracy is still increasing at epoch 8000, accuracy remains relatively stable compared to that of before epoch 5000.

Accuracy	Value
Train	0.754
Validation	0.818
Test	0.795

Table 3: Accuracies for CIFAR-10.

4 Classify CIFAR019 with modifications

In this section, I build a classifier based on Section 3 with some modifications. I added one more convolutional layer in betweeen the two original convolutional layers.

4.1 Train and validation accuracy plot

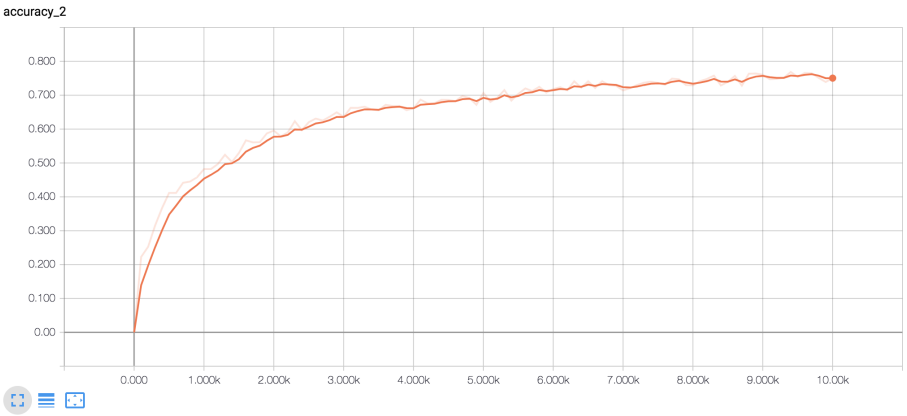

Figure 7: Train accuracy plot for CIFAR-10 with modifications.

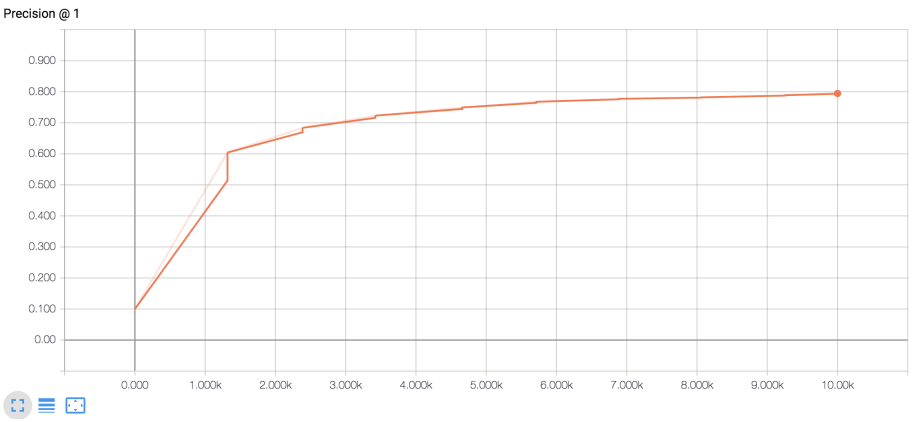

Figure 8: Train accuracy plot for CIFAR-10 with modifications.

4.2 Accuracies from the best model

Due to the accuracy and time for training trade-off, I choose the model after 7000 epochs as the best model. The accuracy curve remains relatively stable after epoch 7000. In fact, to obtain higher accuracy, one should use ResNet or VGGNet.

Accuracy	Value
Train	0.756
Validation	0.794
Test	0.787

Table 4: Accuracies for CIFAR-10.

5 Code snippets

5.1 MNIST code

```
def train():
  parser = argparse.ArgumentParser()
  parser.add_argument('--test', default=False, type=bool)
  parser.add_argument('--batch_norm', default=False, type=bool)
  args = parser.parse_args()

  tf.reset_default_graph()
  sess = tf.Session()
  # Input flattened image
  x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
  # Class label
  y = tf.placeholder(tf.float32, shape=[None, 10], name='y')

  x_image = tf.reshape(x, [-1,28,28,1])

  # Modified Model
  conv_out1 = conv_layer(x_image, 5, 1, 8, args.batch_norm, "conv1")
  conv_out2 = conv_layer(conv_out1, 5, 8, 64, args.batch_norm, "conv2")
  flattened_out = tf.reshape(conv_out2, [-1, 7*7*64])
  fc_out1 = fc_layer(flattened_out, 7*7*64, 1024, "fc1")
  relu = tf.nn.relu(fc_out1)
  dropout = tf.layers.dropout(inputs=relu, rate=0.4)
  logits = fc_layer(dropout, 1024, 10, "fc2")

  loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
  tf.summary.scalar("Loss", loss)

  optimizer = tf.train.AdamOptimizer(0.001)
  train_step = optimizer.minimize(loss)

  correct_prediction = tf.equal(tf.argmax(logits,1), tf.argmax(y,1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  train_acc = tf.summary.scalar('Accuracy', accuracy)
  validation_acc = tf.summary.scalar('Validation_accuracy', accuracy)

  merged_summary = tf.summary.merge_all()

  sess.run(tf.global_variables_initializer())
  writer = tf.summary.FileWriter('/tmp/mnist_tutorial')
  writer.add_graph(sess.graph)

  for epoch in range(2001):
    batch = mnist.train.next_batch(50)
    if epoch % 100 == 0:
    #   train_summary_str = sess.run(train_acc, feed_dict={x: batch[0], y: batch[1]})
      [summary, train_accuracy] = sess.run([train_acc, accuracy], feed_dict={x: batch[0], y: batch[1]})
      writer.add_summary(summary, epoch)
      print('Epoch %d, train accuracy: %f'%(epoch, train_accuracy))
      [validation_summary, val_accuracy] = sess.run([validation_acc, accuracy], \
                      feed_dict={x: mnist.validation.images, y:mnist.validation.labels})
      writer.add_summary(validation_summary, epoch)
      print('Epoch %d, validation accuracy: %f'%(epoch, val_accuracy))
    sess.run(train_step, feed_dict={x: batch[0], y: batch[1]})
    if args.test:
        if epoch == 801:
            break
  if args.test:
    print('Test accuracy: %g'%accuracy.eval(session=sess, \
                          feed_dict={x: mnist.test.images, y:mnist.test.labels}))
  writer.close()
  print('Run `tensorboard --logdir=%s` to see the results.' % LOGDIR)
```

Figure 9: Creating model, defining loss, and training.

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import argparse

LOGDIR = "/tmp/mnist_tutorial/"
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

def weight_variable(shape):
  '''Create a weight variable with initialization'''
  return tf.Variable(tf.truncated_normal(shape, stddev=0.1), name="W")

def bias_variable(shape):
  '''Create a bias variable with initialization'''
  return tf.Variable(tf.constant(0.1, shape=shape), name="B")

def conv2d(x, weights):
  '''Perform 2d convolution with specified weights'''
  return tf.nn.conv2d(input=x, filter=weights, strides=[1,1,1,1], padding='SAME')

def max_pool(x):
  '''Perform max pooling'''
  return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

def conv_layer(input, kernel_size, in_channels, out_channels, batch_norm=False, name='conv'):
  '''Convolutional layer with 2d convolution, relu activation, and max pooling'''
  with tf.name_scope(name):
    w = weight_variable([kernel_size, kernel_size, in_channels, out_channels])
    b = bias_variable([out_channels])
    if not batch_norm:
        activation = tf.nn.relu(conv2d(input, w) + b)
    else:
        activation = tf.nn.relu(tf.nn.batch_normalization(conv2d(input, w)+b, 0,1,0.1,10,0.0001))
    tf.summary.histogram("Weights", w)
    tf.summary.histogram("Biases", b)
    tf.summary.histogram("Activations", activation)
    pool = max_pool(activation)
    return pool

def fc_layer(input, in_channels, out_channels, name='fc'):
  '''Fully connected layer'''
  with tf.name_scope(name):
    w = weight_variable([in_channels, out_channels])
    b = bias_variable([out_channels])
    out = tf.matmul(input, w) + b
    tf.summary.histogram("Weights", w)
    tf.summary.histogram("Biases", b)
    tf.summary.histogram("Activations", out)
    return out
```

Figure 10: Other relevant codes for MNIST.

5.2 CIFAR-10 code

Note that most of the codes are the same as the original files.

```
# Modified
# conv3
with tf.variable_scope('conv3') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 32, 64],
                                       stddev=5e-2,
                                       wd=None)
  conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
  pre_activation = tf.nn.bias_add(conv, biases)
  conv3 = tf.nn.relu(pre_activation, name=scope.name)
  _activation_summary(conv3)

# pool3
pool3 = tf.nn.max_pool(conv3, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool3')
# norm3
norm3 = tf.nn.lrn(pool3, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm3')
```

Figure 11: Modified model. Add a convolutional layer to the existing model.

```
def obtain_accuracy(logits, labels):
  correct_prediction = tf.equal(tf.argmax(logits,1), tf.cast(labels, tf.int64))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32), name='accuracy')
  tf.add_to_collection('acc', accuracy)
  return tf.add_n(tf.get_collection('acc'), name='accuracy')
```

Figure 12: Add a function to cifar10_train.py to log accuracy.

```
  # Calculate accuracy
  accuracy = obtain_accuracy(logits, labels)
```

Figure 13: Execution code the train function of cifar10_train.py.

```
def loss(logits, labels):
  """Add L2Loss to all the trainable variables.

  Add summary for "Loss" and "Loss/avg".
  Args:
    logits: Logits from inference().
    labels: Labels from distorted_inputs or inputs(). 1-D tensor
            of shape [batch_size]

  Returns:
    Loss tensor of type float.
  """
  # Calculate the average cross entropy loss across the batch.
  labels = tf.cast(labels, tf.int64)
  cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
      labels=labels, logits=logits, name='cross_entropy_per_example')
  cross_entropy_mean = tf.reduce_mean(cross_entropy, name='cross_entropy')
  tf.add_to_collection('losses', cross_entropy_mean)

  # The total loss is defined as the cross entropy loss plus all of the weight
  # decay terms (L2 loss).
  return tf.add_n(tf.get_collection('losses'), name='total_loss')
```

Figure 14: Defining loss.