1. 代码每一层的维度

```
input_layer: Tensor("Reshape:0", shape=(100, 28, 28, 1), dtype=float32)
conv1: Tensor("conv1/Relu:0", shape=(100, 28, 28, 6), dtype=float32)
pool1: Tensor("max_pooling2d/MaxPool:0", shape=(100, 14, 14, 6), dtype=float32)
conv2: Tensor("conv2d/Relu:0", shape=(100, 10, 10, 16), dtype=float32)
pool2: Tensor("max_pooling2d_2/MaxPool:0", shape=(100, 5, 5, 16), dtype=float32)
dense: Tensor("dense/Relu:0", shape=(100, 120), dtype=float32)
dense2: Tensor("dense_2/Relu:0", shape=(100, 84), dtype=float32)
logits: Tensor("dense_3/BiasAdd:0", shape=(100, 10), dtype=float32)
```

2. 试图为每一batch添加准确率的输出

https://github.com/tensorflow/tensorflow/issues/15115给出了答案 正确做法

```
my_acc = tf.reduce_mean(tf.cast(tf.equal(tf.cast(labels, tf.int64), predictions['classes']),
tf.float32))
tensors_to_log = {'Accuracy': my_acc}
logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log, every_n_iter=100)

return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op, training_hooks=
[logging_hook])
```

使用 `accuracy = tf.metrics.accuracy(labels=labels, predictions=predictions['classes'], name='accuracy')` 的做法是不行的，因为

> tf.metrics.accuracy is not meant to compute the accuracy of a single batch. It returns both the accuracy and an update_op, and update_op is intended to be run every batch, which updates the accuracy.

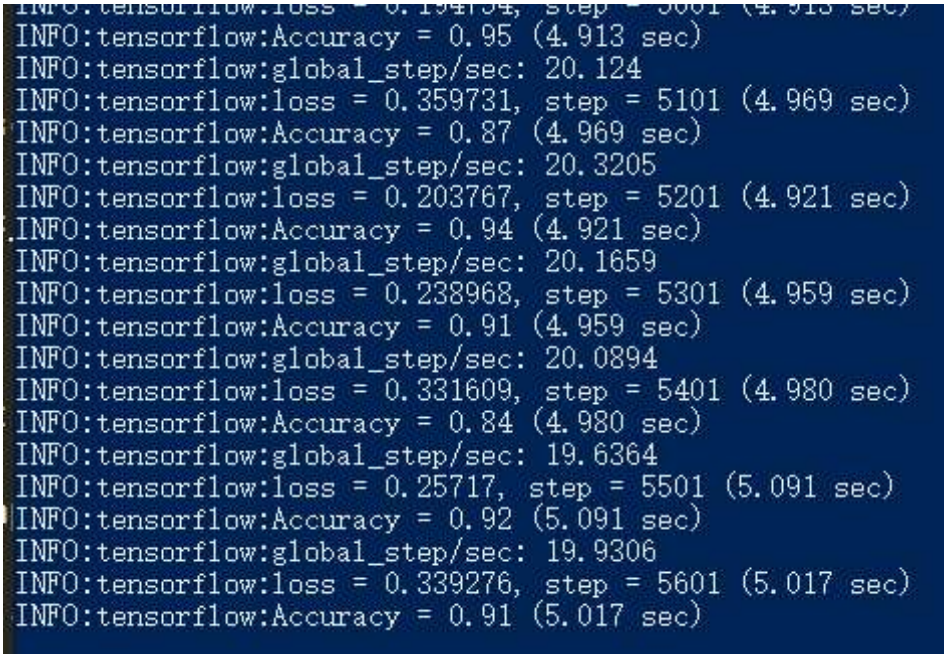
3. 代码

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Imports
import numpy as np
import tensorflow as tf

tf.logging.set_verbosity(tf.logging.INFO)

def cnn_model_fn(features, labels, mode):
  """Model function for CNN."""
  # 一批100组数据

  # 输入层
  input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])
  print("input_layer:", input_layer) # Input Layer: (100, 28, 28, 1)
  # 第一层 卷积层
```

```
conv1 = tf.layers.conv2d( # 使用6个filter
    inputs=input_layer,
    filters=6,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu,
    name="conv1")
print("conv1:", conv1) # conv1: (100, 28, 28, 6)

# 第二层 池化层
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

print("pool1:", pool1) # pool1: (100, 14, 14, 6)

# 第三层卷积层与第四层池化层
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=16,
    kernel_size=[5, 5],
    padding="valid",
    activation=tf.nn.relu)
print("conv2:", conv2) # conv2 (100, 10, 10, 16)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
print("pool2:", pool2) # pool2 (100, 5, 5, 16)

# 拉直矩阵
pool2_shape = pool2.get_shape().as_list()
flat_size = pool2_shape[1] * pool2_shape[2] * pool2_shape[3]
pool2_flat = tf.reshape(pool2, [-1, flat_size])

# 三层全连接层
dense = tf.layers.dense(inputs=pool2_flat, units=120, activation=tf.nn.relu)
dense2 = tf.layers.dense(inputs=dense, units=84, activation=tf.nn.relu)
logits = tf.layers.dense(inputs=dense2, units=10)

print("dense:", dense) # dense (100, 120)
print("dense2:", dense2) # dense2 (100, 84)
print("logits:", logits) # logits (100, 10)
predictions = {
    # Generate predictions (for PREDICT and EVAL mode)
    "classes": tf.argmax(input=logits, axis=1),
    # Add `softmax_tensor` to the graph. It is used for PREDICT and by the
    # `logging_hook`.
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
}

# Configure the Predict Op (for PREDICT mode)
if mode == tf.estimator.ModeKeys.PREDICT:
  return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

# Calculate Loss (for both TRAIN and EVAL modes)
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)

# Configure the Training Op (for TRAIN mode)
if mode == tf.estimator.ModeKeys.TRAIN:
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
  train_op = optimizer.minimize(
      loss=loss,
      global_step=tf.train.get_global_step())

  my_acc = tf.reduce_mean(tf.cast(tf.equal(tf.cast(labels, tf.int64), predictions['classes']),
```

```
tf.float32))
    tensors_to_log = {'Accuracy': my_acc}
    logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log, every_n_iter=100)

    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op, training_hooks=
[logging_hook])

  # Add evaluation metrics (for EVAL mode)
  eval_metric_ops = {
      "accuracy": tf.metrics.accuracy(
          labels=labels, predictions=predictions["classes"])}
  return tf.estimator.EstimatorSpec(
      mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)

def main(unused_param):
  # Load training and eval data
  mnist = tf.contrib.learn.datasets.load_dataset("mnist")
  train_data = mnist.train.images # Returns np.array
  train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
  eval_data = mnist.test.images # Returns np.array
  eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

  mnist_classifier = tf.estimator.Estimator(
    model_fn=cnn_model_fn)

  # Set up logging for predictions
  # tensors_to_log = {"probabilities": "softmax_tensor"}
  tensors_to_log = {}
  logging_hook = tf.train.LoggingTensorHook(
      tensors=tensors_to_log, every_n_iter=50)

  # Train the model
  train_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={"x": train_data},
      y=train_labels,
      batch_size=100,
      num_epochs=None,
      shuffle=True)
  mnist_classifier.train(
      input_fn=train_input_fn,
      steps=20000)

  # Evaluate the model and print results
  eval_input_fn = tf.estimator.inputs.numpy_input_fn(
      x={"x": eval_data},
      y=eval_labels,
      num_epochs=1,
      shuffle=False)
  eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
  print(eval_results)


if __name__ == "__main__":
  tf.app.run()
```

运行结果


INFO:tensorflow:Accuracy = 0.95 (4.913 sec)
INFO:tensorflow:global_step/sec: 20.124
INFO:tensorflow:loss = 0.359731, step = 5101 (4.969 sec)
INFO:tensorflow:Accuracy = 0.87 (4.969 sec)
INFO:tensorflow:global_step/sec: 20.3205
INFO:tensorflow:loss = 0.203767, step = 5201 (4.921 sec)
INFO:tensorflow:Accuracy = 0.94 (4.921 sec)
INFO:tensorflow:global_step/sec: 20.1659
INFO:tensorflow:loss = 0.238968, step = 5301 (4.959 sec)
INFO:tensorflow:Accuracy = 0.91 (4.959 sec)
INFO:tensorflow:global_step/sec: 20.0894
INFO:tensorflow:loss = 0.331609, step = 5401 (4.980 sec)
INFO:tensorflow:Accuracy = 0.84 (4.980 sec)
INFO:tensorflow:global_step/sec: 19.6364
INFO:tensorflow:loss = 0.25717, step = 5501 (5.091 sec)
INFO:tensorflow:Accuracy = 0.92 (5.091 sec)
INFO:tensorflow:global_step/sec: 19.9306
INFO:tensorflow:loss = 0.339276, step = 5601 (5.017 sec)
INFO:tensorflow:Accuracy = 0.91 (5.017 sec)