```
# Code is mostly from this tutorial: https://www.tensorflow.org/tutorials/layers

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

# Imports
import numpy as np
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

def cnn_model_fn(features, labels, mode):
  """Model function for CNN."""
  # Input Layer
  input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

  # Convolutional Layer #1
  conv1 = tf.layers.conv2d(
      inputs=input_layer,
      filters=32,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)

  # Pooling Layer #1
  pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

  # Convolutional Layer #2 and Pooling Layer #2
  conv2 = tf.layers.conv2d(
      inputs=pool1,
      filters=64,
      kernel_size=[5, 5],
      padding="same",
      activation=tf.nn.relu)


  pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

  # # I tried to add a third layer. It didn't work out.
  # conv3 = tf.layers.conv2d(
  #   inputs=pool2,
  #   filters=128,
  #   kernel_size=[5, 5],
  #   padding="same",
  #   activation=tf.nn.relu)
  # pool3 = tf.layers.max_pooling2d(inputs=conv3, pool_size=[2, 2], strides=2)

  # Dense Layer
  # pool2_flat = tf.reshape(pool3, [-1, 7 * 7 * 64])
```

```
  pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 64])
  dense = tf.layers.dense(inputs=pool2_flat, units=1024, activation=tf.nn.relu)
  dropout = tf.layers.dropout(
      inputs=dense, rate=0.4, training=mode == tf.estimator.ModeKeys.TRAIN)

  # Logits Layer
  logits = tf.layers.dense(inputs=dropout, units=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")
  }

  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())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

  # 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_argv):
  # Load training and eval data
  # mnist = tf.contrib.learn.datasets.load_dataset("mnist")
  mnist = input_data.read_data_sets('data/fashion', source_url='http://fashion-mnist.s3-
website.eu-central-1.amazonaws.com/')
  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)
  # Create the Estimator
  mnist_classifier = tf.estimator.Estimator(model_fn=cnn_model_fn,
model_dir="/output/mnist_convnet_model")
  # Set up logging for predictions
  tensors_to_log = {"probabilities": "softmax_tensor"}
  logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log, every_n_iter=50)
```

```
  # I added code for tensorboard to work.
  merged = tf.summary.merge_all()
  train_writer = tf.summary.FileWriter('/output/train')
  test_writer = tf.summary.FileWriter('/output/test')
  # 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, hooks=[logging_hook])
  # 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()
```