1. 입력

```
total_batch  = int(mnist.train.num_examples / batch_size)

for epoch in range(15):
  total_cost = 0

  for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    batch_xs = batch_xs.reshape(-1, 28, 28, 1)

    _, cost_val = sess.run([optimizer, cost],
                           feed_dict={X: batch_xs, Y: batch_ys, keep_prob: 0.8})
    total_cost += cost_val

  print('Epoch:', '%04d' % (epoch + 1), 'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))
```

```
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:',
      sess.run(accuracy, feed_dict={X: mnist.test.images.reshape(-1, 28, 28, 1),
                                    Y: mnist.test.labels, keep_prob: 1}))
```

MNIST 사용

2. Layer

- 첫번째 CNN 계층 및 풀링 계층

```
# 첫 번째 CNN 계층 구성 및 풀링 계층 생성
L1 = tf.layers.conv2d(X, 32, [3, 3], activation=tf.nn.relu, padding='SAME')
L1 = tf.layers.max_pooling2d(L1, [2, 2], [2, 2], padding='SAME')
L1 = tf.layers.dropout(L1, keep_prob)
```

- 두번째 CNN 계층

```
# 두 번째 CNN 계층 구성
L2 = tf.layers.conv2d(L1, 64, [3, 3])
L2 = tf.layers.max_pooling2d(L2, [2, 2], [2, 2])
L2 = tf.layers.dropout(L2, keep_prob)
```

- 완전 연결 계층

```
# 완전 연결 계층
L3 = tf.contrib.layers.flatten(L2)
L3 = tf.layers.dense(L3, 256, activation=tf.nn.relu)
L3 = tf.layers.dropout(L3, keep_prob)

# 완전 연결 계층(2)
L4 = tf.contrib.layers.flatten(L3)
L4 = tf.layers.dense(L4, 256, activation=tf.nn.relu)
L4 = tf.layers.dropout(L4, keep_prob)
```

3. 결과

- 학습 시 droptout 의 수치는 0.8 로 하였으며, 비용 함수는 cross-entropy 를 적용하였다.

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
```

```
for epoch in range(15):
  total_cost = 0

  for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    batch_xs = batch_xs.reshape(-1, 28, 28, 1)

    _, cost_val = sess.run([optimizer, cost],
                           feed_dict={X: batch_xs, Y: batch_ys, keep_prob: 0.8})
    total_cost += cost_val

  print('Epoch:', '%04d' % (epoch + 1), 'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))
```

- 최적화 함수로 AdamOptimizer 를 사용한 경우

```
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

```
Epoch: 0001 Avg. cost = 0.227
Epoch: 0002 Avg. cost = 0.051
Epoch: 0003 Avg. cost = 0.034
Epoch: 0004 Avg. cost = 0.023
Epoch: 0005 Avg. cost = 0.020
Epoch: 0006 Avg. cost = 0.016
Epoch: 0007 Avg. cost = 0.015
Epoch: 0008 Avg. cost = 0.010
Epoch: 0009 Avg. cost = 0.010
Epoch: 0010 Avg. cost = 0.012
Epoch: 0011 Avg. cost = 0.008
Epoch: 0012 Avg. cost = 0.007
Epoch: 0013 Avg. cost = 0.008
Epoch: 0014 Avg. cost = 0.007
Epoch: 0015 Avg. cost = 0.007
최적화 완료!
정확도: 0.9902
```

- 최적화 함수로 RMSPropOptimizer 를 사용한 경우

```
optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
```

```
Epoch: 0001 Avg. cost = 0.684
Epoch: 0002 Avg. cost = 0.059
Epoch: 0003 Avg. cost = 0.036
Epoch: 0004 Avg. cost = 0.025
Epoch: 0005 Avg. cost = 0.017
Epoch: 0006 Avg. cost = 0.013
Epoch: 0007 Avg. cost = 0.011
Epoch: 0008 Avg. cost = 0.009
Epoch: 0009 Avg. cost = 0.008
Epoch: 0010 Avg. cost = 0.006
Epoch: 0011 Avg. cost = 0.007
Epoch: 0012 Avg. cost = 0.005
Epoch: 0013 Avg. cost = 0.004
Epoch: 0014 Avg. cost = 0.004
Epoch: 0015 Avg. cost = 0.004
최적화 완료!
정확도: 0.992
```