从官网上下的数据集

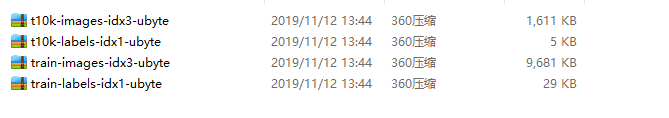
**from** matplotlib **import** pyplot **as** plt**from** tensorflow.examples.tutorials.mnist **import** input\_data​mnist **=** input\_data**.**read\_data\_sets("MNIST", one\_hot**=**True)**for** i **in** range(12):

plt**.**subplot(3, 4, i**+**1)

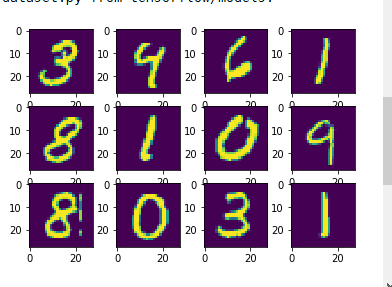
img **=** mnist**.**train**.**images[i **+** 1]

img **=** img**.**reshape(28, 28)

plt**.**imshow(img)plt**.**show()



实际上就是图片的压缩形式：



具体模型及训练：

import os

import tensorflow as tf

from matplotlib import pyplot as plt

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets("MNIST", one\_hot=True)

BATCH\_SIZE = 64

def visualization(\_mnist):

for i in range(12):

plt.subplot(3, 4, i+1)

img = \_mnist.train.images[i + 1]

img = img.reshape(28, 28)

plt.imshow(img)

plt.show()

#可视化

class LeNet(object):

def \_\_init\_\_(self):

pass

def create(self, X):

#第一层网络

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

with tf.variable\_scope("layer\_1") as scope:

w\_1 = tf.get\_variable("weights", shape=[5, 5, 1, 6])

b\_1 = tf.get\_variable("bias", shape=[6])

conv\_1 = tf.nn.conv2d(X, w\_1, strides=[1, 1, 1, 1], padding="SAME")

act\_1 = tf.sigmoid(tf.nn.bias\_add(conv\_1, b\_1))

max\_pool\_1 = tf.nn.max\_pool(act\_1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

#第二层网路

with tf.variable\_scope("layer\_2") as scope:

w\_2 = tf.get\_variable("weights", shape=[5, 5, 6, 16])

b\_2 = tf.get\_variable("bias", shape=[16])

conv\_2 = tf.nn.conv2d(max\_pool\_1, w\_2, strides=[1, 1, 1, 1], padding="SAME")

act\_2 = tf.sigmoid(tf.nn.bias\_add(conv\_2, b\_2))

max\_pool\_2 = tf.nn.max\_pool(act\_2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

#展开成向量

flatten = tf.reshape(max\_pool\_2, shape=[-1, 7 \* 7 \* 16])

#全连接1

with tf.variable\_scope("fc\_1") as scope:

w\_fc\_1 = tf.get\_variable("weight", shape=[7 \* 7 \* 16, 120])

b\_fc\_1 = tf.get\_variable("bias", shape=[120], trainable=True)

fc\_1 = tf.nn.xw\_plus\_b(flatten, w\_fc\_1, b\_fc\_1)

act\_fc\_1 = tf.nn.sigmoid(fc\_1)

#全连接2

with tf.variable\_scope("fc\_2") as scope:

w\_fc\_2 = tf.get\_variable("weight", shape=[120, 84])

b\_fc\_2 = tf.get\_variable("bias", shape=[84], trainable=True)

fc\_2 = tf.nn.xw\_plus\_b(act\_fc\_1, w\_fc\_2, b\_fc\_2)

act\_fc\_2 = tf.nn.sigmoid(fc\_2)

#全连接3

with tf.variable\_scope("fc\_3") as scope:

w\_fc\_3 = tf.get\_variable("weight", shape=[84, 10])

b\_fc\_3 = tf.get\_variable("bias", shape=[10], trainable=True)

tf.summary.histogram("weight", w\_fc\_3)

tf.summary.histogram("bias", b\_fc\_3)

fc\_3 = tf.nn.xw\_plus\_b(act\_fc\_2, w\_fc\_3, b\_fc\_3)

return fc\_3

def train():

# 1. 输入数据的占位符

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [BATCH\_SIZE, 10])

# 2. 初始化LeNet模型，构造输出标签y\_

le = LeNet()

y\_ = le.create(x)

# 3. 损失函数，使用交叉熵作为损失函数

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=y\_, labels=y))

# 4. 优化函数，首先声明I个优化函数，然后调用minimize去最小化损失函数

optimizer = tf.train.AdamOptimizer()

train\_op = optimizer.minimize(loss)

# 5. summary用于数据保存，用于tensorboard可视化

tf.summary.scalar("loss", loss)

# 6. 构造验证函数，如果对应位置相同则返回true，否则返回false

correct\_pred = tf.equal(tf.argmax(y\_, 1), tf.argmax(y, 1))

# 7. 通过tf.cast把true、false布尔型的值转化为数值型，分别转化为1和0，然后相加就是判断正确的数量

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

merged = tf.summary.merge\_all()

writer = tf.summary.FileWriter("logs")

# 8. 初始化一个saver，用于后面保存训练好的模型

saver = tf.train.Saver()

with tf.Session() as sess:

sess.run((tf.global\_variables\_initializer()))

writer.add\_graph(sess.graph)

i = 0

for epoch in range(5):

for step in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(BATCH\_SIZE)

summary, loss\_value, \_ = sess.run(([merged, loss, train\_op]),

feed\_dict={x: batch\_xs,

y: batch\_ys})

print("epoch : {}----loss : {}".format(epoch, loss\_value))

writer.add\_summary(summary, i)

i += 1

# 验证准确率

test\_acc = 0

test\_count = 0

for \_ in range(10):

batch\_xs, batch\_ys = mnist.test.next\_batch(BATCH\_SIZE)

acc = sess.run(accuracy, feed\_dict={x: batch\_xs, y: batch\_ys})

test\_acc += acc

test\_count += 1

print("accuracy : {}".format(test\_acc / test\_count))

saver.save(sess, os.path.join("temp", "mode.ckpt"))

train()

训练完之后

在tensorboard上可视化

大概在数据集1K-1.5K左右损失逐渐收敛

