# 验证码识别

验证码（CAPTCHA）是“Completely Automated Public Turing test to tell Computers and Humans Apart”（全自动区分[计算机](http://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA" \t "_blank)和人类的[图灵测试](http://baike.baidu.com/item/%E5%9B%BE%E7%81%B5%E6%B5%8B%E8%AF%95" \t "_blank)）的缩写，是一种区分用户是计算机还是人的公共全自动[程序](http://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F" \t "_blank)。可以防止：恶意破解密码、[刷票](http://baike.baidu.com/item/%E5%88%B7%E7%A5%A8" \t "_blank)、论坛灌水，有效防止某个黑客对某一个特定注册用户用特定程序暴力破解方式进行不断的登陆尝试，实际上用验证码是现在很多网站通行的方式，我们利用比较简易的方式实现了这个功能。这个问题可以由计算机生成并评判，但是必须只有人类才能解答。由于计算机无法解答CAPTCHA的问题，所以回答出问题的用户就可以被认为是人类。

常见的数字型验证码变种多样。比如某招聘网站验证码，字母周围有噪点，字体扭曲。



1. 某招聘网站验证码

某电商网站验证码，不同样式，字母阴影，字母粘连，背景色干扰



1. 某电商网站验证码

某社交网站验证码，主体干扰线，背景色干扰，背景字母干扰，字体扭曲，字母粘连



1. 某社交网站验证码

本章主要以MNIST数据集为例子介绍识别数字型验证码。

本章1.1介绍验证码识别使用的数据集。

本章1.2介绍使用的特征提取方法，包括一维向量和二维向量。

本章1.3介绍使用的模型以及对应的验证结果，包括K近邻、支持向量机和深度学习。

本章演示代码请参考本书Github上的mnist.py文件。

## 数据集

验证码识别使用的数据集为MNIST数据集，MNIST是一个入门级的计算机视觉数据集，它包含各种手写数字图片如图1-4 所示：



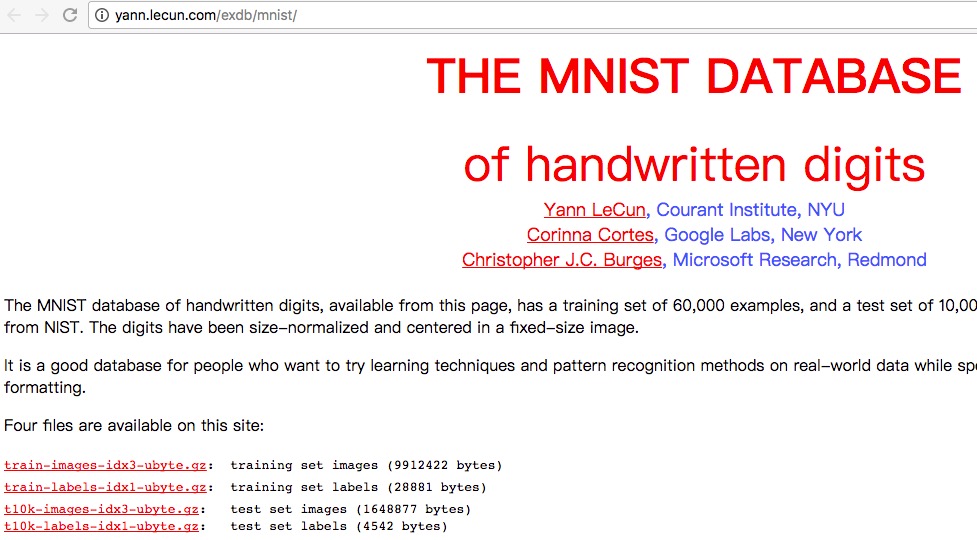
1. MNIST图片示例

它也包含每一张图片对应的标签，告诉我们这个是数字几。比如图1-4 这四张图片的标签分别是5，0，4，1。数据集包括60000行的训练数据集和10000行的测试数据集。每一个MNIST数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。每一张图片包含28X28个像素点，可以把这个数组展开成一个向量，长度是 28x28 = 784。

1. MNIST数据集合详解

|  |  |
| --- | --- |
| **文件名称** | **文件用途** |
| [train-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz) | 60000个图片训练样本 |
| [train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz) | 60000个图片训练样本的标注 |
| [t10k-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz) | 10000个图片测试样本 |
| [t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz) | 10000个图片测试样本的标注 |

MNIST的网址为<http://yann.lecun.com/exdb/mnist/>



1. MNIST官网

也可以使用离线版的MNIST文件，下载链接为：

http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz

文件读取方式为：

import pickle

import gzip

def load\_data():

with gzip.open('./mnist.pkl.gz') as fp:

training\_data, valid\_data, test\_data = pickle.load(fp)

return training\_data, valid\_data, test\_data

## 特征提取

### 方法一：一维向量

这也是MNIST默认的特征提供方式，将28乘以28的图片转换成了维度为784的一维向量。

import tflearn.datasets.mnist as mnist  
X, Y, testX, testY = mnist.load\_data(one\_hot=True)

### 方法二：二维向量

为了适应特定模型，需要将原本的一维向量转换回原有的28乘以28的二维向量，具体方法使用数组的reshape函数即可。

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

## 模型训练与验证

### 方法一：K近邻算法

特征提取方式使用一维向量，其中邻居数量n\_neighbors设置为15。

clf = neighbors.KNeighborsClassifier(n\_neighbors=15)

实例化K近邻，对训练数据集进行训练。

y\_pred = clf.predict(x\_test)  
print metrics.accuracy\_score(y\_test, y\_pred)

运行程序，经过训练，在测试数据集上预测准确率为96.28%。

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=15, p=2,

weights='uniform')

0.9628

### 方法二：支持向量机算法

特征提取方式使用一维向量，使用支持多分类的支持向量机。

clf = svm.SVC(decision\_function\_shape='ovo')

实例化支持向量机，对训练数据集进行训练。

y\_pred = clf.predict(x\_test)  
print metrics.accuracy\_score(y\_test, y\_pred)

运行程序，经过训练，在测试数据集上预测准确率为94.39%。

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovo', degree=3, gamma='auto', kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

0.9439

### 方法三：深度学习算法之DNN

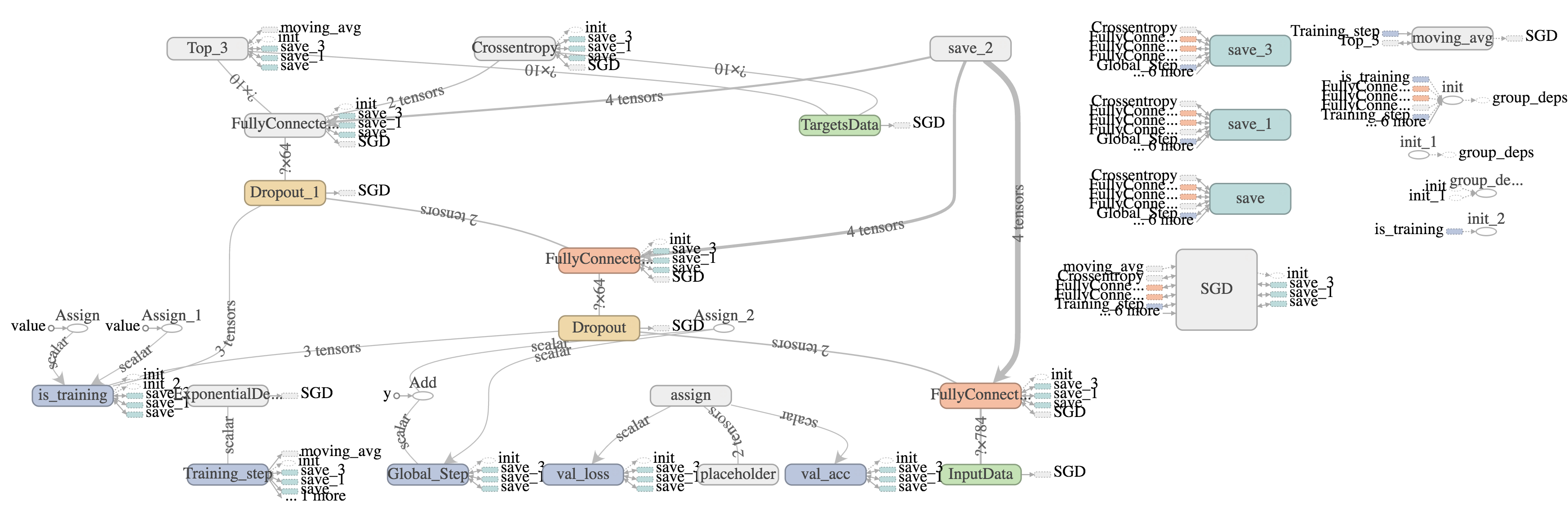
特征提取方式使用一维向量，定义输入层为维度为784的一维向量。

input\_layer = tflearn.input\_data(shape=[None, 784])

构造DNN模型，使用3个全连接层。

dense1 = tflearn.fully\_connected(input\_layer, 64, activation='tanh',  
 regularizer='L2', weight\_decay=0.001)  
dropout1 = tflearn.dropout(dense1, 0.8)  
dense2 = tflearn.fully\_connected(dropout1, 64, activation='tanh',  
 regularizer='L2', weight\_decay=0.001)  
dropout2 = tflearn.dropout(dense2, 0.8)  
softmax = tflearn.fully\_connected(dropout2, 10, activation='softmax')  
# Regression using SGD with learning rate decay and Top-3 accuracy  
sgd = tflearn.SGD(learning\_rate=0.1, lr\_decay=0.96, decay\_step=1000)  
top\_k = tflearn.metrics.Top\_k(3)  
net = tflearn.regression(softmax, optimizer=sgd, metric=top\_k,  
 loss='categorical\_crossentropy')

完整的DNN结构如图1-6 所示。



1. 用于识别MNIST的DNN结构图

实例化DNN算法并训练10轮。

# Training  
model = tflearn.DNN(net, tensorboard\_verbose=0)  
model.fit(X, Y, n\_epoch=10, validation\_set=(testX, testY),  
 show\_metric=True, run\_id="mnist")

运行程序，经过10轮训练，准确率达到了99.49%。

Training Step: 8600 | total loss: 0.28404 | time: 5.892s

| SGD | epoch: 010 | loss: 0.28404 - top3: 0.9842 | val\_loss: 0.12718 - val\_acc: 0.9949 -- iter: 55000/55000

### 方法四：深度学习算法之CNN

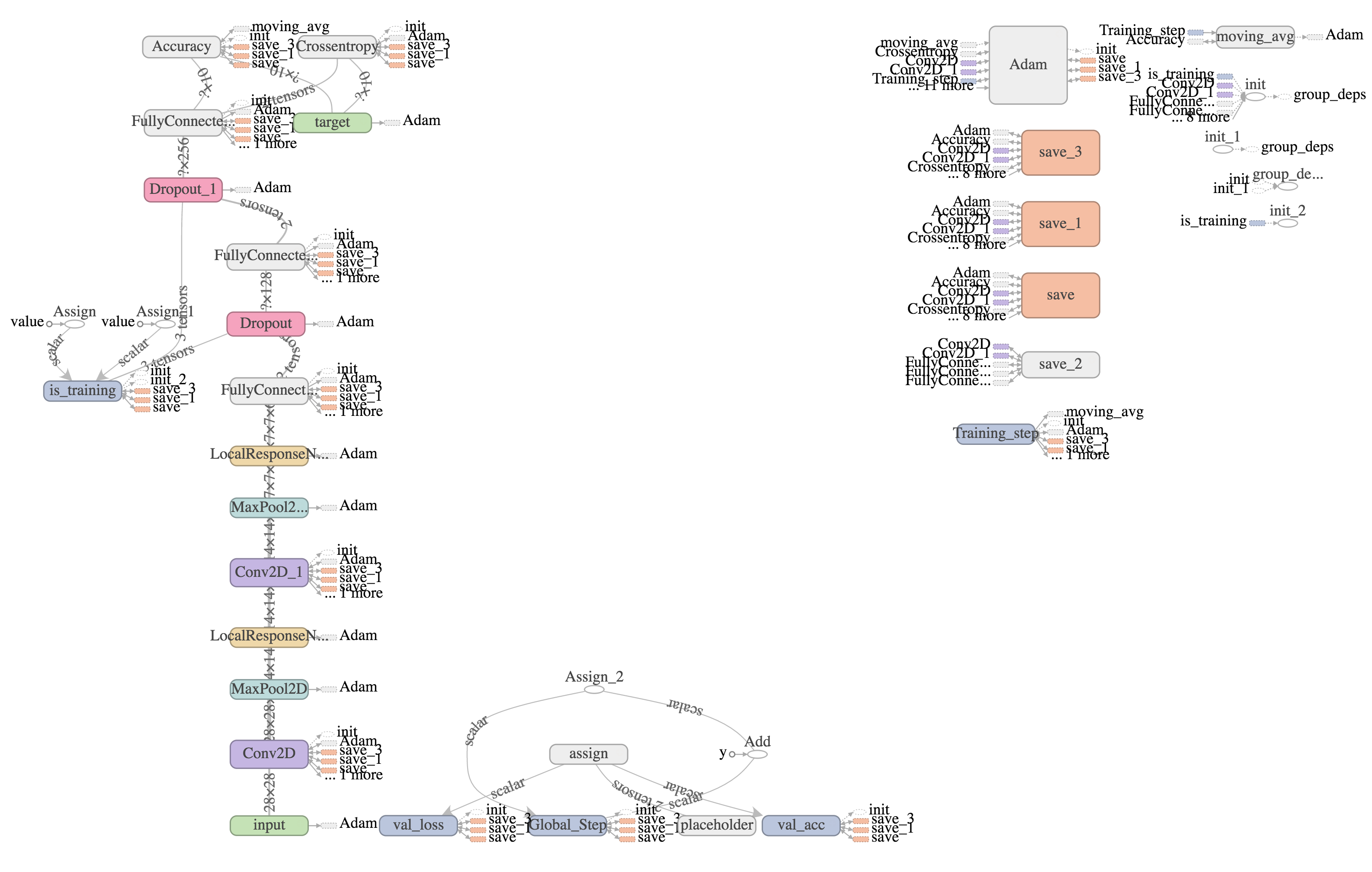
特征提取方式使用二维向量，定义输入层为28乘以28的二维向量。

network = input\_data(shape=[None, 28, 28, 1], name='input')

构造CNN模型，使用两层卷积两层全连接结构。

network = conv\_2d(network, 32, 3, activation='relu', regularizer="L2")  
network = max\_pool\_2d(network, 2)  
network = local\_response\_normalization(network)  
network = conv\_2d(network, 64, 3, activation='relu', regularizer="L2")  
network = max\_pool\_2d(network, 2)  
network = local\_response\_normalization(network)  
network = fully\_connected(network, 128, activation='tanh')  
network = dropout(network, 0.8)  
network = fully\_connected(network, 256, activation='tanh')  
network = dropout(network, 0.8)  
network = fully\_connected(network, 10, activation='softmax')  
network = regression(network, optimizer='adam', learning\_rate=0.01,  
 loss='categorical\_crossentropy', name='target')

完整的CNN结构如图1-7 所示。



1. 用于识别MNIST的CNN结构图

实例化CNN算法并训练10轮。

model = tflearn.DNN(network, tensorboard\_verbose=0)  
model.fit({'input': X}, {'target': Y}, n\_epoch=10,  
 validation\_set=({'input': testX}, {'target': testY}),  
 snapshot\_step=100, show\_metric=True, run\_id='convnet\_mnist')

运行程序，经过10轮训练，准确率达到了96.98%。

Training Step: 8600 | total loss: 0.39358 | time: 139.542s

| Adam | epoch: 010 | loss: 0.39358 - acc: 0.9366 | val\_loss: 0.12829 - val\_acc: 0.9698 -- iter: 55000/55000

## 本章小结

本章基于MNIST数据集介绍了验证码的识别方法。针对MNIST数据集，特征提取方法有一维向量和二维向量两种方法。训练模型介绍了K近邻、支持向量机以及深度学习的DNN和CNN算法，其中DNN算法表现优异，识别率达到了99%以上。

### 参考文献

* [https://github.com/tflearn/tflearn/blob/master/examples/images/](https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_mnist.py)
* <http://www.zhima365.com/shownews.php?id=139>
* <http://scikit-learn.org/stable/modules/svm.html#multi-class-classification>
* <http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py>
* http://www.2cto.com/article/201704/626783.html