1.5.2 CNN (LeNet-5) recognizes handwritten number

The following content between the following two dividing lines is what you need to do in the process of building your own development environment. You can skip this tutorial because the environment has been built.

Only for environment construction learning.

! Note, the operating environment must be Python2.

--Dividing line---

If prompted by the system, the "error: the cloud module is missing".

We can enter **sudo pip install cloud to install**.

First,we need to install the ipykernel package in python2

Input following command:

sudo pip install ipykernel

Then,we enter the following command to install the kernel.

sudo python2 -m ipykernel install --name python2

The first **python2** in this command indicates that this is the version of python installed by itself, which is used to distinguish it from python3.

The second **python2** is the name of the kernel displayed in jupyterlab after installing the kernel (can be customized).

Tensorflow website: https://www.tensorflow.org/

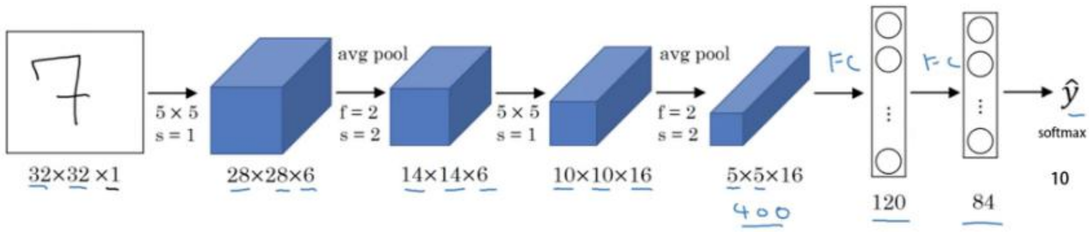
Solutions for Error reporting:

pip uninstall protobuf
pip uninstall google
pip install google
pip install protobuf
pip uninstall setuptools
pip install setuptools

--Dividing line---

The basic structure of CNN includes two layers:
1) Feature extraction layer.
2) Feature mapping layer.

In this course, we will use LeNet-5 (modern) to learn the recognition process of handwritten digits.



- Input layer

The image size is 32×32×1, 1 represents a black-white image, only one channel.

- Convolutional layer

The filter size is 5×5, the filter depth (number) is 6, padding is 0, size of the convolution step s = 1, size of the output matrix is 28×28×6, 6 represents the number of filters.

- Pooling layer

Average pooling, filter size 2 × 2 (f = 2), step size s = 2, no padding, size of output matrix is 14×14×6.

- Convolutional layer

The filter size is 5×5, the number of filters is 16, the padding is 0, size of the convolution step is s = 1, and size of output matrix is 10×10×16, 16 represents the number of filters.

- Pooling layer

Average pooling, filter size 2 × 2 (f = 2), size of step s = 2, no padding, size of output matrix is 5 × 5 × 16.
!Note, the end of this layer, the 5 × 5 × 16 matrix needs to be flattened into a 400-dimensional vector.

- Fully Connected layer (FC)

The number of neuron is 120.

- Fully Connected layer (FC)

The number of neuron is 84.

- Fully connected layer, output layer

The number of neurons in this layer is 10, 0~9 represents ten number categories

Properties of LeNet-5

- If the number of neural network layers does not include the input layer, LeNet-5 is a 7-layer network.
- LeNet-5 possess about 60,000 parameters.
- As the network gets deeper, the height and width of the image are shrinking. At the same time, the number of channels of image has been increasing.
- The currently used LeNet-5 structure is different from the structure proposed by

Professor Yann LeCun in the 1998 paper. For example, the use of the activation function, ReLU is used as the activation function, but the output layer will choose softmax generally.

MNIST_data

We can get MNIST data set on this website: http://yann.lecun.com/exdb/mnist/

It include four part.

Training set images: train-images-idx3-ubyte.gz (9.9 MB, after decompression 47 MB, include 60,000 samples)
Training set labels: train-labels-idx1-ubyte.gz (29 KB, after decompression 60 KB, include 60,000 labels)
Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, after decompression 7.8 MB, include 10,000 samples)
Test set labels: t10k-labels-idx1-ubyte.gz (5KB, after decompression 10 KB, include 10,000 labels)

Tensorflow project directory : https://github.com/tensorflow/tensorflow

Code path:

/home/pi/Yahboom_Project/1.OpenCV_course/05machine_learning/02CNN/CNN.ipynb

Steps to make a handwritten digital picture, please refer to last course. We need numbers in black and on a white background with a resolution of 28 * 28. The numbers must be fill the entire area.

```
# training part
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data", one_hot=True)

# Create nodes for input images and target output categories
x = tf.placeholder(tf.float32, shape=[None, 784]) # Data required for training
Placeholder
y_ = tf.placeholder(tf.float32, shape=[None, 10]) # Label data required for training
Placeholder

# ********* Construct a multi-layer convolutional network ************ #

# Initialize weights, offsets, convolutions, and pooling operations to avoid repeated
initialization operations when building models
def weight_variable(shape):
```

```
    # Take a random value, the mean is 0, and the standard deviation stddev is 0.1
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# The first parameter of # x is the number of pictures, the second and third parameters are the height and width of the picture, and the fourth parameter is the number of picture channels.
# W The first two parameters are the size of the convolution kernel, the third parameter is the number of image channels, and the fourth parameter is the number of convolution kernels
# strides is the convolution step size, the first and fourth parameters must be 1, because the step size of the convolution layer is only valid for the length and width of the matrix
# padding indicates the form of convolution, that is, whether to consider the boundary. "SAME" is to consider the boundary, when there is not enough to fill the surrounding with 0, "VALID" is not considered

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

# x The format of the parameter is the same as the x format in tf.nn.conv2d, ksize is the scale of the pooling layer filter, and strides is the filter step size
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# Change x to a 4-dimensional tensor, the first dimension represents the number of samples, the second and third dimensions represent the image length and width, and the fourth dimension represents the number of image channels
x_image = tf.reshape(x, [-1,28,28,1]) # -1 means any number of samples, the size is 28x28, and tensor with depth is 1

# First layer:convolution
W_conv1 = weight_variable([5, 5, 1, 32]) # Convolution calculates 32 features in each 5x5 patch.
b_conv1 = bias_variable([32])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

# Second layer: Pooling
h_pool1 = max_pool_2x2(h_conv1)
```

```
# Third layer: convolution
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

# Fourth layer: Pooling
h_pool2 = max_pool_2x2(h_conv2)

# Fifth layer: Fully connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Add dropout before the output layer to reduce overfitting
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Sixth layer: Fully connected layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

# Seventh layer: output layer
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# *************** Train and evaluate models *************** #

cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))

# Use backpropagation, use the optimizer to minimize the loss function
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

# Check whether our prediction matches the real label (the same index position
means matching)
# tf.argmax(y_conv,dimension), the subscript that returns the largest value is usually
used with tf.equal () to calculate the model accuracy
# dimension=0 Find by column    dimension=1 Find by row
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))

# Statistical test accuracy, convert the boolean value of correct_prediction to a
floating point number to represent right and wrong, and take the average value.
```

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

saver = tf.train.Saver() # Define saver

# *************** Start training *************** #
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for i in range(1000):
            batch = mnist.train.next_batch(50)
            if i%50 == 0:
                # Assess the accuracy of the model, Dropout is not used at this stage
                train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_: batch[1],
keep_prob: 1.0})
                print("step %d, training accuracy %g"%(i, train_accuracy))
            # Train the model, use 50% Dropout at this stage
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
        saver.save(sess, './model/model.ckpt') # Model storage location
        print("test accuracy %g"%accuracy.eval(feed_dict={x: mnist.test.images
[0:2000], y_: mnist.test.labels [0:2000], keep_prob: 1.0}))
```

```
# Official handwritten digit recognition
from PIL import Image
import tensorflow as tf
import numpy as np
import sys

im = Image.open('./images/3.png')
data = list(im.getdata())

result = [(255-x[0])*1.0/255.0 for x in data]
print(result)

# Create nodes for input images and target output categories
x = tf.placeholder("float", shape=[None, 784]) # Data required for training
Placeholder

# *********** Construct a multi-layer convolutional network ************ #
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1) # Take a random value, the mean
is 0, and the standard deviation stddev is 0.1
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

x_image = tf.reshape(x, [-1,28,28,1]) # -1 means any number of samples, the size is
28x28, and tensor with depth is 1

W_conv1 = weight_variable([5, 5, 1, 32]) # Convolution calculates 32 features in each
5x5 patch.

b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Add dropout before the output layer to reduce overfitting
keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# Fully connected layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

# Output layer
# tf.nn.softmax() Turn the output layer of the neural network into a probability
distribution
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

saver = tf.train.Saver() # Define saver

# *************** Start recognize *************** #
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
saver.restore(sess, "./model/model.ckpt")# The previously saved model parameters are used here

prediction = tf.argmax(y_conv,1)
predint = prediction.eval(feed_dict={x: [result],keep_prob: 1.0}, session=sess)

print("recognize result: %d" %predint[0])
sess.close()
```

Original picture：

Result, as shown below.

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0]
INFO:tensorflow:Restoring parameters from ./model/model.ckpt
recognize result: 3
```