1. 实验内容

用CIFA10数据集，在tensorflow框架下，搭建CNN模型，对每一层进行特征图可视化并分析。

1. CNN模型

设计的网络模型结构如图1所示，主要由输入层，2个卷积层，2个降采样层，全连接层，和输出层组成。

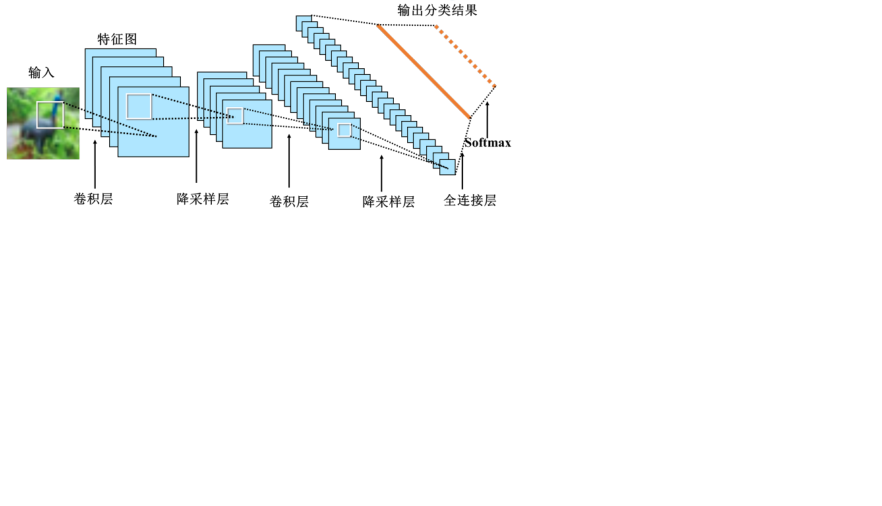


图1：网络模型结构图

1. 输入层：输入大小为32\*32的原图像，通道数为3（RGB）。
2. 卷积层1：进行第一次卷积，通道数由3变为32，图像尺寸不变。
3. 降采样层1：进行第一次降采样，图像尺寸由32\*32缩小为16\*16，通道数仍为32
4. 卷积层2：进行第二次卷积，通道数由32变为64，图像尺寸不变，仍为16\*16。
5. 降采样层2：进行第二次降采样，图像尺寸由16\*16缩小为8\*8，通道数仍为64
6. 将64个8\*8的图像转换为长度是4096的一维向量，该层有128个神经元。
7. 输出层：输出层共有10个神经元，对应10个类别。
8. 实验细节

对中间层细节进行输出显示，如图2至6所示。

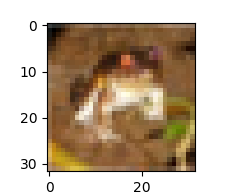


图2：输入的第一张图

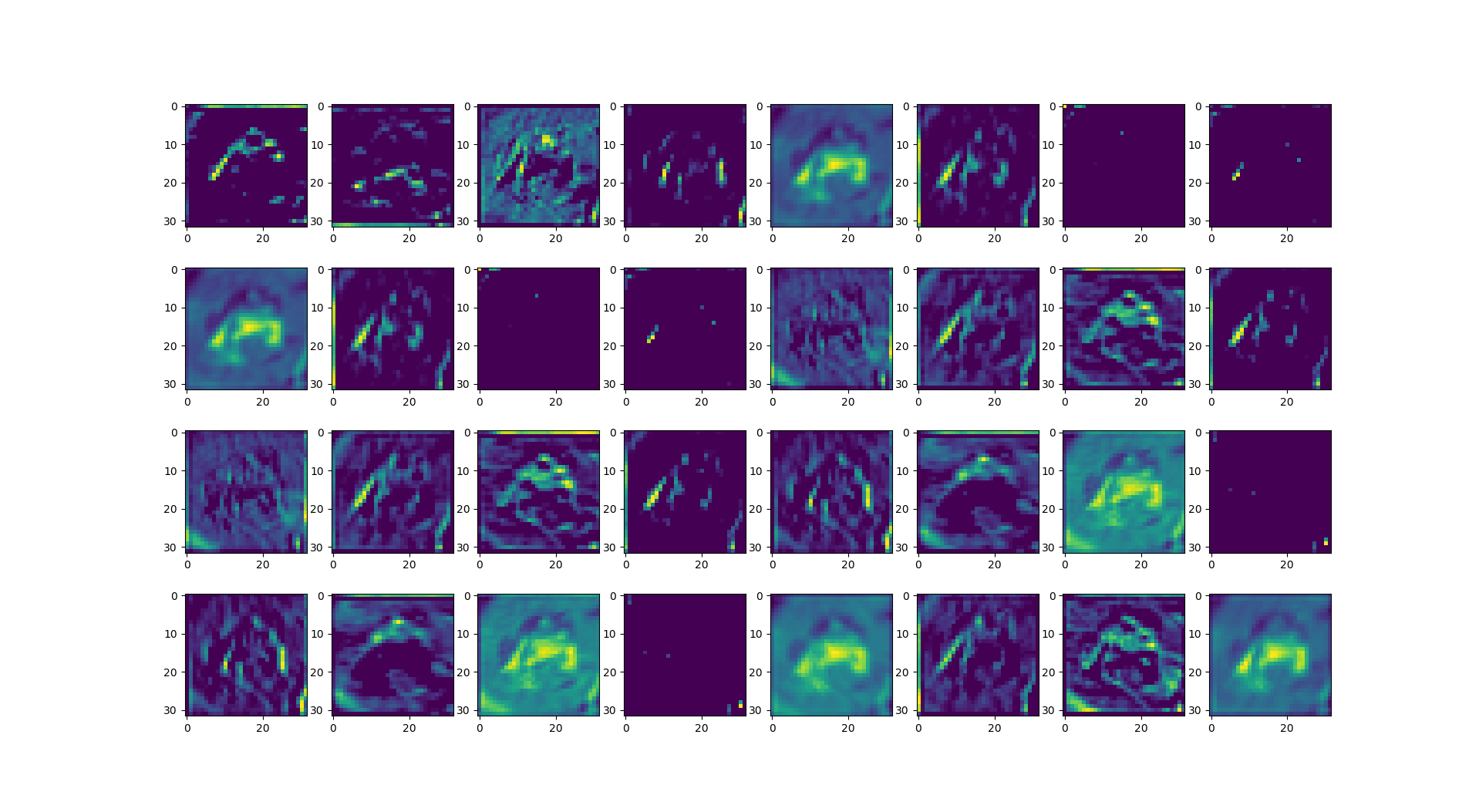


图3：第一层卷积后结果图

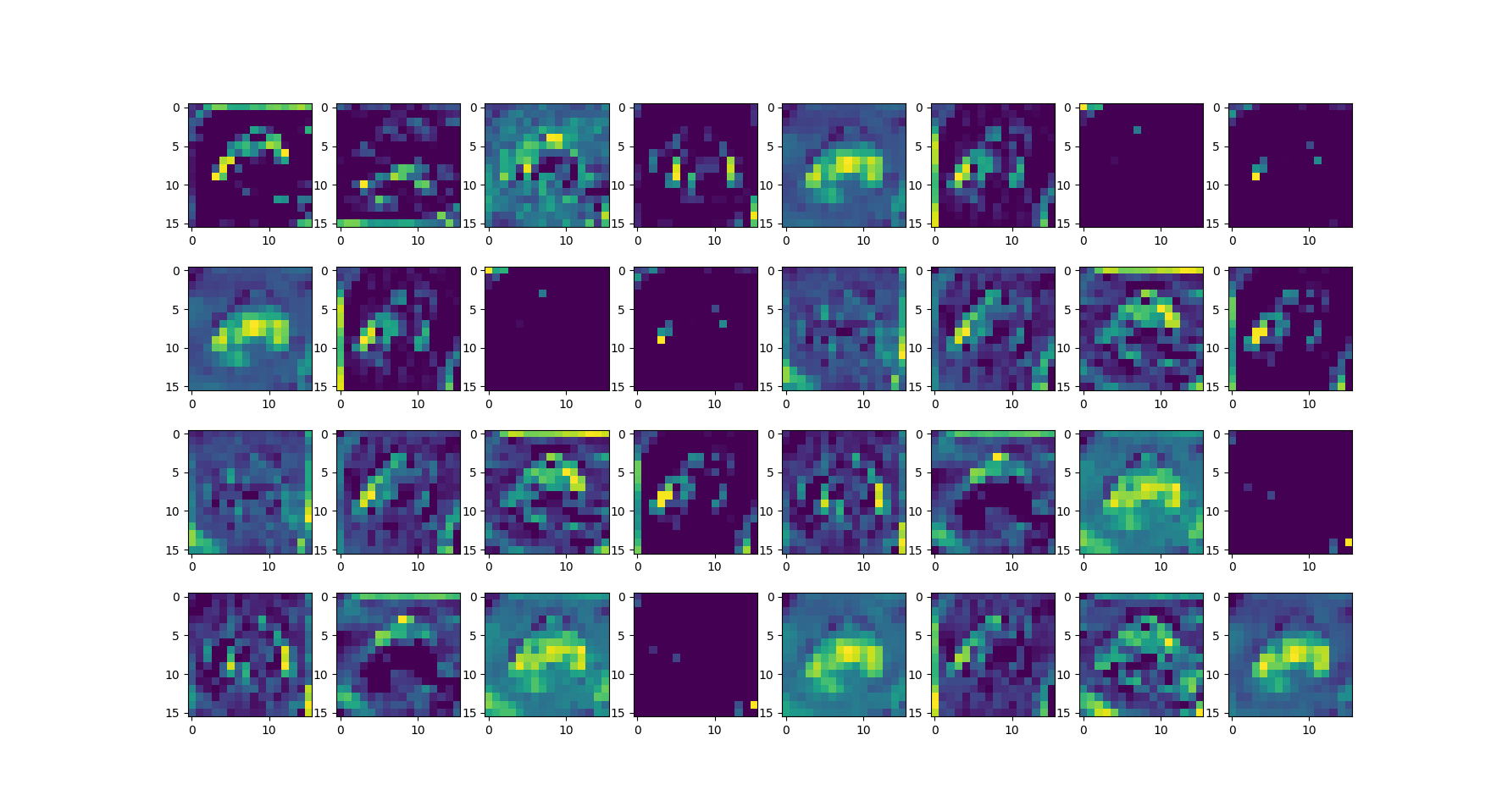


图4：第一次池化后结果图

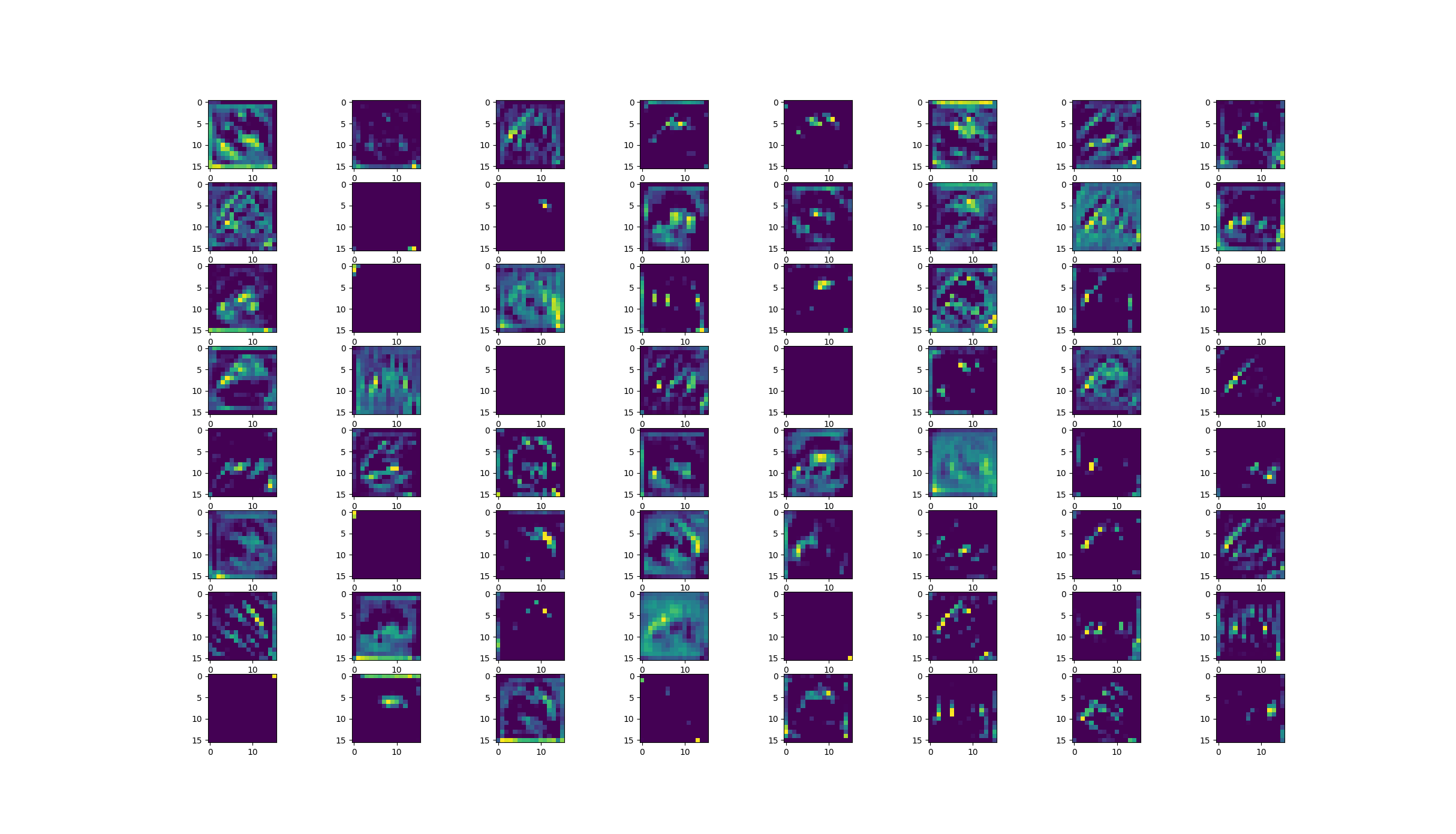


图5：第二层卷积后结果图

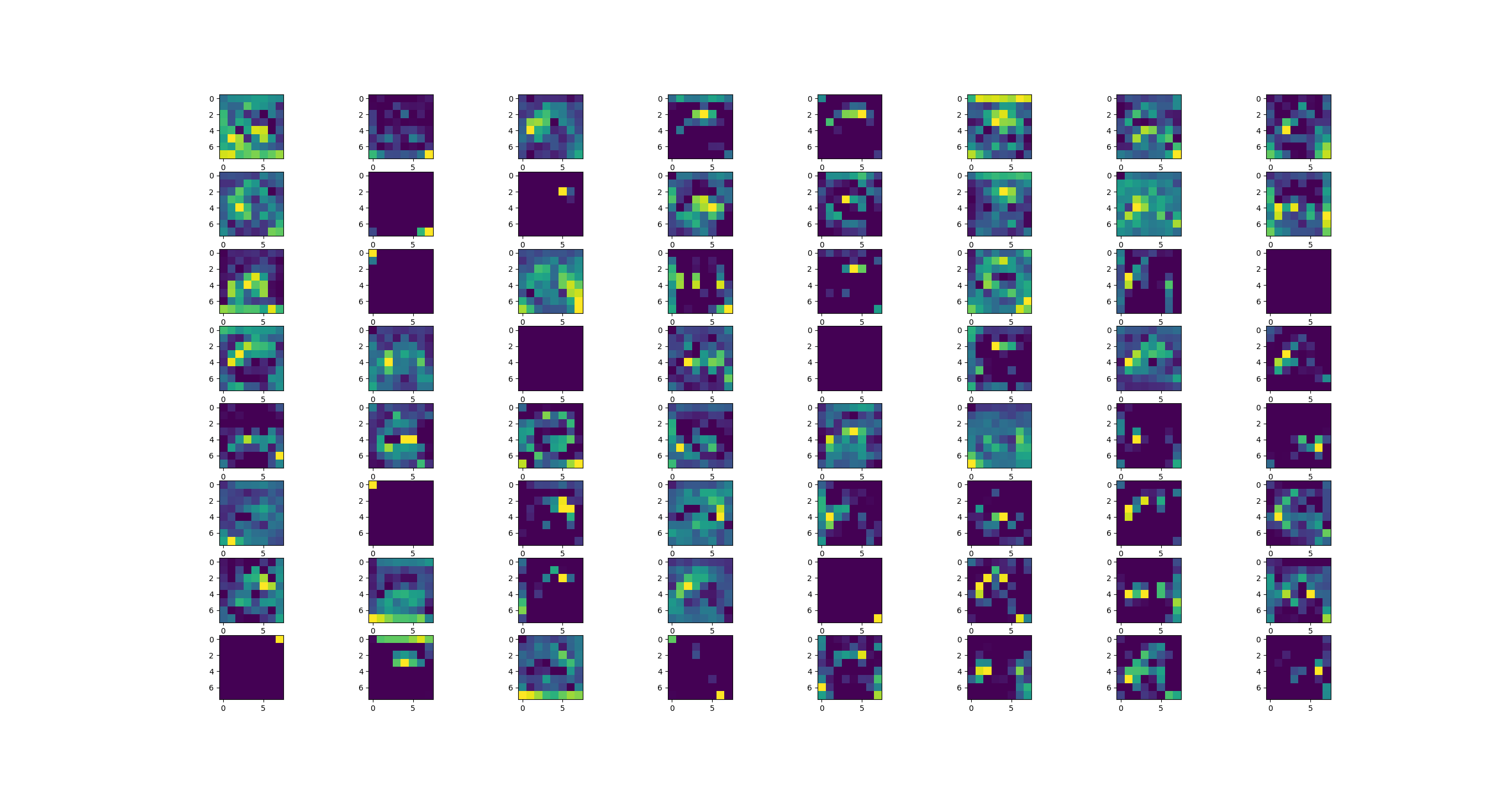


图6：第二次池化后结果图

可以发现，进行卷积和池化的过程，就是在不断提取输入图中青蛙的特征。

1. 实验结果

训练过程中，损失值和准确率如图7至8所示。

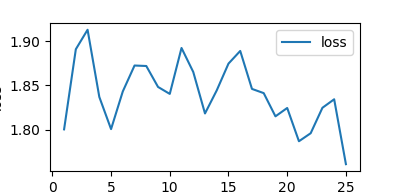


图7：训练过程中损失值变化

可以发现，随着训练轮次的增加，损失值整体趋势是减小的，并会趋向收敛。

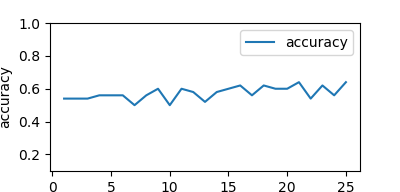


图8：训练过程中准确率变化

可以发现，随着训练轮次的增加，准确率整体是上升的，但上升趋势较慢。

1. 结果分析

最终在测试集的识别结果为0.89，仍有提升空间。比如可以调整batch size,训练的轮数，CNN的层数等。

1. 程序

import urllib.request

import os

import tarfile

import numpy as np

import pickle as p

import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder

import tensorflow as tf

tf.reset\_default\_graph()

from time import time

#下载

url = 'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz'

filepath = 'data/cifar-10-python.tar.gz'

if not os.path.isfile(filepath):

result = urllib.request.urlretrieve(url, filepath)

print('downloaded:', result)

else:

print('Data file already exists.')

#解压

if not os.path.exists("data/cifar-10-batches-py"):

tfile = tarfile.open("data/cifar-10-python.tar.gz", 'r:gz')

result = tfile.extractall('data/')

print('Extracted to ./data/cifar-10-batches-py/')

else:

print('Dictionary already exists.')

#导入CIFAR数据集

def load\_CIFAR\_batch(filename):

"""load single batch of cifar"""

with open(filename, 'rb') as f:

#一个样本由标签和图像数据组成

#3072 = 32 x 32 x 3

data\_dict = p.load(f, encoding= 'bytes')

images = data\_dict[b'data']

labels = data\_dict[b'labels']

#把原始数据结构调整为BCWH batches, channels, width, height

images = images.reshape(10000, 3, 32, 32)

#tensorflow 处理图像数据的结构：BWHC

#把C移动到最后一个维度

images = images.transpose(0, 2, 3, 1)

labels = np.array(labels)

return images, labels

def load\_CIFAR\_data(data\_dir):

"""load CIFAR data"""

images\_train = []

labels\_train = []

for i in range(5):

f = os.path.join(data\_dir, 'data\_batch\_%d' %(i+1))

print('loading ', f)

#调用load\_CIFAR\_batch()获取批量的图像及其对应的标签

image\_batch, label\_batch = load\_CIFAR\_batch(f)

images\_train.append(image\_batch)

labels\_train.append(label\_batch)

Xtrain = np.concatenate(images\_train)

Ytrain = np.concatenate(labels\_train)

del image\_batch, label\_batch

Xtest, Ytest = load\_CIFAR\_batch(os.path.join(data\_dir, 'test\_batch'))

print('finished loading CIFAR-10 data')

#返回训练集和测试集的图像和标签

return Xtrain, Ytrain, Xtest, Ytest

data\_dir = 'data/cifar-10-batches-py/'

Xtrain, Ytrain, Xtest, Ytest = load\_CIFAR\_data(data\_dir)

#显示数据集信息

print('training data shape:', Xtrain.shape)

print('training labels shape:', Ytrain.shape)

print('test data shape:', Xtest.shape)

print('test labels shape:', Ytest.shape)

#查看单项image 和 label

#plt.imshow(Xtrain[0])

#print(Ytrain[0])

#plt.show()

#定义标签字典，每一个数字所代表的图像类别名称

label\_dict = {0:"airplane", 1:"automobile", 2:"bird", 3:"cat", 4:"deer",

5:"dog", 6:"frog", 7:"horse", 8:"ship", 9:"trunk"}

#查看多项images 与 labels

def plot\_images\_labels\_prediction(images,

labels,

prediction,

idx,

num = 10):

fig = plt.gcf()

fig.set\_size\_inches(12, 6)

if num > 10 :

num = 10

for i in range(0, num):

ax = plt.subplot(2, 5, i+1)

ax.imshow(images[idx], cmap='binary')

title = str(i) + ',' +label\_dict[labels[idx]]

if len(prediction) > 0 :

title += '=>' + label\_dict[prediction[idx]]

ax.set\_title(title, fontsize = 10)

idx += 1

plt.show()

#先是图像数据及对应标签

#plot\_images\_labels\_prediction(Xtest, Ytest, [], 1, 10)

#数据预处理

#将图像进行数字标准化

Xtrain\_normalize = Xtrain.astype('float32') / 255.0

Xtest\_normalize = Xtest.astype('float32') / 255.0

encoder = OneHotEncoder(sparse= False)

#标签数据转为one-hot编码

yy = [[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]]

encoder.fit(yy)

Ytrain\_reshape = Ytrain.reshape(-1, 1)

Ytrain\_onehot = encoder.transform(Ytrain\_reshape)

Ytest\_reshape = Ytrain.reshape(-1, 1)

Ytest\_onehot = encoder.transform(Ytest\_reshape)

#定义共享函数

#定义权值

def weight(shape):

#使用截断的正态分布 生成标准差为0.1的随机数来初始化权值

return tf.Variable(tf.truncated\_normal(shape, stddev=0.1), name='W')

#定义偏置

#初始化为0.1

def bias(shape):

return tf.Variable(tf.constant(0.1, shape=shape), name='b')

#定义卷积操作

#步长为1，padding 为same（填充）valid(不填充）

def conv2d(x, W):

#tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)

#strides为长度为4的一阶张量，并且要求strides[0]=strides[3]=1，

# strides[1]，strides[2]决定卷积核在输入图像in\_hight，in\_width方向的滑动步长，

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

#定义池化操作

#步长为2，即原尺寸的长和宽各除以2

def max\_pool\_2x2(x):

#tf.nn.max\_pool(value, ksize, strides, padding, data\_format="NHWC", name=None)

#ksize:池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]，

# 因为我们不想在batch和channels上做池化，所以这两个维度设为了1

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1],padding='SAME')

#定义网络结构

#输入层

#32\*32图像，通道为3(RGB)

with tf.name\_scope('input\_layer'):

x = tf.placeholder('float', shape= [None, 32, 32, 3], name="x")

#第1个卷积层

#输入通道：3，输出通道：32，卷积后图像尺寸不变，依然是32x32

with tf.name\_scope('conv\_1'):

W1 = weight([3, 3, 3, 32])#[k\_width, k\_height, input\_chn, output\_chn]

b1 = bias([32]) #与output\_chn一致

conv\_1 = conv2d(x, W1) + b1

conv\_1 = tf.nn.relu(conv\_1)

#第1个池化层

#将32\*32图像缩小为16\*16，池化不改变通道数量，因此依然是32个

with tf.name\_scope('pool\_1'):

pool\_1 = max\_pool\_2x2(conv\_1)

#第2个卷积层

#输入通道：32，输出通道：64，卷积后图像尺寸不变，依然是16x16

with tf.name\_scope('conv\_2'):

W2 = weight([3, 3, 32, 64])

b2 = bias([64])

conv\_2 = conv2d(pool\_1, W2) + b2

conv\_2 = tf.nn.relu(conv\_2)

#第2个池化层

#将16\*16图像缩小为8\*8，池化不改变通道数量，因此依然是64个

with tf.name\_scope('pool\_2'):

pool\_2 = max\_pool\_2x2(conv\_2)

#全连接层

#将第二个池化层的64个8\*8的图像转化为一维向量，长度是64\*8\*8 = 4096

#128个神经元

with tf.name\_scope('fc'):

W3 = weight([4096, 128])

b3 = bias([128])

flat = tf.reshape(pool\_2, [-1, 4096])

h = tf.nn.relu(tf.matmul(flat, W3) + b3)

h\_dropout = tf.nn.dropout(h, keep\_prob = 0.8)

#输出层

#输出层共有10个神经元，对应0-9这10个类别

with tf.name\_scope('output\_layer'):

W4 = weight([128, 10])

b4 = bias([10])

pred = tf.nn.softmax(tf.matmul(h\_dropout, W4) + b4)

#构建模型

with tf.name\_scope("optimizer"):

y = tf.placeholder("float", shape= [None, 10],

name="label")

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

#选择优化器

optimizer = tf.train.AdamOptimizer(learning\_rate = 0.0001).minimize(loss\_function)

#定义准确率

with tf.name\_scope("evaluation"):

correct\_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

train\_epochs = 25

batch\_size = 50

total\_batch = int(len(Xtrain)/batch\_size)

epoch\_list = []

accuracy\_list = []

loss\_list = []

epoch = tf.Variable(0, name='epoch', trainable=False)

startTime = time()

sess = tf.Session()

init = tf.global\_variables\_initializer()

sess.run(init)

#断点续训

#设置检查点存储目录

ckpt\_dir = "CIFAR\_log/"

if not os.path.exists(ckpt\_dir):

os.makedirs(ckpt\_dir)

#生成saver,max\_to\_keep 参数，这个是用来设置保存模型的个数，默认为5

saver = tf.train.Saver(max\_to\_keep= 1)

#如果有检查点文件，读取最新的检查点文件，恢复各种变量值

ckpt = tf.train.latest\_checkpoint(ckpt\_dir)

if ckpt != None:

saver.restore(sess, ckpt) #加载所有的参数

#从这里开始就可以直接使用模型进行预测，或者接着继续预测了

else:

print("Training from scratch.")

#获取续训参数

start = sess.run(epoch)

if start >= 24:

start = 0

print("Training starts from {} epoch.".format(start + 1))

#迭代训练

def get\_train\_batch(number, batch\_size):

return Xtrain\_normalize[number\*batch\_size:(number + 1)\*batch\_size],\

Ytrain\_onehot[number\*batch\_size:(number + 1)\*batch\_size]

for ep in range(start, train\_epochs):

for i in range(total\_batch):

batch\_x, batch\_y = get\_train\_batch(i, batch\_size)

sess.run(optimizer, feed\_dict={x: batch\_x, y: batch\_y})

if i % 100 == 0:

print("Step{}".format(i), "finished")

loss, acc = sess.run([loss\_function, accuracy], feed\_dict={x: batch\_x, y: batch\_y})

epoch\_list.append(ep + 1)

loss\_list.append(loss)

accuracy\_list.append(acc)

print("Train epoch:", '%02d' %(sess.run(epoch) + 1), "Loss = ",'{:.6f}'.format(loss), "Accuracy = ", acc)

#保存检查点

saver.save(sess, ckpt\_dir + "CIFAR10\_cnn\_model.cpkt", global\_step = ep + 1)

sess.run(epoch.assign(ep + 1))

duration = time() - startTime

print("Train finished takes:", duration)

#可视化损失值

fig = plt.gcf()

fig.set\_size\_inches(4, 2)

plt.plot(epoch\_list, loss\_list, label = 'loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['loss'], loc = 'upper right')

plt.show()

#可视化准确率

plt.plot(epoch\_list, accuracy\_list, label = 'accuracy')

fig = plt.gcf()

fig.set\_size\_inches(4, 2)

plt.ylim(0.1, 1)

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend()

plt.show()

#评估模型及预测

test\_total\_batch = int(len(Xtest\_normalize)/batch\_size)

test\_acc\_sum = 0.0

for i in range(test\_total\_batch):

test\_image\_batch = Xtest\_normalize[i\*batch\_size:(i+1)\*batch\_size]

test\_label\_batch = Ytest\_onehot[i\*batch\_size:(i+1)\*batch\_size]

test\_batch\_acc = sess.run(accuracy, feed\_dict={x:test\_image\_batch, y:test\_label\_batch})

test\_acc\_sum += test\_batch\_acc

test\_acc = float(test\_acc\_sum/test\_total\_batch)

print("Test accuracy:{:.6f}".format(test\_acc))

test\_pred = sess.run(pred, feed\_dict={x:Xtest\_normalize[:10]})

prediction\_result = sess.run(tf.argmax(test\_pred, 1))

plot\_images\_labels\_prediction(Xtest, Ytest, prediction\_result, 0, 10)

plt.show()

# ----------------------------------各个层特征可视化-------------------------------

# imput image

fig2, ax2 = plt.subplots(figsize=(2, 2))

ax2.imshow(np.reshape(Xtrain[0], (32, 32, 3)))

plt.show()

# 第一层的卷积输出的特征图

input\_image = np.reshape(Xtrain[0], (1, 32, 32, 3))

conv1\_16 = sess.run(conv\_1, feed\_dict={x: input\_image})

conv1\_transpose = sess.run(tf.transpose(conv1\_16, [3, 0, 1, 2]))

print(conv1\_transpose.shape) #32x1x32x32

fig3, ax3 = plt.subplots(nrows=4, ncols=8, figsize=(8, 4))

for i in range(4):

for j in range(8):

ax3[i][j].imshow(conv1\_transpose[i\*4+j][0]) # tensor的切片[row, column]

plt.show()

# 第一层池化后的特征图

pool1\_16 = sess.run(pool\_1, feed\_dict={x: input\_image})

pool1\_transpose = sess.run(tf.transpose(pool1\_16, [3, 0, 1, 2]))

fig4, ax4 = plt.subplots(nrows=4, ncols=8, figsize=(8, 4))

print(pool1\_transpose.shape) #32x1x16x16

#plt.title('Pool2 32x16x16')

for i in range(4):

for j in range(8):

ax4[i][j].imshow(pool1\_transpose[i\*4+j][0])

plt.show()

# 第二层卷积输出特征图

conv2\_32 = sess.run(conv\_2, feed\_dict={x: input\_image})

conv2\_transpose = sess.run(tf.transpose(conv2\_32, [3, 0, 1, 2]))

fig5, ax5 = plt.subplots(nrows=8, ncols=8, figsize=(8, 8))

print(conv2\_transpose.shape) #64x1x16x16

#plt.title('Conv2 64x16x16')

for i in range(8):

for j in range(8):

ax5[i][j].imshow(conv2\_transpose[i\*8+j][0])

plt.show()

# 第二层池化后的特征图

pool2\_32 = sess.run(pool\_2, feed\_dict={x: input\_image})

pool2\_transpose = sess.run(tf.transpose(pool2\_32, [3, 0, 1, 2]))

fig6, ax6 = plt.subplots(nrows=8, ncols=8, figsize=(8, 8))

print(pool2\_transpose.shape) #64x1x8x8

#plt.title('Pool2 64x8x8')

for i in range(8):

for j in range(8):

ax6[i][j].imshow(pool2\_transpose[i\*8+j][0])

plt.show()