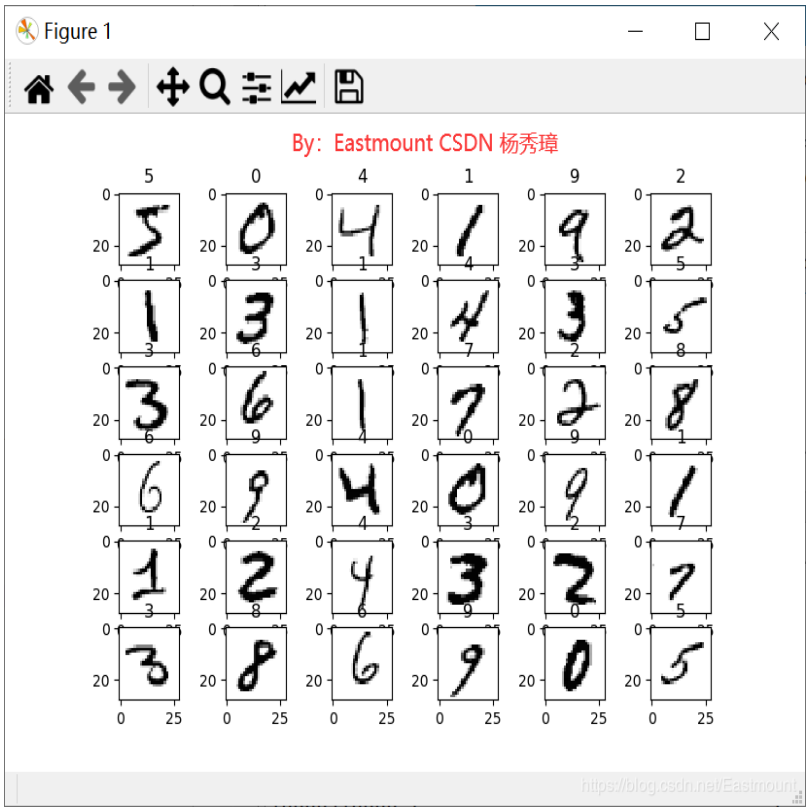
从本专栏开始，作者正式研究Python深度学习、神经网络及人工智能相关知识。前一篇文章详细讲解了Keras环境搭建、入门基础及回归神经网络案例。本篇文章将通过Keras实现分类学习，以MNIST数字图片为例进行讲解。基础性文章，希望对您有所帮助！



本专栏主要结合作者之前的博客、AI经验和相关视频（强推"莫烦大神"视频）及论文介绍，后面随着深入会讲解更多的Python人工智能案例及应用。基础性文章，希望对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作者作为人工智能的菜鸟，希望大家能与我在这一笔一划的博客中成长起来。写了这么多年博客，尝试第一个付费专栏，但更多博客尤其基础性文章，还是会继续免费分享，但该专栏也会用心撰写，望对得起读者，共勉！

代码下载地址：https://github.com/eastmountyxz/AI-for-TensorFlow
代码下载地址：https://github.com/eastmountyxz/AI-for-Keras

PS：百度网盘链接总被下线，需要的私聊我，或从CSDN、Github下载。

文章目录

同时推荐前面作者另外五个Python系列文章。从2014年开始，作者主要写了三个Python系列文章，分别是基础知识、网络爬虫和数据分析。2018年陆续增加了Python图像识别和Python人工智能专栏。

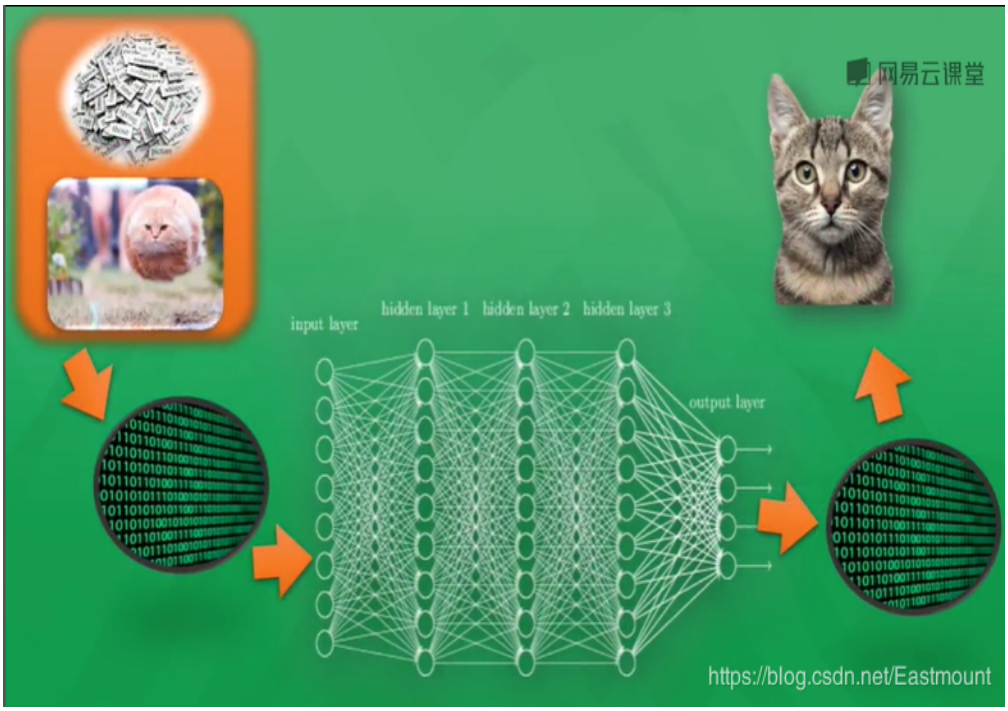
- Python基础知识系列：Python基础知识学习与提升
- Python网络爬虫系列：Python爬虫之Selenium+BeautifulSoup+Requests
- Python数据分析系列：知识图谱、web数据挖掘及NLP
- Python图像识别系列：Python图像处理及图像识别
- Python人工智能系列：Python人工智能及知识图谱实战

前文：
[Python人工智能] 一.TensorFlow2.0环境搭建及神经网络入门
[Python人工智能] 二.TensorFlow基础及一元直线预测案例
[Python人工智能] 三.TensorFlow基础之Session、变量、传入值和激励函数
[Python人工智能] 四.TensorFlow创建回归神经网络及Optimizer优化器
[Python人工智能] 五.Tensorboard可视化基本用法及绘制整个神经网络
[Python人工智能] 六.TensorFlow实现分类学习及MNIST手写体识别案例
[Python人工智能] 七.什么是过拟合及dropout解决神经网络中的过拟合问题
[Python人工智能] 八.卷积神经网络CNN原理详解及TensorFlow编写CNN
[Python人工智能] 九.gensim词向量Word2Vec安装及《庆余年》中文短文本相似度计算
[Python人工智能] 十.Tensorflow+Opencv实现CNN自定义图像分类案例及与机器学习KNN图像分类算法对比
[Python人工智能] 十一.Tensorflow如何保存神经网络参数
[Python人工智能] 十二.循环神经网络RNN和LSTM原理详解及TensorFlow编写RNN分类案例
[Python人工智能] 十三.如何评价神经网络、loss曲线图绘制、图像分类案例的F值计算
[Python人工智能] 十四.循环神经网络LSTM RNN回归案例之sin曲线预测
[Python人工智能] 十五.无监督学习Autoencoder原理及聚类可视化案例详解
[Python人工智能] 十六.Keras环境搭建、入门基础及回归神经网络案例
《人工智能狂潮》读后感——什么是人工智能？(一)

一.什么是分类学习

1.Classification

我们之前文章解决的都是回归问题，它预测的是一个连续分布的值，例如房屋的价格、汽车的速度、Pizza的价格等。而当我们遇到需要判断一张图片是猫还是狗时，就不能再使用回归解决了，此时需要通过分类学习，把它分成计算机能够识别的那一类（猫或狗）。

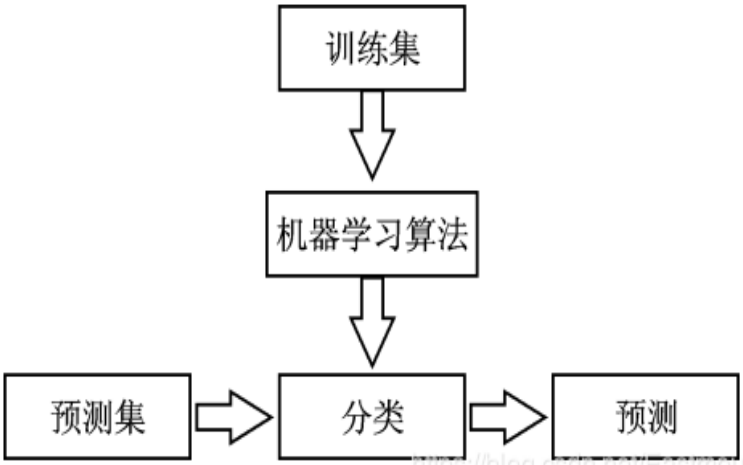


如上图所示，通常来说，计算机处理的东西和人类有所不同，无论是声音、图片还是文字，它们都只能以数字0或1出现在计算机神经网络里。神经网络看到的图片其实都是一堆数字，对数字的加工处理最终生成另一堆数字，并且具有一定认知上的意义，通过一点点的处理能够得知计算机到底判断这张图片是猫还是狗。

分类（Classification） 属于有监督学习中的一类，它是数据挖掘、机器学习和数据科学中一个重要的研究领域。分类模型类似于人类学习的方式，通过对历史数据或训练集的学习得到一个目标函数，再用该目标函数预测新数据集的未知属性。分类模型主要包括两个步骤：

- 训练。给定一个数据集，每个样本都包含一组特征和一个类别信息，然后调用分类算法训练模型。
- 预测。利用生成的模型对新的数据集（测试集）进行分类预测，并判断其分类结果。

通常为了检验学习模型的性能会使用校验集。数据集会被分成不相交的训练集和测试集，训练集用来构造分类模型，测试集用来检验多少类标签被正确分类。



那么，回归和分类有什么区别呢?

分类和回归都属于监督学习，它们的区别在于：回归是用来预测连续的实数值，比如给定了房屋面积来预测房屋价格，返回的结果是房屋价格；而分类是用来预测有限的离散值，比如判断一个人是否患糖尿病，返回值是“是”或“否”。也就是说，明确对象属于哪个预定义的目标类，预定义的目标类是离散值时为分类，连续值时为回归。

2.MNIST

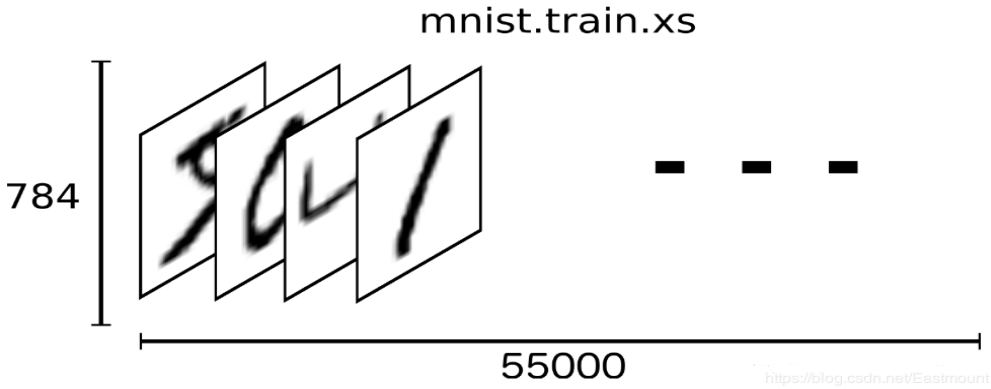
MNIST是手写体识别数据集，它是非常经典的一个神经网络示例。MNIST图片数据集包含了大量的数字手写体图片，如下图所示，我么可以尝试用它进行分类实验。

MNIST数据集是含标注信息的，上图分别表示数字5、0、4和1。该数据集共包含三部分：

- 训练数据集：55,000个样本，mnist.train
- 测试数据集：10,000个样本，mnist.test
- 验证数据集：5,000个样本，mnist.validation

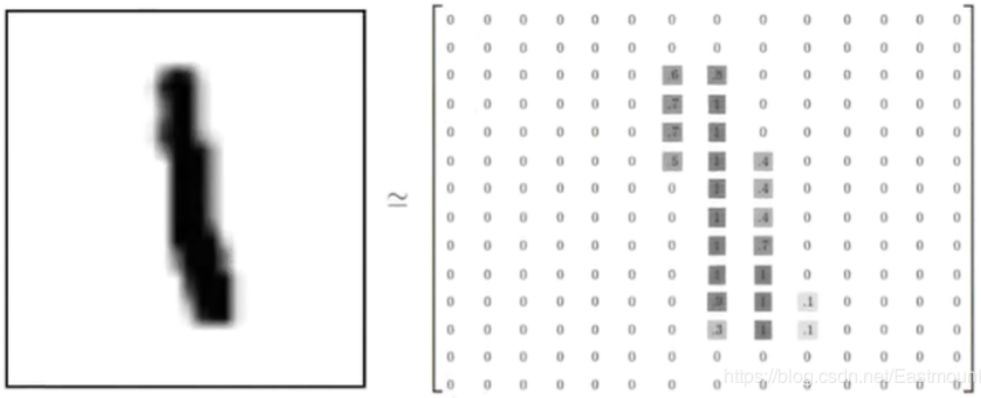
通常，训练数据集用来训练模型，验证数据集用来检验所训练出来的模型的正确性和是否过拟合，测试集是不可见的（相当于一个黑盒），但我们最终的目的是使得所训练出来的模型在测试集上的效果（这里是准确性）达到最佳。

如下图所示，数据是以该形式被计算机所读取，比如28*28=784个像素点，白色的地方都是0，黑色的地方表示有数字的，总共有55000张图片。



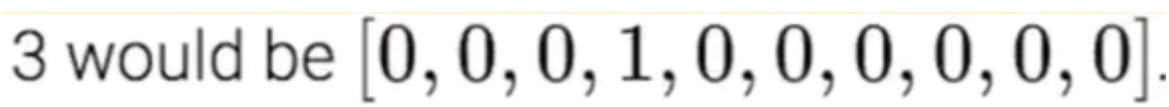
MNIST数据集中的一个样本数据包含两部分内容：手写体图片和对应的label。这里我们用xs和ys分别代表图片和对应的label，训练数据集和测试数据集都有xs和ys，使用mnist.train.images和mnist.train.labels表示训练数据集中图片数据和对应的label数据。

如下图所示，它表示由28*28的像素点矩阵组成的一张图片，这里的数字784（28*28）如果放在我们的神经网络中，它就是x输入的大小，其对应的矩阵如下图所示，类标label为1。

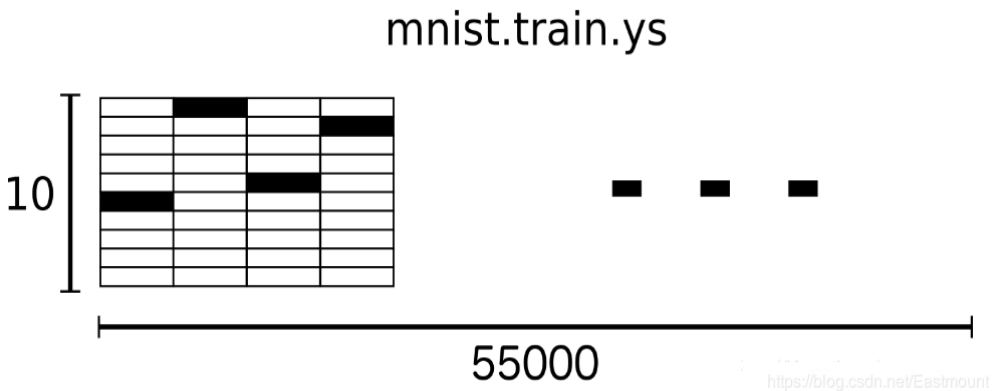


最终MNIST的训练数据集形成了一个形状为55000*784位的tensor，也就是一个多维数组，第一维表示图片的索引，第二维表示图片中像素的索引（tensor中的像素值在0到1之间）。

这里的y值其实是一个矩阵，这个矩阵有10个位置，如果它是1的话，它在1的位置（第2个数字）上写1，其他地方写0；如果它是2的话，它在2的位置（第3个数字）上写1，其他位置为0。通过这种方式对不同位置的数字进行分类，例如用[0,0,0,1,0,0,0,0,0,0]来表示数字3，如下图所示。



mnist.train.labels是一个55000*10的二维数组，如下图所示。它表示55000个数据点，第一个数据y表示5，第二个数据y表示0，第三个数据y表示4，第四个数据y表示1。



知道了MNIST数据集的组成，以及x和y具体的含义，我们就开始编写Keras吧！

二.Keras实现MNIST分类

本文通过Keras搭建一个分类神经网络，再训练MNIST数据集。其中X表示图片，28*28，y对应的是图像的标签。

第一步，导入扩展包。

```
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop
```

第二步，载入MNIST数据及预处理。

- X_train.reshape(X_train.shape[0], -1) / 255
  将每个像素点进行标准化处理，从0-255转换成0-1的范围。
- np_utils.to_categorical(y_train, nb_classes=10)
  调用up_utils将类标转换成10个长度的值，如果数字是3，则会在对应的地方标记为1，其他地方标记为0，即{0,0,0,1,0,0,0,0,0,0}。

由于MNIST数据集是Keras或TensorFlow的示例数据，所以我们只需要下面一行代码，即可实现数据集的读取工作。如果数据集不存在它会在线下载，如果数据集已经被下载，它会被直接调用。

```
# 下载MNIST数据
# X shape(60000, 28*28) y shape(10000, )
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255  # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255     # normalize

# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)
```

第三步，创建神经网络层。

前面介绍创建神经网络层的方法是定义之后，利用add()添加神经层。

- model = Sequential()
- model.add(Dense(output_dim=1, input_dim=1))

而这里采用另一种方法，在Sequential()定义的时候通过列表添加神经层。同时需要注意，这里增加了神经网络激励函数并调用RMSprop加速神经网络。

- from keras.layers import Dense, Activation
- from keras.optimizers import RMSprop

该神经网络层为：

- 第一层为Dense(32, input_dim=784)，它将传入的784转换成32个输出
- 该数据加载一个激励函数Activation('relu')，并转换成非线性化数据
- 第二层为Dense(10)，它输出为10个单位。同时Keras定义神经层会默认其输入为上一层的输出，即32（省略）
- 接着加载一个激励函数Activation('softmax')，用于分类

```
# Another way to build your neural net
model = Sequential([
        Dense(32, input_dim=784),  # 输入值784(28*28) => 输出值32
        Activation('relu'),        # 激励函数 转换成非线性数据
        Dense(10),                 # 输出为10个单位的结果
        Activation('softmax')      # 激励函数 调用softmax进行分类
        ])

# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0) #学习率lr
```

```
# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                    # 加速神经网络
        loss = 'categorical_crossentropy',    # 损失函数
        metrics = ['accuracy'],                 # 计算误差或准确率
        )
```

第四步，神经网络训练及预测。

```
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)     # 训练次数及每批
print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)

print("loss:", loss)
print("accuracy:", accuracy)
```

完整代码：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 14 16:43:21 2020
@author: Eastmount CSDN YXZ
O(∩_∩)O Wuhan Fighting!!!
"""
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop

#--------------------------载入数据及预处理--------------------------
# 下载MNIST数据
# X shape(60000, 28*28) y shape(10000, )
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255  # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255     # normalize
```

```
# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)

#--------------------------创建神经网络层--------------------------
# Another way to build your neural net
model = Sequential([
        Dense(32, input_dim=784),  # 输入值784(28*28) => 输出值32
        Activation('relu'),        # 激励函数 转换成非线性数据
        Dense(10),                 # 输出为10个单位的结果
        Activation('softmax')      # 激励函数 调用softmax进行分类
        ])

# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0) #学习率lr

# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                  # 加速神经网络
        loss = 'categorical_crossentropy',    # 损失函数
        metrics = ['accuracy'],                # 计算误差或准确率
        )

#----------------------------训练及预测-----------------------------
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)     # 训练次数及每批i
print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)

print("loss:", loss)
print("accuracy:", accuracy)
```

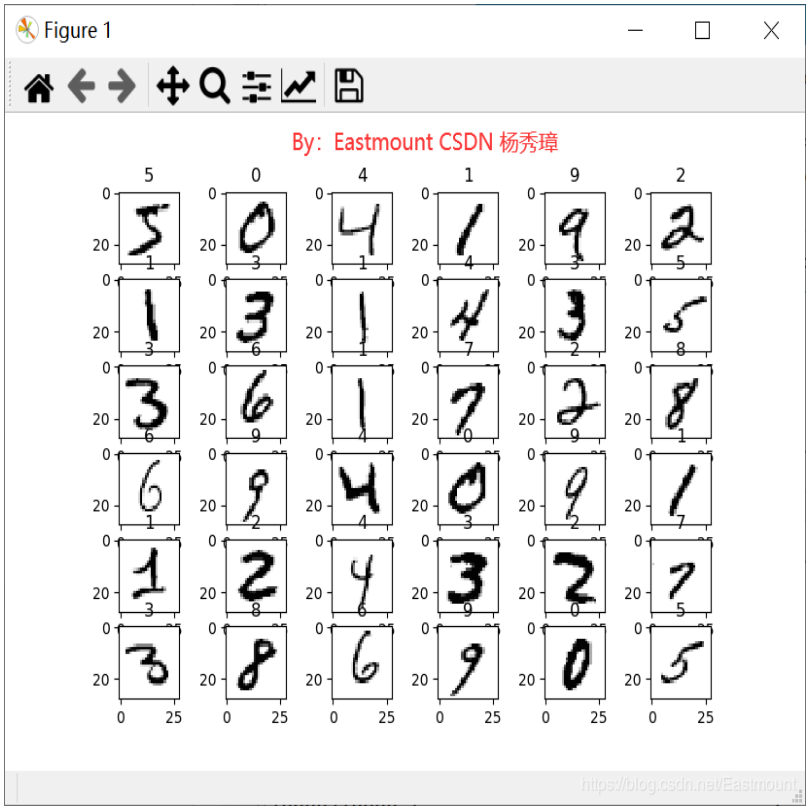
运行代码，首先会下载MNIT数据集。

```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 18s 2us/step
```

接着输出两次训练的结果，可以看到误差不断减小、正确率不断增大。最终测试输出的误差loss为“0.185575”，正确率为“0.94690”。

```
Epoch 1/2
60000/60000 [==============================] - 4s 74us/step - loss: 0.3599 - accuracy: 0.9002
Epoch 2/2
60000/60000 [==============================] - 4s 69us/step - loss: 0.2071 - accuracy: 0.9413
Testing
10000/10000 [==============================] - 0s 38us/step
loss: 0.18557512020245195
accuracy: 0.9469000101089478
```


如果读者想更直观地查看我们数字分类的图形，可以定义函数并显示。



此时的完整代码如下所示：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 14 16:43:21 2020
@author: Eastmount CSDN YXZ
O(n_n)O Wuhan Fighting!!!
"""
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import RMSprop
import matplotlib.pyplot as plt
from PIL import Image

#----------载入数据及预处理----------
#  下载MNIST数据
# X_shape(60000, 28*28) y_shape(10000, )
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#------------------------------显示图片------------------------------
def show_mnist(train_image, train_labels):
    n = 6
    m = 6
    fig = plt.figure()
    for i in range(n):
        for j in range(m):
            plt.subplot(n,m,i*n+j+1)
            index = i * n + j #当前图片的标号
            img_array = train_image[index]
            img = Image.fromarray(img_array)
            plt.title(train_labels[index])
            plt.imshow(img, cmap='Greys')
    plt.show()

show_mnist(X_train, y_train)

# 数据预处理
X_train = X_train.reshape(X_train.shape[0], -1) / 255  # normalize
X_test = X_test.reshape(X_test.shape[0], -1) / 255     # normalize

# 将类向量转化为类矩阵  数字 5 转换为 0 0 0 0 0 1 0 0 0 0 矩阵
y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np_utils.to_categorical(y_test, num_classes=10)

#--------------------------创建神经网络层--------------------------
# Another way to build your neural net
model = Sequential([
        Dense(32, input_dim=784),  # 输入值784(28*28) => 输出值32
        Activation('relu'),        # 激励函数 转换成非线性数据
        Dense(10),                 # 输出为10个单位的结果
        Activation('softmax')      # 激励函数 调用softmax进行分类
        ])

# Another way to define your optimizer
rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0) #学习率lr

# We add metrics to get more results you want to see
# 激活神经网络
model.compile(
        optimizer = rmsprop,                # 加速神经网络
        loss = 'categorical_crossentropy',  # 损失函数
        metrics = ['accuracy'],              # 计算误差或准确率
        )
```

```
#----------------------------训练及预测----------------------------
print("Training")
model.fit(X_train, y_train, nb_epoch=2, batch_size=32)     # 训练次数及每批
print("Testing")
loss, accuracy = model.evaluate(X_test, y_test)

print("loss:", loss)
print("accuracy:", accuracy)
```

三.总结

写到这里，这篇文章就结束了。本文主要通过Keras实现了一个分类学习的案例，并详细介绍了MNIST手写体识别数据集。最后，希望这篇基础性文章对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作为人工智能的菜鸟，我希望自己能不断进步并深入，后续将它应用于图像识别、网络安全、对抗样本等领域，指导大家撰写简单的学术论文，一起加油！武汉必胜，中国必胜。

(By:Eastmount 2020-02-18 晚上9点夜于贵阳 http://blog.csdn.net/eastmount/)

作者theano人工智能系列：
[Python人工智能] 一.神经网络入门及theano基础代码讲解
[Python人工智能] 二.theano实现回归神经网络分析
[Python人工智能] 三.theano实现分类神经网络及机器学习基础
[Python人工智能] 四.神经网络和深度学习入门知识
[Python人工智能] 五.theano实现神经网络正规化Regularization处理
[Python人工智能] 六.神经网络的评价指标、特征标准化和特征选择
[Python人工智能] 七.加速神经网络、激励函数和过拟合

参考文献：

[1] 神经网络和机器学习基础入门分享 - 作者的文章
[2] 斯坦福机器学习视频NG教授： https://class.coursera.org/ml/class/index
[3] 书籍《游戏开发中的人工智能》、《游戏编程中的人工智能技术》
[4] 网易云莫烦老师视频（强推 我付费支持老师一波）
[5] 神经网络激励函数 - deeplearning
[6] 机器学习实战—MNIST手写体数字识别 - RunningSucks
[7] https://github.com/siucaan/CNN_MNIST
[8] https://study.163.com/course/courseLearn.htm?courseId=1003340023