# 人工智能学习笔记四——手写数字识别

本文将用卷积神经网络模型，对手写数字集 minist 进行分类识别，用的框架是 keras

MNIST 是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所发起整理，一共统计了来自 250 个不同的人手写数字图片，其中 50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。

训练集一共包含了 60,000 张图像和标签，而测试集一共包含了 10,000 张图像和标签。测试集中前 5000 个来自最初 NIST 项目的训练集.，后 5000 个来自最初 NIST 项目的测试集。前 5000 个比后 5000 个要规整，这是因为前 5000 个数据来自于美国人口普查局的员工，而后 5000 个来自于大学生

该数据集自 1998 年起，被广泛地应用于机器学习和深度学习领域，用来测试算法的效果，例如线性分类器（Linear Classifiers）、K-近邻算法（K-Nearest Neighbors）、支持向量机（SVMs）、神经网络（Neural Nets）、卷积神经网络（Convolutional nets）等等。

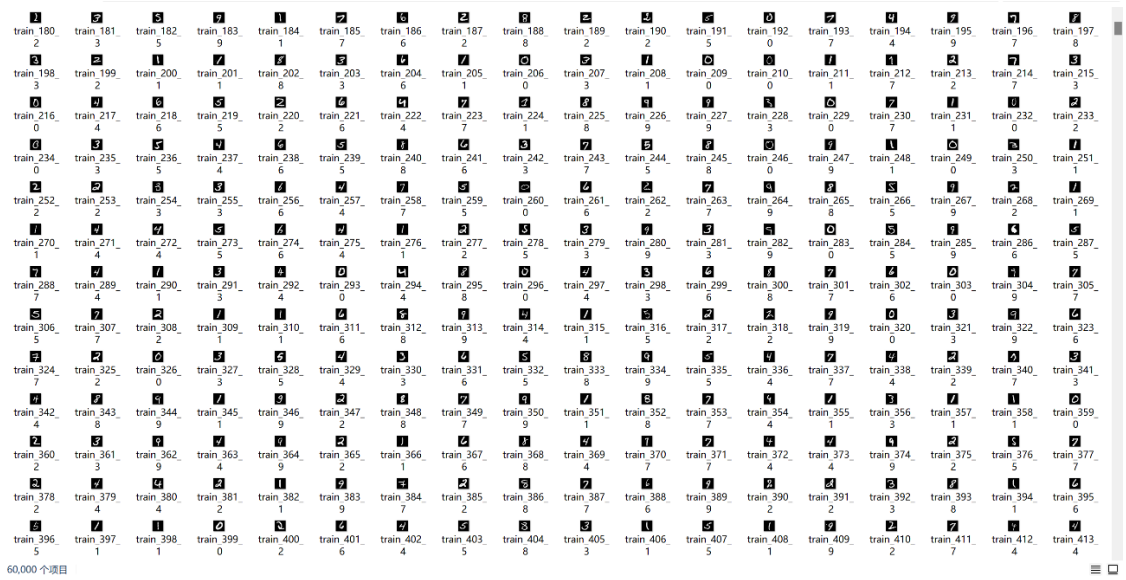


图 1（minist 部分手写数据集)

而 Keras 是一个由 Python 编写的开源人工神经网络库，可以作为 Tensorflow、Microsoft-CNTK 和 Theano 的高阶应用程序接口，进行深度学习模型的设计、调试、评估、应用和可视化 。

Keras 在代码结构上由面向对象方法编写，完全模块化并具有可扩展性，其运行机制和说明文档有将用户体验和使用难度纳入考虑，并试图简化复杂算法的实现难度。Keras 支持现代人工智能领域的主流算法，包括前馈结构和递归结构的神经网络，也可以通过封装参与构建统计学习模型。在硬件和开发环境方面，Keras 支持多操作系统下的多 GPU 并行计算，可以根据后台设置转化为 Tensorflow、Microsoft-CNTK 等系统下的组件。因而本文用 keras 做为框架。

由于手写数字的输入集的长宽都是 28 像素，色彩空间是黑白的，所以不需要太过于复杂的结构。我将输入数据先进行两次卷积操作，卷积核大小为 3×3，

再进行一次池化操作，然后接着进行两次卷积操作，再进行一次池化操作，随后将数据展平，构造 128 维度的全连接层，最后输出 10 维的数组。在整个网络中，除了最后一次外所有层的激活函数都是 relu 函数，最后一层的激活函数使用的是 softmax 函数，损失函数用的是 crossentropy 函数。如图 2 所示。

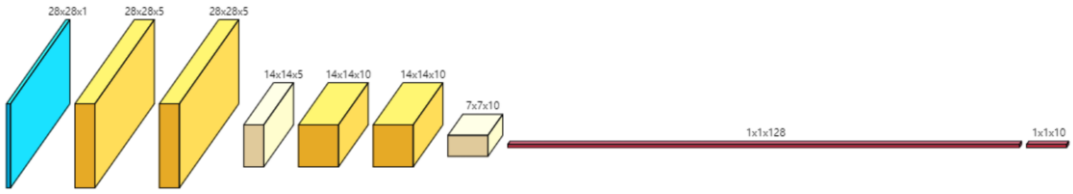


图 2 卷积神经网络示意图

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 28, 28, 5) | 50 |
| conv2d_1 (Conv2D) | (None, 28, 28, 5) | 230 |
| max_pooling2d (MaxPooling2D) | (None, 14, 14, 5) | 0 |
| conv2d_2 (Conv2D) | (None, 14, 14, 10) | 460 |
| conv2d_3 (Conv2D) | (None, 14, 14, 10) | 910 |
| max_pooling2d_1 (MaxPooling2D) | (None, 7, 7, 10) | 0 |
| flatten (Flatten) | (None, 490) | 0 |
| dense (Dense) | (None, 128) | 62848 |
| dense_1 (Dense) | (None, 10) | 1290 |

Total params: 65,788
Trainable params: 65,788
Non-trainable params: 0

图 3 卷积神经网络示意图

网络构造的代码如下：

```
#构建卷积神经网络
def create_model():
    model = keras.Sequential()
    model.add(layers.Conv2D(5, (3, 3),activation='relu',input_shape=(28,28,1),p
adding = 'same'))
    model.add(layers.Conv2D(5, (3, 3), activation='relu', padding = 'same'))
    model.add(layers.MaxPooling2D(pool_size = (2,2)))
    model.add(layers.Conv2D(10, (3, 3), activation='relu', padding = 'same'))
    model.add(layers.Conv2D(10, (3, 3), activation='relu', padding = 'same'))
    model.add(layers.MaxPooling2D(pool_size = (2,2)))
    model.add(layers.Flatten())#将数据压缩成一维数组
    model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
return model
```

经过 25 个 epoch 的训练，最后网络的损失值降到了 0.0307，准确率达到了 0.99。

准确率的训练变化如图 4 所示，Train 表示训练集的准确率变化，Test 表示测试集的准确率变化。

损失值的训练变化如图 5 所示，Train 表示训练集的损失值变化，Test 表示测试集的损失值变化。

总训练过程的变化如图 6 所示。

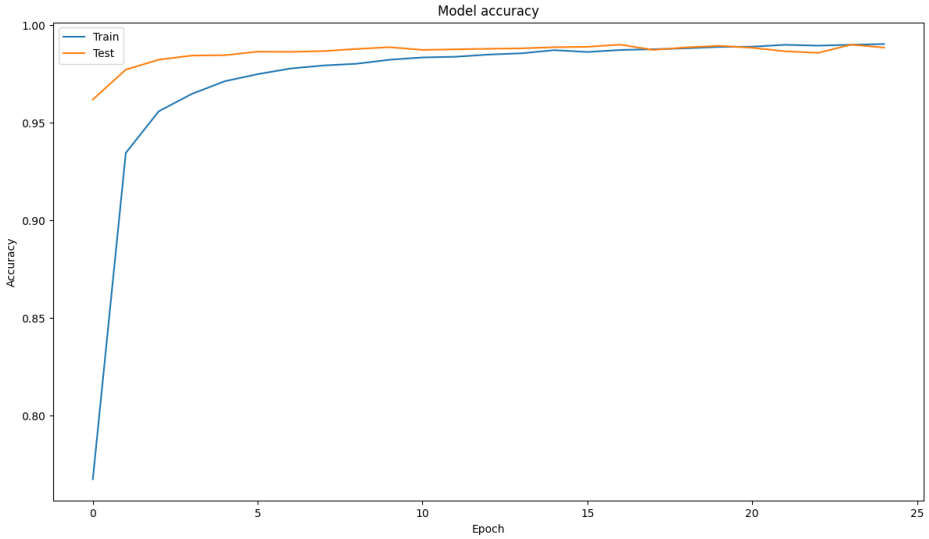


图 4 模型准确率变化曲线

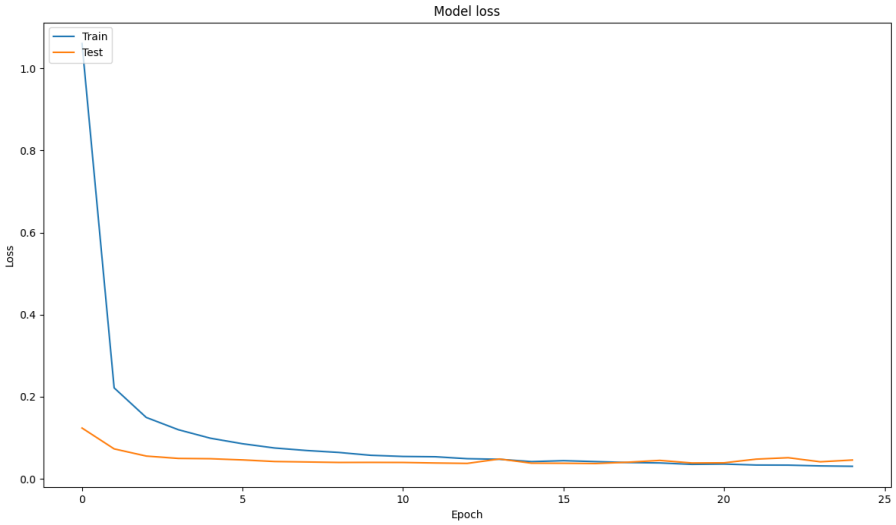


图 5 模型损失值变化曲线

```
Epoch 1/25
1000/1000 - 14s - loss: 1.0605 - accuracy: 0.7674 - val_loss: 0.1239 - val_accuracy: 0.9619 - 14s/epoch - 14ms/step
Epoch 2/25
1000/1000 - 16s - loss: 0.2217 - accuracy: 0.9345 - val_loss: 0.0732 - val_accuracy: 0.9772 - 16s/epoch - 16ms/step
Epoch 3/25
1000/1000 - 16s - loss: 0.1497 - accuracy: 0.9559 - val_loss: 0.0557 - val_accuracy: 0.9823 - 16s/epoch - 16ms/step
Epoch 4/25
1000/1000 - 16s - loss: 0.1198 - accuracy: 0.9647 - val_loss: 0.0499 - val_accuracy: 0.9844 - 16s/epoch - 16ms/step
Epoch 5/25
1000/1000 - 16s - loss: 0.0991 - accuracy: 0.9712 - val_loss: 0.0492 - val_accuracy: 0.9846 - 16s/epoch - 16ms/step
Epoch 6/25
1000/1000 - 16s - loss: 0.0856 - accuracy: 0.9749 - val_loss: 0.0463 - val_accuracy: 0.9864 - 16s/epoch - 16ms/step
Epoch 7/25
1000/1000 - 16s - loss: 0.0753 - accuracy: 0.9778 - val_loss: 0.0423 - val_accuracy: 0.9863 - 16s/epoch - 16ms/step
Epoch 8/25
1000/1000 - 16s - loss: 0.0692 - accuracy: 0.9793 - val_loss: 0.0413 - val_accuracy: 0.9867 - 16s/epoch - 16ms/step
Epoch 9/25
1000/1000 - 16s - loss: 0.0645 - accuracy: 0.9802 - val_loss: 0.0401 - val_accuracy: 0.9878 - 16s/epoch - 16ms/step
Epoch 10/25
1000/1000 - 16s - loss: 0.0575 - accuracy: 0.9823 - val_loss: 0.0403 - val_accuracy: 0.9887 - 16s/epoch - 16ms/step
Epoch 11/25
1000/1000 - 16s - loss: 0.0548 - accuracy: 0.9834 - val_loss: 0.0401 - val_accuracy: 0.9873 - 16s/epoch - 16ms/step
Epoch 12/25
1000/1000 - 16s - loss: 0.0540 - accuracy: 0.9838 - val_loss: 0.0388 - val_accuracy: 0.9876 - 16s/epoch - 16ms/step
Epoch 13/25
1000/1000 - 16s - loss: 0.0492 - accuracy: 0.9849 - val_loss: 0.0378 - val_accuracy: 0.9879 - 16s/epoch - 16ms/step
Epoch 14/25
1000/1000 - 16s - loss: 0.0478 - accuracy: 0.9856 - val_loss: 0.0485 - val_accuracy: 0.9881 - 16s/epoch - 16ms/step
Epoch 15/25
1000/1000 - 16s - loss: 0.0421 - accuracy: 0.9872 - val_loss: 0.0383 - val_accuracy: 0.9887 - 16s/epoch - 16ms/step
Epoch 16/25
1000/1000 - 16s - loss: 0.0444 - accuracy: 0.9862 - val_loss: 0.0381 - val_accuracy: 0.9889 - 16s/epoch - 16ms/step
Epoch 17/25
1000/1000 - 16s - loss: 0.0422 - accuracy: 0.9873 - val_loss: 0.0374 - val_accuracy: 0.9900 - 16s/epoch - 16ms/step
Epoch 18/25
1000/1000 - 16s - loss: 0.0399 - accuracy: 0.9876 - val_loss: 0.0407 - val_accuracy: 0.9873 - 16s/epoch - 16ms/step
Epoch 19/25
1000/1000 - 16s - loss: 0.0389 - accuracy: 0.9881 - val_loss: 0.0450 - val_accuracy: 0.9886 - 16s/epoch - 16ms/step
Epoch 20/25
1000/1000 - 16s - loss: 0.0354 - accuracy: 0.9888 - val_loss: 0.0388 - val_accuracy: 0.9894 - 16s/epoch - 16ms/step
Epoch 21/25
1000/1000 - 16s - loss: 0.0362 - accuracy: 0.9890 - val_loss: 0.0392 - val_accuracy: 0.9884 - 16s/epoch - 16ms/step
Epoch 22/25
1000/1000 - 16s - loss: 0.0338 - accuracy: 0.9899 - val_loss: 0.0481 - val_accuracy: 0.9866 - 16s/epoch - 16ms/step
Epoch 23/25
1000/1000 - 16s - loss: 0.0336 - accuracy: 0.9895 - val_loss: 0.0516 - val_accuracy: 0.9858 - 16s/epoch - 16ms/step
Epoch 24/25
1000/1000 - 16s - loss: 0.0317 - accuracy: 0.9900 - val_loss: 0.0417 - val_accuracy: 0.9900 - 16s/epoch - 16ms/step
Epoch 25/25
1000/1000 - 16s - loss: 0.0307 - accuracy: 0.9903 - val_loss: 0.0460 - val_accuracy: 0.9885 - 16s/epoch - 16ms/step
```

图 6 训练过程图

完整代码为：

```python
#coding:gbk
from PIL import Image
import numpy as np
from keras import layers
import keras
import matplotlib.pyplot as plt
import glob
np.set_printoptions(threshold=np.inf)
wid = 28#定义图片的宽
hei = 28#定义图片的长
def process(preimg):#读取图片，将它转化成 np.array 的格式
    imge = Image.open(preimg,'r')
    imge = imge.convert('L')
    imge = imge.resize((wid,hei))
```

```
    return (np.asarray(imge))
trainset = []
trainexpe = []
testset = []
testexpe = []
train_src = glob.glob("train_images//*.jpg")#训练数据
test_src = glob.glob("test_images//*.jpg")#测试数据
for data in train_src:#数据填充到数组中
    trainset.append(process(data))
    datatem = [0] * 10
    datatem[int(data [-5])] = 1
    trainexpe.append(datatem)
for data in test_src:
    testset.append(process(data))
    datatem = [0] * 10
    datatem[int(data [-5])] = 1
    testexpe.append(datatem)
trainset = np.array(trainset).reshape((-1,wid,hei,1))
trainexpe = np.array(trainexpe)
testset = np.array(testset).reshape((-1,wid,hei,1))
testexpe = np.array(testexpe)
def create_model():
    model = keras.Sequential()
    model.add(layers.Conv2D(5, (3, 3), activation='relu', input_shape=(wid,hei,1)
,padding = 'same'))
    '''''
    filters 要去训练多少个卷积核
    kernel_size: 卷积核大小
    activation: 非线性化所需要去使用的激活函数
    input_shape:输入数据的形状
    padding:same 表示填充一圈，valid 表示不填充
    strides 表示滑动的步长
    '''
    model.add(layers.Conv2D(5, (3, 3), activation='relu', padding = 'same'))#第二
层添加的时候就不需要 input_shape 这个参数了，因为默认是去根据上一层输出的形状进行计
算。
    model.add(layers.MaxPooling2D(pool_size = (2,2)))#默认 maxpooling 大小为（2，
2）
    model.add(layers.Conv2D(10, (3, 3), activation='relu', padding = 'same'))
    model.add(layers.Conv2D(10, (3, 3), activation='relu', padding = 'same'))
    model.add(layers.MaxPooling2D(pool_size = (2,2)))
    #model.summary()#显示 model 信息
    model.add(layers.Flatten())#将数据压缩成一维数组
    model.add(layers.Dense(128, activation='relu'))
```

```
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(10, activation='softmax'))
    return model

model = create_model()#load_model("test.h5")
model.compile(optimizer="adam",loss= "categorical_crossentropy",metrics=['accuracy'])
history = model.fit(trainset,trainexpe,batch_size=60,epochs=25,verbose=2, validation_data=(testset,testexpe))

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# 绘制训练 & 验证的损失值
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
model.save('test.h5')
```

本文地址：[TLearning (caodong0225.github.io)]