三峡大学

计 算 机 与 信 息 学 院

《人工智能》课 程 作 业

2024 年 秋 季 学 期

课程类型：	专业核心课
学　　号：	202210120518
姓　　名：	胡国昌
专　　业：	计算机科学与技术
授课教师：	臧兆祥

完成日期：2024 年 10 月 20 日

手写体数字识别

一、 案例内容与要求

1.1. 实验内容与任务

MNIST 数据集是一个手写体数字的图像数据集，训练集包括 60000 张图片，测试集包括 10000 张图片，每张图片是一个 8 位的灰度图片，尺寸为 28×28,训练集的前 20 张图片如图 1.1 所示。现要求训练一个卷积神经网络，用于识别数字图片。

1.2. 实验过程及要求

1. 实验环境要求：Windows/Linux 操作系统，Python 编译环境，numpy、keras、matplotlib 等程序库。
2. 学习理解神经网络、卷积网络层、图像处理等知识。
3. 下载数据集，构建卷积神经网络，进行网络训练与评估。
4. 调整网络超参数，记录网络训练的过程。
5. 撰写实验报告。

二、 原理论述及解决方法

2.1. 原理概述

神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的激活函数，每两个节点间的连接都代表一个对于通过该连接信号的加权值，网络的输出是连接方式、权重值和激活函数的综合作用。通过调整神经网络的结构、规模、参数，它可以逼近任何函数，因此神经网络目前成为机器学习的重要工具。图像处理中可以通过卷积计算获得图像特征，卷积神经网络可以进行图像特征提取，以实现图像分类、目标检测等任务。2012 年，在大规模机器视觉识别竞赛(ILSVRC)上，卷积神经网络 AlexNet 超出其他学习方法，取得了最好结果。从此，卷积神经网络技术在图像处理领域得到广泛的应用。

2.2. 解决方法

2.2.1. 神经网络与层

各种信息处理过程可以看作是一个函数处理 $y = f(x)$，然而 f 的形式往往是未知并且复杂的。神经网络使用简单的线性处理或非线性处理进行连续复合的方式，来逼近或者模拟 f。用 x^0 表示输入数据，用 x^{i-1}，f^i，x^i 表示第 i 次复合处理的输入、简单处理函数和输出，则神经网络处理过程为

$$x^0 = x$$

$$x^1 = f^1(x^0)$$
$$\cdots$$
$$x^k = f^k(x^{k-1})$$
$$y = f^{k+1}(x^k)$$

其中，$x^0 = x$是输入层，$x^i = f^i(x^{i-1})$，$i = 1,\ldots,k$是k个隐含层，$y = f^{k+1}(x^k)$称为输出层。在神经网络中，f通常定义为x的线性函数或非线性函数的组合，例如常见的全连接层的定义为

$$f(x) = \sigma(wx + b)$$

这些简单函数的复合可以逼近复杂函数，而每个网络层的w,b等参数一起构成了神经网络的参数，x的每个元素是一个神经元。神经网络学习的任务是通过大量的训练数据来找到各网络层的参数。

2.2.2. 激活函数

网络层要加入非线性成分，因为多个线性处理的复合还是线性的，只有线性处理的神经网络的函数模拟能力有限。可以如同公式 1.2 一样，在线性处理后加入一个非线性的成分，称为激活函数。当然也可以把激活函数定义为一个单独的层。常用的激活函数有 sigmoid、tanh、relu 函数等。因为神经网络是使用梯度下降法完成参数训练，sigmoid 函数在多数区域梯度为 0，会影响训练的速度，因此常常使用 relu 函数作为激活函数。

$$relu(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

2.2.3. 损失函数与优化计算

给定一批训练数据$x_1, y_1, x_2, y_2, \ldots, x_n, y_n$。为了计算网络参数，定义一个损失函数来衡量网络预测与真实数据之间的差异，通过在训练集上最小化损失函数来获得对参数的估计。假设神经网络用函数$f_w(x)$表示，用神经网络进行回归处理时，常用均方误差(MSE)作为损失函数，

$$loss = \sum_{i=1}^{n} (f_w(x_i) - y_i)^2$$

当用神经网络进行分类处理时，输出一个概率向量，用负的对数似然作为损失函数，也称为交叉熵损失函数，

$$loss = -\sum_{i=1}^{n} y_i \log\big(f_w(x_i)\big)$$

其中y_i用 one-hot 形式表示。

由于神经网络$f(x)$的线性和非线性函数都是基本光滑的，因此损失函数也是光滑的。神经网络中用梯度下降法来最小化损失函数，从而获得最佳的参数。虽然主要应用梯度下降，但在具体的计算中，嵌入了不同的处理手段，目前常用的优化算子有随机梯度下降(SGD)，自适应学习率优化算子 AdaGrad, RMSProp 等。

2.2.4. 图像处理与卷积层

用一个 2 值图片 1(a)为例，其中有一个“7”字。图 1(b)是一个尺寸较小的模板，称为卷积核。将模板在图片 1(a)上滑动，并与覆盖的区域作乘法，

$$F(x,y)=\sum_{i=0}^{a-1}\sum_{j=0}^{b-1}I(x+i,y+i)w(x+i,y+i)$$

其中F是输出特征，I是输入的图片，w是卷积核，a和b是卷积核的尺寸。这个操作称为卷积(跟一般信号处理的严格定义有差别)。卷积的结果按位置列在图 1(c)中。可以发现，在位置（2,3）处出现最大的卷积值 5。这个示意图说明通过卷积操作，能大致判断在图片 1(a)的(2,3)处，可能有一个“7”字存在。卷积核可以有不同的尺寸，我们还可以用两个小一点的模板组合起来处理，如模板 1(d)通过卷积判断有一个水平线，模板 1(e)通过卷积判断有一个垂直线，则很可能有一个“7”存在。组合的方法可以避免使用复杂的卷积核。通常情况下，图像处理中卷积核是方形的。在进行卷积操作时，除了卷积核尺寸可以不同外，在图片上滑动的步幅也可以不同。卷积核的尺寸和卷积的步幅，可以影响到输出的尺寸。

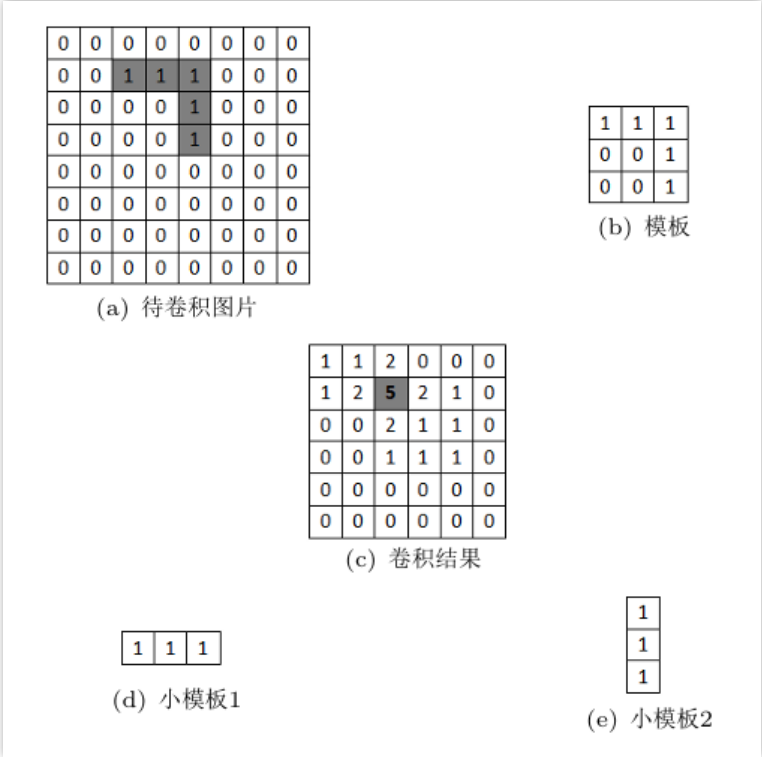

图 1 卷积操作示意图

卷积处理能获取空间关系的特征，从而特别适用于图像处理。神经网络中的卷积层，由一批不同值的卷积核构成，它们进行卷积操作后的输出，相当于提取了一次图片特征。下一个网路层还可以继续是卷积层，提取更高级的特征。

2.2.5. 池化层

神经网络的一个网络层输入的数据量越多，需要配置的参数就越多，计算耗费的时间就越大，训练也就越困难。而图片作为一个空间信息的载体，大部分邻近的信息

是相同的，因此通过采样的方式，扔掉一部分数据，但图片基本特征还能保持。池化层完成的就是这种采样来缩小输入的工作。跟卷积操作类似，用一个池化窗口在图片上滑动，池化结果是窗口内数据取平均值或者最大值，称为平均池化或者最大池化。数据缩小的程度与窗口尺寸和步幅相关。

2.2.6. 丢弃处理 Dropout

神经网络具有大量的参数，有很强的数据拟合能力。但是拟合能力过强，在用来预测时并不一定有好的效果，因为训练数据本身可能会有误差，实际模型也可能并没有那么复杂。避免过拟合的方法之一是训练过程中，随机丢弃一些神经元，使得网络结构变小，从而能部分抑制过拟合。另外，丢弃一些神经元，可以迫使网络其他的神经元能学到一些更一般的特征。

三、 计算结果与讨论

3.1. 计算结果

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	0.4414	0.8619	0.0882	0.9726
2	0.1268	0.9626	0.0662	0.9800
3	0.0900	0.9730	0.0524	0.9843
4	0.0713	0.9781	0.0447	0.9865
5	0.0605	0.9819	0.0402	0.9884
6	0.0527	0.9843	0.0397	0.9893
7	0.0486	0.9853	0.0341	0.9895
8	0.0442	0.9864	0.0330	0.9907
9	0.0402	0.9872	0.0356	0.9902
10	0.0364	0.9886	0.0341	0.9903
11	0.0338	0.9895	0.0317	0.9916
12	0.0320	0.9896	0.0304	0.9912
13	0.0282	0.9911	0.0327	0.9911
14	0.0271	0.9919	0.0308	0.9920
15	0.0265	0.9915	0.0305	0.9920

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	0.4540	0.8569	0.0954	0.9710
2	0.1333	0.9601	0.0620	0.9806
3	0.0970	0.9714	0.0542	0.9843
4	0.0766	0.9770	0.0460	0.9849
5	0.0667	0.9801	0.0407	0.9877
6	0.0568	0.9827	0.0370	0.9892
7	0.0506	0.9847	0.0392	0.9887
8	0.0475	0.9854	0.0367	0.9898
9	0.0417	0.9870	0.0361	0.9897
10	0.0379	0.9889	0.0363	0.9898
11	0.0361	0.9890	0.0328	0.9914
12	0.0344	0.9894	0.0320	0.9911
13	0.0325	0.9901	0.0353	0.9908
14	0.0298	0.9912	0.0328	0.9908
15	0.0290	0.9906	0.0340	0.9910

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	0.4674	0.8519	0.1046	0.9682
2	0.1373	0.9583	0.0622	0.9812
3	0.0954	0.9710	0.0541	0.9837
4	0.0792	0.9763	0.0441	0.9874
5	0.0671	0.9792	0.0410	0.9882
6	0.0573	0.9826	0.0401	0.9884
7	0.0514	0.9847	0.0372	0.9883
8	0.0495	0.9851	0.0373	0.9893
9	0.0432	0.9868	0.0343	0.9901
10	0.0400	0.9879	0.0333	0.9899
11	0.0370	0.9885	0.0324	0.9910
12	0.0370	0.9888	0.0333	0.9911
13	0.0340	0.9894	0.0309	0.9916
14	0.0312	0.9903	0.0348	0.9900
15	0.0294	0.9909	0.0319	0.9904

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	0.4677	0.8555	0.1027	0.9677
2	0.1392	0.9586	0.0666	0.9804
3	0.0968	0.9708	0.0542	0.9841
4	0.0781	0.9765	0.0511	0.9846
5	0.0655	0.9799	0.0439	0.9875
6	0.0590	0.9820	0.0382	0.9889
7	0.0516	0.9847	0.0380	0.9887
8	0.0449	0.9864	0.0380	0.9893
9	0.0414	0.9871	0.0345	0.9908
10	0.0380	0.9880	0.0311	0.9911
11	0.0349	0.9893	0.0349	0.9904
12	0.0324	0.9898	0.0318	0.9909
13	0.0311	0.9903	0.0325	0.9909
14	0.0302	0.9906	0.0302	0.9914
15	0.0276	0.9913	0.0305	0.9914

Epoch	Loss	Accuracy	Val_Loss	Val_Accuracy
1	0.4576	0.8566	0.0977	0.9699
2	0.1342	0.9599	0.0643	0.9806
3	0.0948	0.9716	0.0537	0.9841
4	0.0763	0.9770	0.0465	0.9859
5	0.0650	0.9803	0.0415	0.9880
6	0.0565	0.9829	0.0388	0.9890
7	0.0506	0.9849	0.0371	0.9888
8	0.0465	0.9858	0.0363	0.9898
9	0.0416	0.9870	0.0351	0.9902
10	0.0381	0.9884	0.0337	0.9903
11	0.0355	0.9891	0.0330	0.9911
12	0.0340	0.9894	0.0319	0.9911
13	0.0315	0.9902	0.0329	0.9911
14	0.0296	0.9910	0.0322	0.9911
15	0.0281	0.9911	0.0317	0.9912

在四次实验的平均数据中（图 1-4 为三次实验数据，图 5 为其平均值），模型在 15 个训练轮次内表现出显著的性能提升，具体结果如下：

1. 训练集损失（Loss）：从第 1 轮的 0.4576 逐渐降低至第 15 轮的 0.0281，表明模型在训练过程中逐渐收敛。

2. 训练集准确率（Accuracy）：从第 1 轮的 85.66%提升至第 15 轮的 99.11%，显示了模型在手写数字识别任务上的学习效果。

3. 验证集损失（Val_Loss）：验证集损失从第 1 轮的 0.0977 降低至第 15 轮的 0.0317，表明模型在验证集上的泛化能力有所提高。

4. 验证集准确率（Val_Accuracy）：验证集准确率从第 1 轮的 96.99%提升至第 15 轮的 99.12%。

这些结果表明，随着训练轮次的增加，模型的训练和验证性能均得到了显著的改善。

3.2. 实验讨论

从实验数据可以看出，模型在初期轮次内的损失值和准确率变化较为显著。在前 5 轮训练中，损失值迅速下降，准确率则快速上升。这说明模型在初期阶段能够快速学习到 MNIST 数据集的特征。随着训练的进行，损失值下降和准确率上升的速度逐渐放缓，并在最后几轮趋于稳定，这表明模型逐渐收敛。

在验证集上的表现也类似，验证集的损失和准确率从初始轮次到最后轮次逐步改善，说明模型不仅在训练集上表现良好，而且具备良好的泛化能力。

此外，模型在验证集上的损失值和准确率趋于稳定，表明没有出现明显的过拟合现象。可以认为，此次训练过程中使用的模型结构和参数设置在 MNIST 手写数字识别任务中较为适合，达到了预期的识别效果。

四、 作业总结

在本次实验中，我进行了基于 MNIST 数据集的手写体数字识别实验。本实验的核心任务是利用卷积神经网络（CNN）对手写数字进行分类识别，实验步骤包括数据集下载、模型构建、训练、评估等，环境为 Python 编程，并使用了 NumPy、Keras、Matplotlib 等库。

实验过程与心得：

1. 数据处理与模型搭建：通过对 MNIST 数据集的预处理，将 28×28 的灰度图像输入 CNN 模型。模型的主要结构包含卷积层、池化层以及全连接层，通过多次实验调整了网络的超参数以优化识别效果。

2. 训练与评估：在训练过程中，通过不断优化损失函数，模型的训练集和验证集的准确率均达到了 99%以上，展现了较好的性能。损失值在 15 轮次后收敛，准确率也逐渐趋于稳定，说明模型适合该数据集。

3. 结果分析：实验表明，通过适当的卷积核尺寸、池化方式以及正则化手段，模型能够有效避免过拟合，同时具备良好的泛化能力。模型在训练集和验证集上的表现均较为稳定。

本次实验加深了我对神经网络和卷积神经网络的理解，尤其是在图像处理方面的

应用。通过本实验，我基本了解了图像特征提取、模型调参等技能，也对机器学习的应用有了更深的体会。未来，希望进一步探索深度学习在更复杂数据集上的应用，提高模型的识别精度和泛化能力。

五、 参考文献

[1] 徐义春. 人工智能案例与实验[M]. 2024 年 5 月第 1 版. 清华大学出版社, 2024.

六、 实验代码

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.optimizers import RMSprop
import numpy as np
import matplotlib.pyplot as plt


def show_train_history(item,valid):
  plt.plot(train_history.history[item])
  plt.plot(train_history.history[valid])
  plt.title("Train History")
  plt.ylable(item)
  plt.xlable("Epoch")
  plt.legend(['Train','Valid'],loc = 'best')
  plt.show()

#构建神经网络
model = Sequential()
model.add(Conv2D(filters = 16,     #2 维卷积层，16 个卷积核
        kernel_size = (5,5),#卷积核尺寸
        padding = 'same',
        input_shape = (28,28,1),
        activation = 'relu'
        ))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Conv2D(filters = 36,     #2 维卷积层，16 个卷积核
        kernel_size = (5,5),#卷积核尺寸
        padding = 'same',
        activation = 'relu'
```

```
        ))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))
model.add(Flatten())          #展开成一维向量
model.add(Dense(128,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10,activation = 'softmax'))

(x_Train,y_Train),(x_Test,y_Test) = mnist.load_data()
x_Train = x_Train.reshape(x_Train.shape[0],28,28,1).astype('float32')/256
x_Test = x_Test.reshape(x_Test.shape[0],28,28,1).astype('float32')/256
y_Train = to_categorical(y_Train)
y_Test = to_categorical(y_Test)

model.compile(loss = 'categorical_crossentropy',optimizer = RMSprop(lr
                = 0.001),metrics = ['accuracy'])
train_history = model.fit(x = x_Train,y = y_Train,validation_split = 0.2,epochs
                = 15,batch_size = 300,verbose = 2)
score = model.evaluate(x_Test,y_Test,batch_size = 512)
print(score)

def show_train_history(train,valid):
  plt.plot(train_history.history[train])
  plt.plot(train_history.history[valid])
  plt.title("Train History")
  plt.ylable(train)
  plt.xlable("Epoch")
  plt.legend(['Train','Valid'],loc = 'upper left')
  plt.show()

show_train_history('accuracy','val_accuracy')
show_train_history('loss','val_loss')
```