同济大学

TONGJI UNIVERSITY

期 中 报 告

基于 CNN 的手写数字识别

学　　院：电子与信息工程学院

专　　业：数据科学与大数据技术

学生姓名：胡逸凡

学　　号：2153592

2023 年 11 月

目录

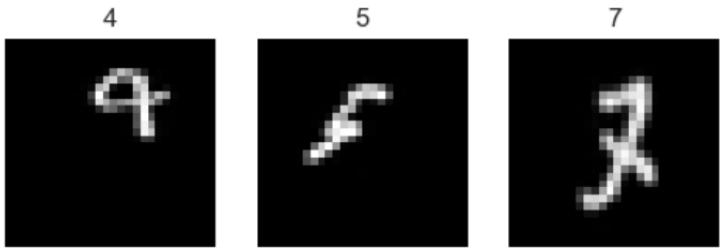

图 1: 易错样例

1 任务描述

1.1 背景

MNIST 数据集是手写数字识别问题中中最经典的数据集之一，许多模型在该数据集上取得了良好效果。但由于其来自美国国家标准与技术研究所, 数字写法不够丰富，不符合中国人的书写习惯，在某些情况下难以识别出正确效果。

以下是我准备的数据集中一些识别出错的样例，如图 1 所示。

1.2 任务要求

扩充已有的 MNIST 数据集，添加中国人手写数字数据集训练，并一个全连接神经网络模型（BP），使之适应中国人的书写习惯，提升识别效果。

1.3 相关工作

CNN（Convolutional Neural Networks, ConvNets, 卷积神经网络）是神经网络的一种，是理解图像内容的最佳学习算法之一，并且在图像分割、分类、检测和检索相关任务中表现出色。

CNN 是一种带有卷积结构的前馈神经网络，卷积结构可以减少深层网络占用的内存量，其中三个关键操作——局部感受野、权值共享、池化层，有效的减少了网络的参数个数，缓解了模型的过拟合问题。

卷积层和池化层一般会取若干个，采用卷积层和池化层交替设置，即一个卷积层连接一个池化层，池化层后再连接一个卷积层，依此类推。由于卷积层中输出特征图的每个神经元与其输入进行局部连接，并通过对应的连

接权值与局部输入进行加权求和再加上偏置值，得到该神经元输入值，该过程等同于卷积过程，CNN 也由此而得名。

CNN 主要应用于图像识别（计算机视觉，CV），应用有：图像分类和检索、目标定位检测、目标分割、人脸识别、骨骼识别和追踪，具体可见 MNIST 手写数据识别、猫狗大战、ImageNet LSVRC 等，还可应用于自然语言处理和语音识别。

本次实验主要将 CNN 用于数字图像识别，通过训练 CNN 模型，从而对手写数字图像进行分类预测。

2 数据集来源

数据集分为两部分。其一为经典的 MNIST 数据集，共 60000 条数据；其二是我自行添加的中国人手写的阿拉伯数字，共 1000 条数据。我首先拍照得到 RGB 图像，再转为 28 像素 ×28 像素的灰度图，从而得到 784 长度列表的一行数据。

3 方法

3.1 数据集处理方法

我们需要对两部分数据集进行处理，其一是经典 MNIST 数据集，其二是补充中国人手写数字数据集。

- 经典 MNIST 数据集处理方法

  直接通过 tensorflow.keras.datasets 导入 mnist 数据集即可。其中共有 60000 条数据，包括 (60000,28,28) 与 (60000,)，我们将其转为 (60000,785) 大小的 ndarray 格式，即可作为训练集与验证集。如图 2 所示。

- 补充数据集处理方法

  得到手写的 png 格式图片后，在本地 Adobe Photoshop CC 2019 中进行切片，得到多张每张只包含一个手写数字的 RGB 色彩格式 png 图片格式的图片。

  之后通过 python 程序将其转为 28 像素 ×28 像素格式的灰度图，之后将 784 个像素点的数据转为 ndarray 格式，并添加 label，得到处理后的补充数据集。

图 2: 经典 MNIST 数据集示例

```
for dirname in os.listdir(name):
    path = os.path.join(name, dirname)
    FileList = []
    for filename in os.listdir(path):
        if filename.endswith(".png"):
            FileList.append(os.path.join(name, dirname, filename))
    for filename in tqdm(FileList, desc=dirname): # 展示进度
            # 转为灰度图
            img = Image.open(filename).convert('L')
            # 改为28像素x28像素
            img = img.resize((28, 28))

            # 转为ndarray
            pixel_values = list(img.getdata())
            pixel_values = [255 - x for x in pixel_values]
            data_image.append(pixel_values)
            # 添加label
            label = dirname
            data_label.append(label)
```

3.2 模型结构设计

采用了 CNN 模型，CNN 技术已经比较成熟，一般的 CNN 主要包括以下结构：

- 输入层 (Input layer)：输入数据；
- 卷积层 (Convolution layer，CONV)：使用卷积核进行特征提取和特征映射；
- 激活层：非线性映射（ReLU）
- 池化层 (Pooling layer，POOL)：进行下采样降维；
- 光栅化 (Rasterization)：展开像素，与全连接层全连接，某些情况下这一层可以省去；
- 全连接层 (Affine layer / Fully Connected layer，FC)：在尾部进行拟合，减少特征信息的损失；
- 激活层：非线性映射（ReLU）
- 输出层 (Output layer)：输出结果。

其中，卷积层、激活层和池化层可叠加重复使用，这是 CNN 的核心结构。

本次实验中，我通过不断 trial and error，并考虑到计算压力，得出一个效果不错的网络结构，共 5 层，具体介绍如下：

```
class DigitRecognizerModel(nn.Module):

    def __init__(self):
        super().__init__()
        # layer 1
        self.conv1 =
            nn.Conv2d(in_channels=1,out_channels=128,kernel_size=5,stride=1,padding=0)
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2))
        self.drop1 = nn.Dropout(p=0.3)

        # layer 2
        self.conv2 =
            nn.Conv2d(in_channels=128,out_channels=256,kernel_size=5,stride=1,padding=0)
```

```
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2))
        self.drop2 = nn.Dropout2d(p=0.4)

        # layer 3
        self.linear1 = nn.Linear(in_features=256*4*4,out_features=64)
        self.drop3 = nn.Dropout(p=0.4)

        # layer 4
        self.linear2 = nn.Linear(in_features=64,out_features=32)
        self.drop4 = nn.Dropout(p=0.4)

        # layer 5
        self.linear3 = nn.Linear(in_features=32,out_features=10)
        self.softmax = nn.Softmax(dim=1)

    # 前向传播
    def forward(self,x):
        out = self.drop1(self.pool1(F.relu(self.conv1(x))))
        out = self.drop2(self.pool2(F.relu(self.conv2(out))))
        out = out.view(-1,256*4*4)
        out = self.drop3(F.relu(self.linear1(out)))
        out = self.drop4(F.relu(self.linear2(out)))
        out = self.linear3(out)
        return out
```

- Layer 1：
  卷积层: Conv2d1，输入通道为 1（单通道灰度图像），输出通道为 128，卷积核大小为 5x5，步幅为 1，填充为 0。
  激活函数: 采用 ReLU 激活函数。
  池化层: MaxPool2d1，2x2 最大池化。
  丢弃层: Dropout1，以概率 0.3 进行丢弃。

- Layer 2：
  卷积层: Conv2d2，输入通道为 128，输出通道为 256，卷积核大小为 5x5，步幅为 1，填充为 0。
  激活函数: 采用 ReLU 激活函数。
  池化层: MaxPool2d2，2x2 最大池化。

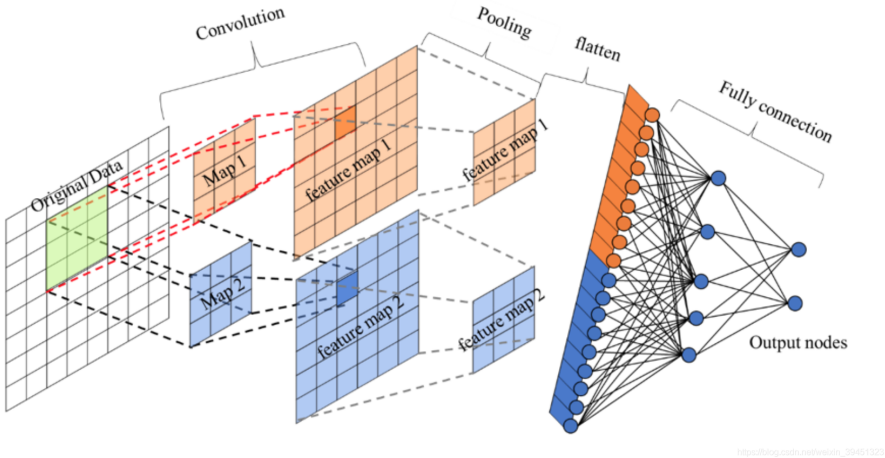

图 3: Model architecture of digit recognition CNN.

二维丢弃层: Dropout2，以概率 0.4 进行丢弃。
展平层: view，将 torch 张量展平，准备输入全连接层。

- Layer 3:
  全连接层: Linear1，输入特征数为 25644，输出特征数为 64。
  激活函数: 采用 ReLU 激活函数。
  丢弃层: Dropout3，以概率 0.4 进行丢弃。

- Layer 4:
  全连接层: Linear2，输入特征数为 64，输出特征数为 32。
  激活函数: 采用 ReLU 激活函数。
  丢弃层: Dropout4，以概率 0.4 进行丢弃。

- Layer 5:
  输出层: Linear，输入特征数为 32，输出特征数为 10（对应 10 个数字类别）。
  激活函数: 采用 Softmax 激活函数，用于多类别分类。

模型结构按照卷积神经网络的典型结构设计，通过卷积、激活、池化、全连接等层来提取并学习图像特征，最终输出每个类别的概率分布，从而对手写图像进行分类预测。

3.3 模型算法

采用了基于随机梯度下降进行反向传播 [Fukushima(1980)]，我们的网络模型的分类定义为 $f(x^(784), y, \theta)$，其中，$x^{(784)}$ 为 n×784 的输入数据，y 为 n×1 的 label，θ 为神经网络需要学习的参数。从而，我们的目的为训练 θ 使得对于手写数字 t 的图像 $figure(t)$，有 $f(figure(t), \theta) = t$

伪代码如 Algorithm1 中所示，在每一个 epoch 的训练中，我们首先从训练集中得到 $x^(784), y$ 数据，之后进行前向传播输入模型，再通过反向传播计算损失函数并更新梯度，**需要注意的是**，我们在每一个 epoch 中进行了梯度清零操作，而没有选择梯度累加等，因为数据集已经足够大，不需要梯度累加进行变相的样本扩大，更重要的是，我不希望让经典 MNIST 数据集中训练的梯度影响扩充数据集训练梯度的计算。

另外我们采用 nn.CrossEntropyLoss() 即交叉熵损失函数作为 loss function，并采用 torch.optim.Adam(model.parameters(), lr=lr) 即 Adam 优化器来调参。

Algorithm 1: The learning algorithm of Hyf's digit recognition

Input : training set(n×785)
Parameters: learning rate η
Output : classification label 0-9

Initialize *learning rate* η as 0.001;
Initialize max_epoch as 10;
for $epoch = 1$ *to* max_epoch **do**
    Train Pattern(model.train());
    for $batch = 1$ *to* n **do**
        Get $x^{(784)}, y$ samples from the training set;
        Forward propagation;
        Calculate loss function : $L(y, p|\theta_t) = -\sum_i y_i log(p_i)$;
        Back propagation and update parameters :
        $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$;
        Zeroing gradients;
    end
end

4 实验

4.1 实验环境

python version：3.10.13
pycharm version：PyCharm 2023.2.4 (Professional Edition)
pytorch version：2.1.0
CUDA version：12.2
显卡: NVIDIA GeForce GTX 1650

4.2 数据集描述

实验分为两步：首先用经典 MNIST 数据集训练模型，测试扩充的中国人手写阿拉伯数字数据集；之后将补充的中国人手写阿拉伯数字数据集添加到训练集中，再次对非训练集中的扩充数据集进行测试。需要注意的是，0-9 总共 10 个数字的数据集并不是数量相同的，书写差异较大的数字在数据集中占比更多，如 2、5、7、9。

总体数据集大小为由经典 MNIST 数据集得到的 60000×785 大小是数据（28×28=784 个像素值，以及一列 label），补充数据集 548×785 大小的数据。

4.3 模型代码片段描述

训练部分。

```
epochs = 10                     # epoch为10
for epoch in range(epochs):     # 每一个epoch
    print('Epoch {epoch+1}')    # prompt输出
    print('-------------------------------')
    train_loop(train_loader, model, loss_func, optimizer) # 训练
    valid_loop(valid_loader, model, loss_func)    # 验证
```

train_loop 与 valid_loop 与伪代码中相同，进行前向传播，计算损失函数，计算梯度，更新参数。

```
def train_loop(dataloader,model,loss_func,optimizer,print_every=100):
    size = len(dataloader.dataset)
    model.train()                   # 训练模式
```

```
4
5       for batch, (x,y) in enumerate(dataloader):
6           x = (x.view(-1,1,28,28)).type(torch.FloatTensor).to(device)
7           y = y.to(device)
8           logits = model(x)
9           loss = loss_func(logits,y)
10          loss.backward()
11          optimizer.step()
12          optimizer.zero_grad()
13          if batch % print_every == 0:
14              loss, current = loss.item(), (batch + 1) * len(x)
15              print(f'loss: {loss:>7f} [{current:>5d}/{size:>5d}]')
```

4.4 模型训练

分别用原始的经典 NMIST 数据集训练，五折交叉验证中损失函数图如下，可以看到在训练集上得到的损失是逐渐收敛的，并且随着分别在数据集的 5 部分上进行训练，可以看到验证得到的损失虽然没有较好的收敛，但是也得到了显著的降低。最后得到模型的整体性能：准确度为 95.25% 是相当不错的，五折交叉验证的损失函数变化图如图 4.

之后用扩充数据集进行训练，五折交叉验证中损失函数图如下。可以看到训练和验证误差与原始数据基本类似，但是到最后一轮的时候训练误差降的更低一些，而且最后的准确度为 98.36%，相比原始数据集的训练精度有了很大的提升。

4.5 实验结果分析

未使用扩充数据集混淆矩阵如图 5：添加扩充数据集混淆矩阵如图 6：由混淆矩阵和损失值即准确率，通过分析，在扩充后对中国人手写数字的分类效果显著提升，由 95.25%，提升到 98.36%。另外，对于书写不规范的数字也有较好的泛化能力。

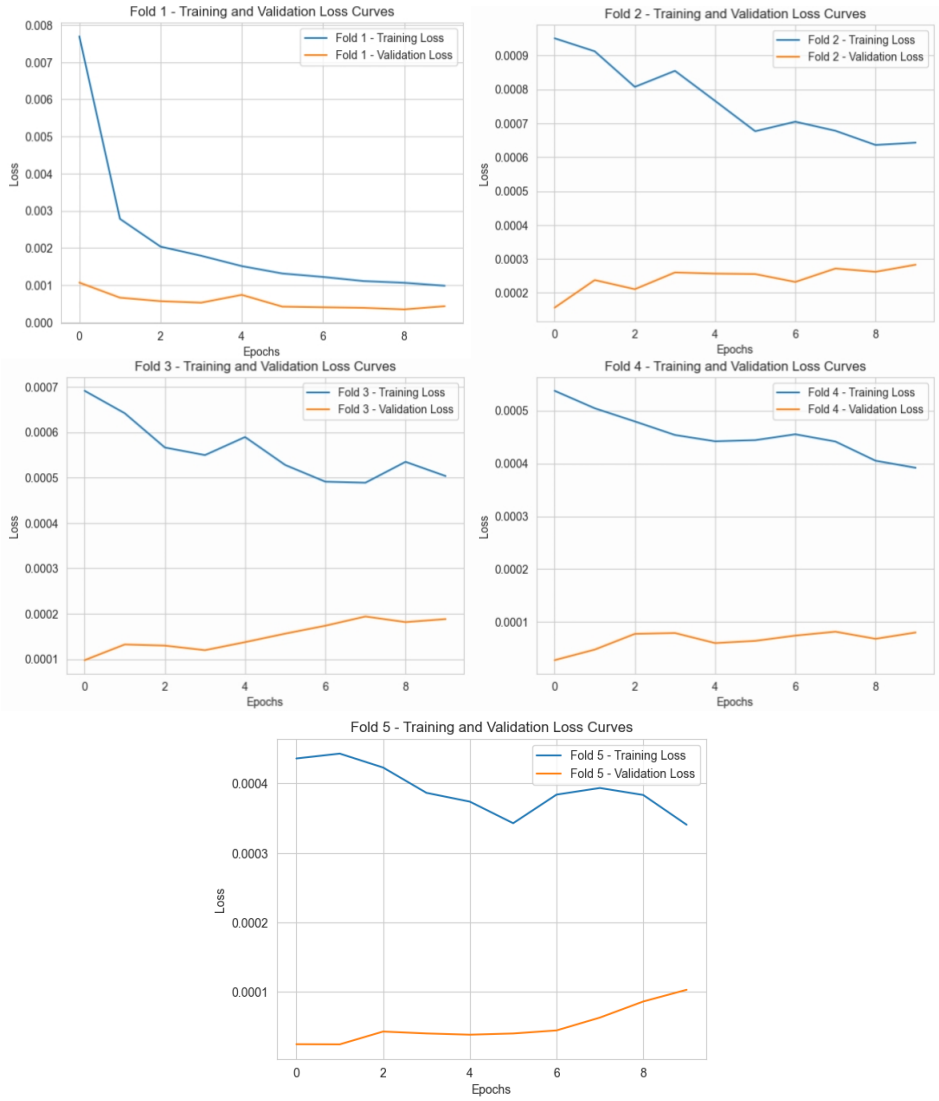
Fold 1 - Training and Validation Loss Curves
Fold 1 - Training Loss
Fold 1 - Validation Loss
Loss
Epochs
Fold 2 - Training and Validation Loss Curves
Fold 2 - Training Loss
Fold 2 - Validation Loss
Loss
Epochs
Fold 3 - Training and Validation Loss Curves
Fold 3 - Training Loss
Fold 3 - Validation Loss
Loss
Epochs
Fold 4 - Training and Validation Loss Curves
Fold 4 - Training Loss
Fold 4 - Validation Loss
Loss
Epochs
Fold 5 - Training and Validation Loss Curves
Fold 5 - Training Loss
Fold 5 - Validation Loss
Loss
Epochs

图 4: 五折交叉验证.

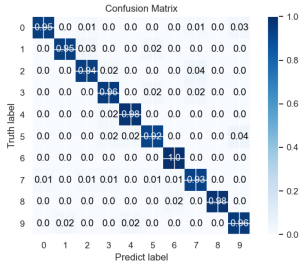

图 5: 未使用扩充数据集混淆矩阵.

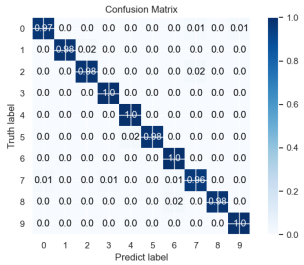

图 6: 使用扩充数据集混淆矩阵.

5 结论和展望

5.1 结论

我的实验首先通过 pytorch 框架构建了含有五层的 CNN 神经网络，采取 sigmoid 函数作为激活函数，并采用随机梯度下降进行反向传播，对 MNIST 原始数据集进行训练，对中国人手写数据集进行预测，准确率约为 95.25% 左右。

通过查看预测错误的样例，我发现数据集对于一些中美特殊的书写习惯泛化能力不强，尤其表现在 2、5、7、9。于是我们通过加入近 1000 张我收集到的认识的朋友等中国人手写数字的照片并转化为数据集，对原有的 MNIST 数据集进行扩充，并重新训练，再对未出现在训练集中的中国人手写数据集进行预测，得到了更小的损失和约 98.36% 的准确率，可以看出有有不小的提升。

之后，通过可视化可以发现，在扩充数据集后，对于一些比较特殊的数字书写也能有较好的泛化能力，数据集扩充和模型训练有较好的效果。

5.2 展望

本次手写数字识别是深度学习最基础的应用之一，深度学习具有更强大的性能和更好的前景。 [Choudhary et al.(2022)] 深度学习是机器学习和人工智能研究的最新趋势之一。它也是当今最流行的科学研究趋势之一。深度学习方法为计算机视觉和机器学习带来了革命性的进步。

如今的深度学习模型有 CNN,RNN,LSTM,Transformer,DRL,GPT 等等，也有在各个领域的各种 sota 模型，深度学习在图像分类与识别，视频分类，序列生成，序列预测，缺陷分类，文本、语音、图像和视频处理，动作识别，歌曲合成等等表现出强大的性能。新的深度学习技术正在不断诞生，超越最先进的机器学习甚至是现有的深度学习技术。随着硬件性能的提升和算法的不断改进，我们可以期待看到更大、更复杂的深度学习模型。新的网络结构和学习算法将进一步提高模型的性能和泛化能力。尽管深度学习（DL）比以往任何时候都更快地推进了世界的发展，但仍有许多方面值得我们去研究。我们仍然无法完全地理解深度学习，我们如何让机器变得更聪明，更接近或比人类更聪明，或者像人类一样学习。DL 一直在解决许多问题，同时将技术应用到方方面面。但是人类仍然面临着许多难题，我们希望深度学习和人工智能将更加致力于改善人类的生活质量，通过开展最困难的科学研究。愿我们的世界变得更加美好!

参考文献

[Choudhary et al.(2022)] Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon J. L. Billinge, Elizabeth Holm, Shyue Ping Ong, and Chris Wolverton. 2022. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials* 8, 1 (apr 2022). https://doi.org/10.1038/s41524-022-00734-6

[Fukushima(1980)] Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* 36 (1980), 193–202. https://api.semanticscholar.org/CorpusID:206775608