

华中科技大学计算机科学与技术学院

《机器学习》 结课报告



专	业	<u>计算机科学与技术</u>
班	级	<u>CS2203</u>
学	号	<u>U202215634</u>
姓	名	<u>黄君翊</u>
成	绩	<u></u>
指导教师		<u>张 腾</u>
时	间	<u>2024 年 5 月 21 日</u>

目录

1	实验要求	1
2	算法设计与实现	2
2.1	算法目标	2
2.2	数据集分析和处理	2
2.3	神经网络层实现	3
2.4	卷积神经网络搭建	6
2.5	模型训练	7
2.6	K 邻近算法	8
3	实验环境与平台	9
4	结果与分析	10
4.1	模型预测结果分析	10
4.2	与现有模型对比	11
4.3	优化方向	11
5	个人体会	12
	参考文献	13

1 实验要求

总体要求：

1. 控制报告页数，不要大段大段贴代码
2. 表格和插图请编号并进行交叉引用，表格使用三线表

2 算法设计与实现

本次实验我选择参考选题四：数字识别，数据集为 MNIST 手写数字数据集，模型使用 CNN 卷积神经网络（基于 numpy 实现）[1]，对 test.csv 预测上传 kaggle，并与 pytorch 现有模型进行对比。

2.1 算法目标

使用 numpy 从零开始搭建 CNN 卷积神经网络，实现其中的全连接层、卷积层、池化层等的正向和反向传播，并以此搭建 Lenet-5 网络进行小批量随机梯度下降（SGD）训练，验证 Lenet-5 在 MNIST 数据集上的效果，另外与 KNN 对比，并在测试集上达到一定准确度，最终解决手写数字识别问题，最后将实现代码上传 github。

2.2 数据集分析和处理

使用 pandas 库读取 train.csv 并分析数据集构成。数据集共 785 个列，第一列为 label，后 784 列为单通道 28*28 的灰度图片，取值 [0, 255) 的整数，使用 matplotlib 读取前 9 张图片，如图2.1

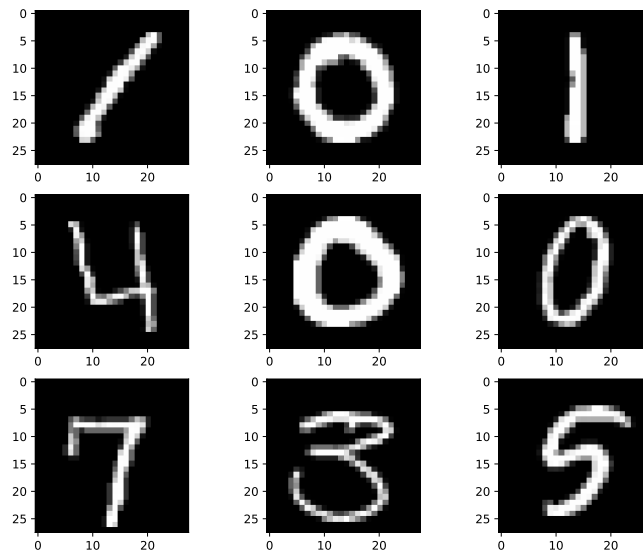


图 2.1: 可视化训练集

此时的数据并不能直接用来训练，模型只接受 numpy 支持的数组 ndarray，首先我们需要将 dataframe 转换为 ndarray，然后将 label 和 feature 分离，取出第一列作为 train_label，剩下的 reshape 到 28*28 的矩阵后作为 train_feature。

之后我们还需要从训练集中划分一部分作为验证集来在训练中实时检测模型的准确率，使用 `sklearn` 包中的 `train_test_split` 来进行数据集的随机划分，我的实现取了 80% 的数据用于训练，剩下的用于验证。

CNN 网络训练需要大量样本，为了加速训练，也保证训练集的随机性，我们需要将数据集划分为小批量 (batch) 进行处理，在这里我使用 `pytorch` 包中的 `DataLoader` 来加载数据，每一次取出 128 个数据作为 (128, 1, 28, 28) (批量大小, 通道, 高, 宽) 的 `tensor` 张量, (按照实验要求) 转换为 `ndarray` 后训练, 数据类型设置为 `np.float64` (后期发现代码中还有一部分是 `np.float32`, 没有完全改过来)。

最后, 得到的训练样本每一个像素取值在 0 到 255 之间, 为了加速模型收敛, 减小数据尺度, 防止梯度爆炸和数据溢出的问题, 将像素值除以 256 进行归一化。

2.3 神经网络层实现

卷积神经网络 (CNN) 是一种深度学习模型 [2], 广泛应用于图像分类、目标检测、图像分割、图像生成、视频分析以及自然语言处理等领域。在这里为了能够搭建一个 CNN 网络, 首先需要实现其基本组件: 卷积层、池化层、全连接层等。

2.3.1 卷积层

卷积层 (Convolution) 是 CNN 的核心组件, 在图像处理和特征提取中起着至关重要的作用。卷积层不一次性处理整个图像, 而是通过一个卷积核逐步扫描图像的局部区域, 自动学习和提取最有用的特征。一张图像共享一个卷积核的参数, 参数共享大大减少了模型的参数数量, 相比全连接层, 是一种稀疏连接的层。

前向传播中, 具体的计算过程, 卷积层包含 `in_channel`, `out_channel`, `padding`, `stride`, `bias` 五个参数, 前两个确定图片输入的通道数和最后输出的通道数, `padding` 决定是否需要对图片四周向外填充数据, `stride` 决定卷积核扫描的步幅, `bias` 决定该卷积层是否需要偏置。设输入数据维度为 $(N, C_{in}, H_{in}, W_{in})$, 其中 N 是批大小 (batch size), C_{in} 是输入通道数, H_{in} 和 W_{in} 分别是输入的高度和宽度, 带偏置。那么卷积核的维度为 $(C_{out}, C_{in}, K_h, K_w)$, 其中 C_{out} 是输出通道数, K_h 和 K_w 分别是卷积核的高度和宽度。卷积操作的输出维度为 $(N, C_{out}, H_{out}, W_{out})$, 其中:

$$H_{out} = \left\lfloor \frac{H_{in} - K_h + 2P}{S} \right\rfloor + 1$$

$$W_{out} = \left\lfloor \frac{W_{in} - K_w + 2P}{S} \right\rfloor + 1$$

P 是 padding 大小, S 是步幅 stride, 常用的情况: 卷积核长宽都为 3, padding 为 1, stride 为 1 时, 保持图片原大小不变。

高维度下卷积公式晦涩难懂, 对于每一个输入通道和输出通道对应一个卷积核, 类似全连接层。设卷积核权重 W , 偏置 b 这一组卷积的前向传播公式如下:

$$Y_{h,w} = \sum_{i=0}^{K_h-1} \sum_{j=0}^{K_w-1} X_{h+i,w+j} \cdot W_{i,j} + b$$

整体的前向传播中高维度卷积公式就类似于多个简化公式的叠加。

对于反向传播，接受的是损失 L 对 Y 的偏导数，利用链式规则可以求出对 X 的偏导数并传递，也可以求出对卷积核和偏置的梯度。值得一提的是，从公式看出，在矩阵运算中，这里的矩阵乘的是卷积核的转置。

$$\begin{aligned} \frac{\partial L}{\partial X_{h,w}} &= \sum_{i=0}^{K_h-1} \sum_{j=0}^{K_w-1} \frac{\partial L}{\partial Y_{h-i,w-j}} \cdot W_{i,j} && \text{for } X \\ \frac{\partial L}{\partial W_{i,j}} &= \sum_{h=0}^{H_{out}-1} \sum_{w=0}^{W_{out}-1} \frac{\partial L}{\partial Y_{h,w}} \cdot X_{h+i,w+j} && \text{for } W \\ \frac{\partial L}{\partial b} &= \sum_{h=0}^{H_{out}-1} \sum_{w=0}^{W_{out}-1} \frac{\partial L}{\partial Y_{h,w}} && \text{for } b \end{aligned}$$

在实际实现上，直接按照公式进行循环会极大损失性能，一直以来很多研究人员提出了众多加速的算法，例如 `im2col` 将卷积转化为全连接计算 [3]，也可以通过 `numpy` 带有的 `stride_trick` 来巧妙地调整视图，利用 `einsum` 来计算等，本实验的代码学习了 `github` 上前人的做法 [4]。

2.3.2 池化层

池化层 (Pooling) 是 CNN 中的另一重要层，用于逐步减小特征图的空间尺寸，从而减少参数和计算量。池化操作通过对输入特征图的某个区域进行降采样，保留最重要的信息，通常包括最大值或平均值，由此分为最大池化层和平均池化层。池化层通常在若干卷积层之后插入，以逐步降低特征图的尺寸和复杂度，并保留有用的信息。。

以最大池化为例，最大池化通过取池化窗口中的最大值来突出显著特征。假设输入特征图的尺寸为 $H \times W$ ，池化窗口的尺寸为 $P_h \times P_w$ ，stride 为 S ，池化后的输出特征图的尺寸为 $H_{out} \times W_{out}$ ，其中： $H_{out} = \lfloor \frac{H-P_h}{S} \rfloor + 1$ ， $W_{out} = \lfloor \frac{W-P_w}{S} \rfloor + 1$ ，则最大池化前向传播公式为

$$y_{i,j} = \max_{(m,n) \in \mathcal{P}_{i,j}} x_{m,n}$$

其中， $\mathcal{P}_{i,j}$ 表示池化窗口在输入特征图中的区域， $x_{m,n}$ 是输入特征图的值。对应的平均池化只需要把取 `max` 换为取 `mean` 就好。

池化层的反向传播过程非常简单，没有任何需要学习的参数，由正向传播的公式，对于一个窗口，只有最大值处对梯度有贡献，所以将梯度传递到最大值处，其他取 0，公式为：

$$\frac{\partial L}{\partial x_{i,j}} = \sum \begin{cases} \frac{\partial L}{\partial y_{m,n}} & \text{if } x_{i,j} \text{ is the max value in the pooling window for } y_{m,n} \\ 0 & \text{otherwise} \end{cases}$$

注意如果 stride 很小，可能有多个窗口对应一个最大值，此时需要累加梯度。对应的平均池化则是将梯度均分累加到窗口所有的梯度上即可。

2.3.3 全连接层

全连接层 (Linear) 也是 CNN 中的一种基本结构。在整个深度学习中，全连接层是神经网络中最经典、最常见的一种层类型之一。相比于卷积层的稀疏连接，全连接层的意义在于实现神经网络中的“全连接”，即每个神经元与前一层中的所有神经元都有连接。这种全连接的结构使得全连接层能够充分地利用前一层的所有特征信息，从而更好地捕捉数据中的复杂关系和模式。在这个实验搭建的 CNN 中，全连接层位于卷积和池化之后，对前面卷积和池化层提取到的特征进行整合和映射以得到最终的预测结果。

全连接层又叫线性层，因为它实际上做矩阵乘法。设输入矩阵为 $X(N, C_{in})$ ，输出矩阵为 $y(N, C_{out})$ ，那么全连接层拥有 (C_{in}, C_{out}) 的权重矩阵 W 和一个偏置 b ，前向传播公式为：

$$y = X \cdot W + b$$

对于反向传播

$$\begin{aligned} \frac{\partial L}{\partial X} &= \frac{\partial L}{\partial y} \cdot W & \text{for } X \\ \frac{\partial L}{\partial W} &= X^T \cdot \frac{\partial L}{\partial y} & \text{for } W \\ \frac{\partial L}{\partial b} &= \sum \frac{\partial L}{\partial y} & \text{for } b \end{aligned}$$

2.3.4 激活函数

激活函数是神经网络中又一重要组成部分。激活函数的主要目的是引入非线性，使得神经网络能够学习并表示复杂的模式。没有激活函数，神经网络将只执行线性变换，

例如在全连接层中，如果在连续的两层间没有激活函数，那么这两层都只能做矩阵意义上的线性变换，失去神经网络的特性。

激活函数常用 Sigmoid, tanh 和 ReLU。因为这个实验我只是用了 ReLU，所以在这一章主要介绍 ReLU。ReLU (Rectified Linear Unit) [5] 当输入值为正时输出该值本身，当输入值为负时输出零。在反向传播中只有输入为正的部分才对梯度有贡献，输入为负的梯度为 0，不列出数学公式。

相比与其他两个函数，ReLU 具有计算简单的优点，而且因为输入为正时保持原值传播，可以避免 Sigmoid 的梯度消失问题，但是也要注意梯度爆炸造成数据溢出。

2.3.5 其他层

1. 展平层

展平层 (Flatten) 用于将多维输入转换为一维向量。CNN 的前几层通常在高维矩阵上卷积和池化，为了连接到后续的全连接层，需要将这些高维张量展平成一维向量。展平层的前向传播和反向传播均只对输入做维度变换。

2. 暂退法

暂退法 (Dropout) [6] 用于随机丢弃一部分神经元及其连接，从而减少模型的复杂性和依赖性。在每次训练迭代中，每个神经元都只有 p 的概率被保留（下一次可能会重新激活）。在神经网络中也作为层存在，并只在训练的时候发挥作用。

2.4 卷积神经网络搭建

现代神经网络大多非常复杂，只是用 numpy 实现不借助 GPU 计算会导致训练非常慢，在这里我搭建一个简单的 Lenet-5 网络，同时使用 pytorch 中预训练的 Resnet50 作为对比，体现残差网络的现代深度神经网络和过去简单网络的不同。

2.4.1 Lenet-5 网络

LeNet-5 是由 Yann LeCun 等人在 1998 年提出的一种经典 CNN 架构 [7]，最初用于手写数字识别任务。LeNet-5 的设计在深度学习的发展历史中具有重要意义，它展示了如何通过卷积层和池化层的组合来有效地提取图像特征，并通过全连接层进行分类。2.2 是一种 Lenet-5 结构。

2.4.2 本次实验使用的网络

这次我搭建的网络基于 Lenet-5，并在其之上调整了一些层，并修改了一些参数，代码采用类似 pytorch 风格搭建网络

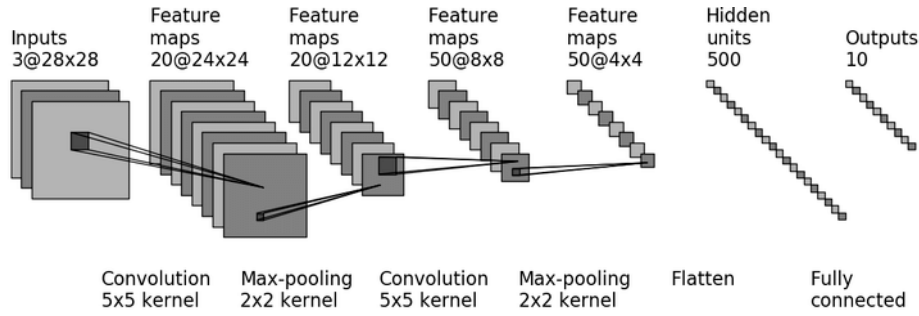


图 2.2: LeNet-5 网络架构

```
1 net = Sequential(  
2     Conv2d(1, 24, 5, 1, 0), ReLU(),  
3     MaxPool2d(2, 2, 0),  
4     Conv2d(24, 60, 5, 1, 0), ReLU(),  
5     MaxPool2d(2, 2, 0),  
6     Flatten(),  
7     Linear(60*4*4, 240), Dropout(0.8), ReLU(),  
8     Linear(240, 84), Dropout(0.8), ReLU(),  
9     Linear(84, 10)  
10 )
```

对比 LeNet 网络，我对层的输入通道数，全连接层的 feature 数进行了一些调整，并从 Sigmoid 函数激活改为 ReLU 函数激活。

第一次训练后发现模型因为被我改复杂了，在 MNIST（本实验数据集）上有过拟合的情况，训练集准确度可达到 99.9%，但验证集不到 98%，所以后期实现并加入了 Dropout 层，保留概率 0.8，训练结果见下一章。

2.5 模型训练

2.5.1 小批量随机梯度下降

随机梯度下降（SGD）是一种常用的训练优化算法，每次更新使用一个样本计算梯度。优点是每次更新计算量小，但梯度估计噪声大，收敛不稳定。通过将训练集升一个维度，一次训练多个样本便演化为小批量随机梯度下降，小批量随机梯度下降的基本步骤如下：

1. **初始化模型参数**：随机初始化模型参数。
2. **划分小批量**：将训练数据集划分为多个小批量（使用 Dataloader），每个小批量包含若干样本，小批量随机从训练集中取出。

3. 循环更新参数：

- (a) 从训练数据集中随机抽取一个小批量样本。
- (b) 计算小批量样本的损失函数对模型参数的梯度。
- (c) 更新模型参数：

$$\theta = \theta - \eta L(\theta)$$

其中， θ 是模型参数， η 是学习率， $L(\theta)$ 是损失函数。

- 4. 重复步骤 3，直到满足停止条件，在这里是达到一定的训练轮数 (epochs)。

2.5.2 学习率衰减

在训练过程中学习率是一个很重要的超参数，太大会导致模型参数震荡不收敛，太小会减慢收敛速度，浪费训练时间，训练过程可以理解为一个逐步向最优点逼近的过程，所以学习率可以先大后小，这就是学习率衰减 (Learning rate decay)，在 pytorch 中一般使用 lr_scheduler 进行管理，在这里直接在 train 函数中手动衰减。

2.5.3 实际模型训练

设置超参数批量大小 128，学习率 0.001，每迭代 10 轮学习率乘以 0.4，训练轮数 25 进行训练。

2.6 K 邻近算法

为了与 CNN 进行对比，我同时也写了一个简单的 K 邻近算法 (KNN)，KNN 的原理非常直观：给定一个未知类别的数据点，它的类别由距离其最近的 K 个已知类别的数据点来决定，如果一个样本在特征空间中与已知类别的样本非常接近，那么它很有可能属于这个类别。

从上面可以知道，KNN 需要一个超参数 K，在确定 K 时需要慎重选择。较小的 K 值会增加模型的复杂度，使其对噪声敏感，而较大的 K 值可能会使得模型过于简单，忽略了数据内部的细节。在这里由于篇幅限制，不展开介绍，仅在最后分析模型结果。

3 实验环境与平台

本次试验的硬件信息如下3.1，在我的个人笔记本上进行训练，表格，包括设备信息，设备配置等。

表 3.1: 硬件信息

项目	信息
设备名称	82RC Legion Y7000P IAH7
CPU	12th Gen Intel i7-12700H (20) @ 4.600GHz
GPU	NVIDIA Corporation GA107BM [GeForce RTX 3050 Ti Mobile]
RAM	16GB
系统类型	x86_64

实验机系统在 linux，表格3.2包括系统信息，版本信息等。

表 3.2: 软件信息

项目	信息
操作系统类型	Arch Linux x86_64
系统版本	6.9.1-arch1-1
cuda 版本	12.4
python 版本	3.11.7
numpy 版本	1.26.3
pytorch 版本	2.20(cuda ver.)
scikit_learn 版本	1.4.2

4 结果与分析

4.1 模型预测结果分析

对于 CNN，训练过程的可视化数据如4.1所示，该网络在 MNIST 训练集下在第一个 epoch 就可以收敛地不错，之后训练集准确度保持在 0.990 左右，验证集准确度在 0.987 左右，基本符合 CNN 训练 MNIST 训练集的情况。

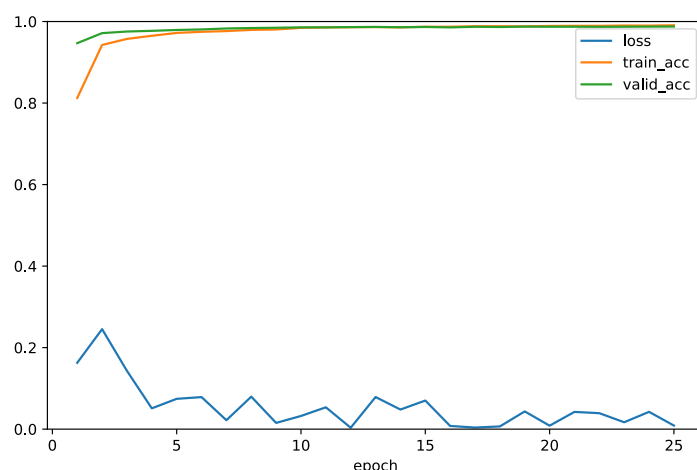


图 4.1: 训练中损失，训练集准确度和验证集准确度

对于 CNN 预测分类的混淆矩阵如4.2,从矩阵中分析,发现 CNN 网络对于‘5’、‘6’、‘9’的识别能力稍差，总体来看识别比较准确。

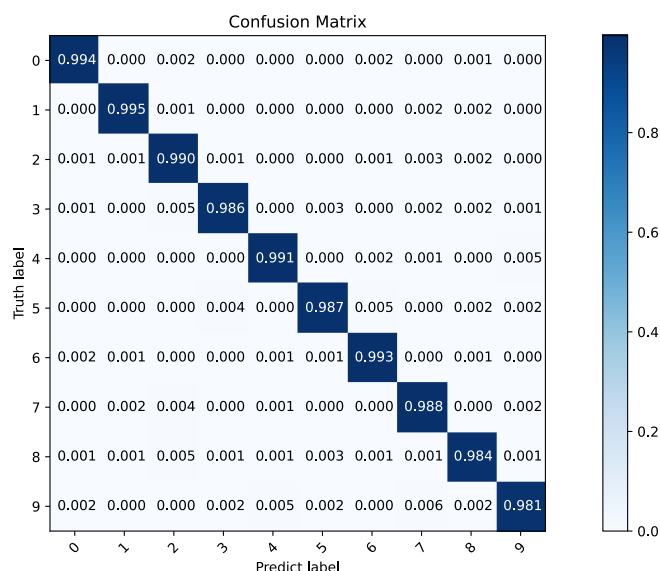


图 4.2: 混淆矩阵

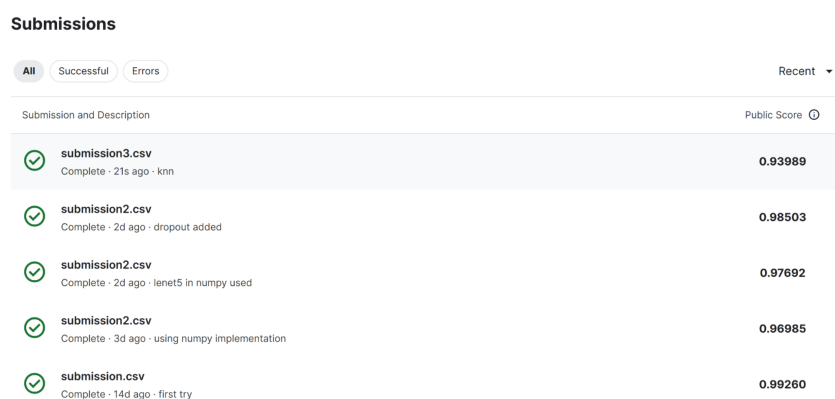
对于 KNN 和 CNN 两个模型，准确度，召回率，F1 值的表格如下4.1，可以看到，在各个评判标准下，KNN 的都略逊色于 CNN 网络，这是因为 CNN 卷积的特性非常适合拿来做图像处理，且模型复杂，表达能力更强。

表 4.1: 准确度，召回率，F1 值

模型	准确度	召回率	F1
CNN	98.864%	98.859%	98.858%
KNN	94.952%	94.925%	94.921%

4.2 与现有模型对比

我在 kaggle 上分别上传了 CNN、KNN 和预训练的 Resnet50 模型训练后的预测，成绩如图4.3所示，submission1, 2, 3 分别对应 ResNet50, CNN, KNN 的预测成绩。



Submissions		Recent ▾
All Successful Errors		
Submission and Description	Public Score ⓘ	
✓ submission3.csv Complete · 21s ago · knn	0.93989	
✓ submission2.csv Complete · 2d ago · dropout added	0.98503	
✓ submission2.csv Complete · 2d ago · lenet5 in numpy used	0.97692	
✓ submission2.csv Complete · 3d ago · using numpy implementation	0.96985	
✓ submission.csv Complete · 14d ago · first try	0.99260	

图 4.3: kaggle 分数

可以看到 Resnet 作为深度卷积神经网络，在没有大量调参的情况下依然比其余两个 numpy 实现的模型效果要好上不少，而 CNN 也是卷积神经网络，结果比 KNN 要好。当然由于 KNN 网络的性质，如果训练得当可以在 kaggle 上做到全对，accuracy 达到 1。

4.3 优化方向

对于 CNN 网络，现有的优化方向有几条，可以改进优化器，使用 Adam 优化器进行梯度下降，也可以加深神经网络，采用使用新的层例如 BN 层 (batch normalization)，使用块 (VGG)，进行预训练等方法。

5 个人体会

通过机器学习这门课和这个实验，我对机器学习这个目前热门的大方向有了一个系统性的认识。在过去我因为其他的课程也略微接触到了 AI 领域的技术，所以在刚接触这个实验的时候，我对于深度学习的任务步骤还是有一个初步的认识，对 MNIST, kaggle, CNN 等名词比较熟悉，这也让我能够顺利跨过实验的初期。

但是在使用 numpy 手动实现 CNN 网络的时候，我发现我对于机器学习实践的细节还是不够熟悉，对于全连接层，卷积层等的前向和反向传播的过程一知半解，遇到了很多困难，好在最后通过查阅文献，阅读 pytorch 代码等方法，逐渐解决了一个个问题。

在整个模型实现后，我开始逐步调参，从最开始准确度只有不到 96，发现模型的反向传播梯度有问题，到尝试使用深度神经网络，却因为使用 CPU 模型训练太慢而放弃，最后转头选择 Lenet-5 这个卷积神经网络的开山鼻祖，通过自己的探索和调参取得了不错的结果。

回头来看这次实验，还有很多做得不足的地方可以改进，对于模型，由于最后发现有一些过拟合的现象，我通过添加 Dropout 来减缓问题，但是其实还可以通过旋转，裁剪等方法进行数据集增强，通过这来提高模型的泛化能力。

通过这个实验，我从零开始使用 numpy 实现了卷积神经网络和一个 K 邻近算法，然后和我已经很熟悉的预训练模型 ResNet50 进行对比，让我对与机器学习和神经网络有了一个更加全面的认识，我也锻炼了自己的编程技能和实际问题解决能力，这都让我受益良多。

参考文献

- [1] 周志华, 机器学习. 清华大学出版社, 2016.
- [2] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.
- [3] M. Dukhan, "The indirect convolution algorithm," *arXiv preprint arXiv:1907.02129*, 2019.
- [4] sebgao, "ctensor," 2021.
- [5] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
- [6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.
- [7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.