电 子 科 技 大 学

实 验 报 告

学生姓名：周杰锋	学 号：**2021060904008**

一、实验项目名称：MNIST 手写数字识别

二、实验目的：

一、基本任务（按照 PPT 实现基本功能）

1.基于 QQ 群里给出已处理的 MINST 数据库，设计和训练 CNN，识别手写体数字。

2.用尽量简单的结构，获得最优性能（最低标准：正确率>97%，训练用时<7min）。

3.在不增加总的可调权值系数的前提下， 可对网络结构做修改(如:加深) (系数≤9x9x20+ 2000x100+ 100x10)。

4.可采用 SGD、批量、小批量、动量等算法。

5.编程语言不限， 但必须有具体训练步骤(每层 δ 的计算代码)不借助任何机器或深度学习库。

6.提交源代码 + 详细实验分析报告

（没有固定模板，报告内不用再附完整的代码，规范整洁说明充分即可。

包含：网络设计、训练和调试方法、实验结果与分析比较，结论等)

二、可选的提升任务

1.多次调试并进行对比（多展示图片表格，并加以标注说明）

2.优化模型，尝试多层卷积池化

3.滤波器、特征图可视化...

4.错误样本可视化及分析...

5.针对错误样本增广扩充数据集，增加噪声、滤波、图片横竖压缩...

...

三、实验内容：

1） 下载并配置好相关 MNIST 数据集、Python 环境

2） 构建自己的 CNN 模型

3） 训练自己的 CNN 模型并展示训练过程中的情况

四、实验环境：

1）硬件平台：CPU:AMD Ryzen 7 5800U@1.90GHz; GPU: NVIDIA GeForce RTX 3050 LAPTOP; 内存: 16GB@4266MHz

2） 开发环境：Windows 11 Pro for Workstations Insider Preview25987.1000，Python 3.9.7，Numpy 1.23.4

3）测试环境：同上

五、CNN 网络设计

1）本次实验仅通过 numpy 实现 CNN 模型的构建，一个典型的 CNN 网络包含的 Layer 有：Linear、Softmax、ReLu、Pooling、Flatten、Conv2D。可选的有 BatchNorm、Dropout 等（源代码已实现，但是出于简单结构考虑并未参与构建 CNN 模型中）

2）构建 CNN 模型后，还需要构建合适的训练方法，这里使用 Numpy 构建经典的 Adam 优化器，损失函数使用 Cross Entropy。

3）本次代码结构按照典型的 Pytorch 结构进行构建，增加代码的可理解性与可读性。

4）在实现过程中，为了方便起见，ReLu 激活函数内置在所需要的 Layer 的 forward 和 Backword 过程中。

1. Linear Layer 设计

```
Linear 类:
    初始化(输入特征数, 输出特征数, 是否使用偏置):
        初始化权重矩阵 W
        初始化偏置 b
        记录是否使用偏置
        初始化偏置梯度 db 为零
        计算参数大小(W 和 b 的元素总数)

    正向传播(输入 A):
        保存输入 A
        计算 Z = A @ W + b
        计算输出 A = max(Z, 0) (ReLU 激活函数)
        返回输出 A

    反向传播(上一层误差 dZ):
        计算当前层误差 dA = dZ * (A > 0) (ReLU 反向传播)
        计算权重梯度 dW = A.T @ dA
        如果使用偏置:
            计算偏置梯度 db = sum(dA, axis=0)
        计算传递到上一层的误差 dZ = dA @ W.T
        返回上一层误差 dZ
```

2. Linear_Softmax Layer 设计

1）在构建过程中，因为 Softmax Layer 的前面一层是 Linear Layer，为了方便起见将两个层合并为一个 Layer.

```
LinearSoftmaxLayer 类:
```

```
    初始化(输入特征数, 输出特征数):
        初始化权重矩阵 W
        初始化偏置 b
        计算参数大小(W 和 b 的元素总数)

    正向传播(输入 A):
        保存输入 A
        计算 Z = A @ W + b
        计算 A_relu = max(Z, 0) (ReLU 激活函数)
        计算 expA = exp(A_relu - max(A_relu))
        计算输出 A = expA / sum(expA, axis=-1) (softmax 函数)
        返回输出 A

    反向传播(上一层误差 dZ):
        计算当前层误差 dA = dZ
        计算 dA_relu = dA * (A > 0) (ReLU 反向传播)
        计算权重梯度 dW = A.T @ dA_relu
        计算偏置梯度 db = sum(dA_relu, axis=0)
        计算传递到上一层的误差 dA_prev = dA_relu @ W.T
        返回上一层误差 dA_prev
```

3. Pooling Layer 设计

```
PoolingLayer 类:
    初始化(filter_size, stride, mode):
        设置池化 filter 大小
        设置池化步长 stride
        设置池化模式(最大池化或平均池化)

    正向传播(输入 A_prev):
        保存输入 A_prev
        根据输入维度、filter 大小、stride 计算输出维度
        初始化输出 A
        对每个样本:
            对输出的每个高度位置:
                对输出的每个宽度位置:
                    对每个通道:
                        从输入取出对应窗口
                        如果是最大池化模式:
                            输出为窗口最大值
                        如果是平均池化模式:
                            输出为窗口平均值
```

```
        返回输出 A

    生成最大值掩码(窗口 x):
        返回窗口 x 的最大值掩码

    分配梯度值(dz, shape):
        计算 dz 在 shape 区域的平均值
        返回形状为 shape,值都等于平均值的矩阵

    反向传播(上层梯度 dA):
        根据输入 A_prev 的维度初始化 dA_prev
        对每个样本:
            对 dA 的每个高度位置:
                对 dA 的每个宽度位置:
                    对 dA 的每个通道:
                        确定对应窗口在 A_prev 中的位置
                        如果是最大池化模式:
                            生成当前窗口的最大值掩码
                            将掩码对应位置的 dA 值加到 dA_prev 对应位置
                        如果是平均池化模式:
                            将 dA 对应值平均分配到 dA_prev 窗口区域
        返回 dA_prev 作为传递到上一层的梯度
```

4. Flatten Layer 设计

```
Flatten 层:
    初始化():
        无需执行任何操作

    正向传播(输入 Z):
        保存输入 Z 的原始形状
        将输入 Z 展平为 2D 张量,第一维是批大小,第二维是其他维度的乘积
        返回展平后的 Z

    反向传播(上层梯度 dZ_prev):
        将 dZ_prev 重新整形为之前保存的输入形状
        返回重新整形后的梯度 dZ
```

5. Conv2D Layer 设计

```
Conv2D 层:
```

```
初始化(输入通道数,输出通道数,卷积核大小,输入形状,padding,stride,是否使用偏置):
    初始化权重 W
    初始化偏置 b
    设置 stride 和 padding 大小
    记录是否使用偏置
    计算参数总数

零填充(输入 X,填充大小 pad):
    在 X 的高度和宽度维度两侧填充 pad 个 0
    返回填充后的 X

正向传播(输入 A_prev):
    保存输入 A_prev
    根据输入 A_prev 的形状计算输出形状
    初始化输出 Z
    填充输入 A_prev
    对输出的每个通道:
        计算当前滤波器与输入的卷积,得到本通道输出
    将偏置 b 加到输出 Z
    对 Z 进行 ReLU 激活
    返回激活后的 Z

反向传播(上层梯度 dZ):
    初始化对应形状的 dA_prev,dW,db
    填充输入 A_prev 和 dA_prev
    根据 ReLU 函数,对 dZ 进行处理
    对每个样本:
        对输出的每个高度位置:
            对输出的每个宽度位置:
                对输出的每个通道:
                    计算当前窗口在 A_prev 中的位置
                    将当前通道梯度加到 dW 对应位置
                    将权重 W 和当前梯度的卷积加到 dA_prev 对应位置
                    如果使用偏置,则将当前梯度加到 db
        将填充后的 dA_prev 复原为实际大小
    返回 dA_prev 作为传递到上一层的梯度
```

6. BatchNorm Layer

```
BatchNorm 层:
    初始化(输入形状):
        初始化缩放权重 W 和偏移量 b
        计算参数总数
```

正向传播(输入 x):
    将输入 x 展平为二维张量(批大小,特征数)
    计算均值 mu = 1/N * sum(x)
    计算偏移量 xmu = x - mu
    计算方差 var = 1/N * sum(xmu**2)
    添加一个很小的数 epsilon 确保数值稳定性
    计算标准差的倒数 ivar = 1 / sqrt(var + epsilon)
    标准化输入 xhat = xmu * ivar
    缩放 xhat: Wx = W * xhat
    偏移 Wx: out = Wx + b
    记录中间结果到 cache
    将 out 恢复到输入的原始形状
    返回 out

反向传播(上层梯度 dout):
    将 dout 展平为二维张量
    从 cache 中取出中间结果
    计算 db = sum(dout)
    计算 dW = sum(dWx * xhat)
    ...
    利用链式法则依次计算 divar,dsqrtvar,dvar,dsq,dxmu1,dxmu2,dx1,dmu,dx2
    最终得到传递到上一层的梯度 dZ = dx1 + dx2
    将 dZ 恢复到 dout 的原始形状
    返回 dZ

7. CNN 网络设计

CNN 类:
    初始化(输入形状,输出类别数,卷积层参数,池化层参数,全连接层参数):
        初始化有序字典 layers 用于存储网络层
        根据输入参数依次添加以下层:
            Conv2D 层: 添加指定数量的卷积层
            PoolingLayer 层: 添加对应数量的池化层
            Flatten 层: 添加展平层
            Linear 层: 添加指定数量的全连接层
            LinearSoftmaxLayer 层: 添加最后一层全连接+softmax 层

    正向传播(输入 X,是否打印中间输出):
        对 layers 中的每一层进行正向传播计算
        可选地打印每层输出的形状和参数大小
        返回网络的输出

反向传播(上层梯度 dZ):
按相反顺序对 layers 中的每一层进行反向传播计算
返回传递到数据层的梯度 dZ

设置权重(权重字典):
根据权重字典,为具有可训练参数的层设置权重 W 和偏置 b

获取权重():
返回网络中所有可训练参数层的当前权重 W 和偏置 b

获取权重梯度():
返回网络中所有可训练参数层的当前权重梯度 dW 和偏置梯度 db

该 CNN 类封装了卷积神经网络的结构,包括卷积层、池化层、全连接层等。在初始化时根据指定参数构建网络层,提供了正向传播和反向传播的方法,并且可以设置和获取网络中可训练参数的值和梯度。这个类的设计使得构建、训练和测试卷积神经网络变得更加方便。

最后在本次实验中构建的 CNN 网络的可训练参数以及结构如下:

```
-----
20994
input shape (2, 28, 28, 1)
after  conv1 (2, 14, 14, 16)
param:  160
after  pool1 (2, 7, 7, 16)
after  conv2 (2, 4, 4, 24)
param:  13848
after  pool2 (2, 2, 2, 24)
after  flatten (2, 96)
after  linear1 (2, 64)
param:  6208
after  batch_norm_linear1 (2, 64)
param:  128
after  linear_softmax (2, 10)
param:  650
(2, 28, 28, 1)
-----
```

8. Adam 优化器设计

Adam 优化器:
初始化(网络 net, 学习率,beta1,beta2,epsilon):
获取网络中所有可训练参数
初始化学习率,beta1,beta2,epsilon
初始化动量向量 VW,Vb

```
        初始化 RMSProp 向量 SW,Sb,初值全为 0

    执行一步优化(网络 net):
        获取网络当前参数
        获取网络当前参数梯度
        对每个可训练参数:
            计算动量 VW,Vb
            计算 RMSProp SW,Sb
            根据动量和 RMSProp,更新当前参数

    设置新学习率(新学习率 lr):
        更新学习率
```

Adam 优化器是结合了动量(Momentum)和 RMSProp 的优化算法。它通过引入动量项和 RMSProp 自适应学习率,帮助优化过程更快地收敛并且减少振荡。
在初始化时,Adam 为每个可训练参数分别初始化一个动量向量 V 和 RMSProp 向量 S,用于记录动量和 RMSProp 的累积值。
在每一步优化时,Adam 首先根据当前梯度和先前动量更新动量 V,再根据当前梯度平方和先前 RMSProp 向量 S 更新 RMSProp 向量 S。最后,Adam 利用这两个向量的值对参数进行更新。
通过调整 beta1,beta2 等超参数,Adam 可以控制动量和自适应学习率的效果。

五、实验数据及结果分析：

1）所构建的 CNN 模型包括 2 层卷积层，2 层 max-pooling 层，经过 Flattern 层后，还有 3 层 Linear 层，一层 BatchNorm 层，以及最后的 softmax 层。每个卷积层以及 Linear 层之间使用 RELU 激活函数。输出层为 log_softmax 层归一化输出。模型可训练参数为 20994。

1. CNN 模型结构

模型的具体网络结构如下（第一个维度为 Color Channel）

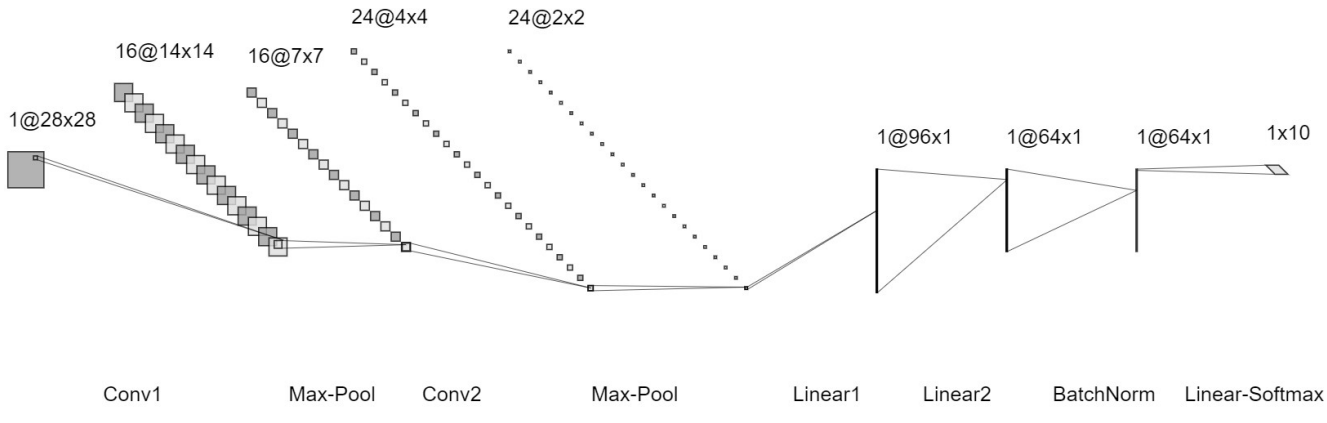

图 1：本次实验所构建的 CNN 网络结构拓扑图

2. 数据集情况

2）模型的训练数据为通过读取 `MNISTData.mat`获取，所得到结果如下：

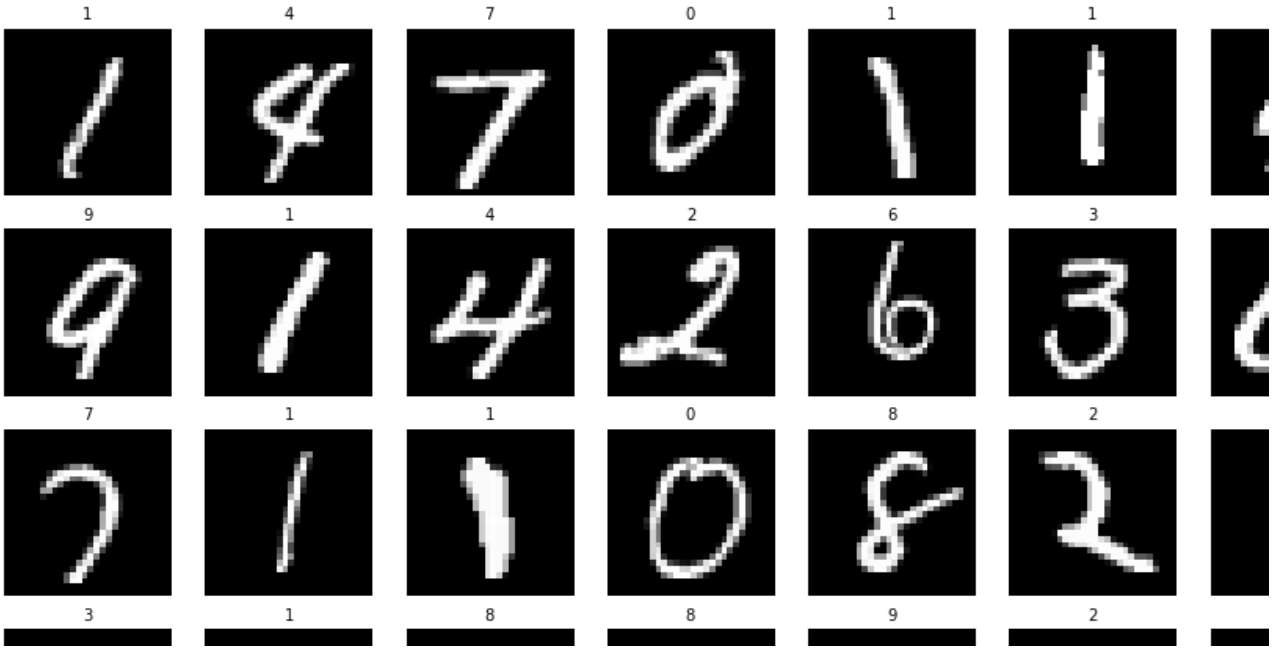

图 2：随机读取到的 MNIST 数据，并可视化结果

3. 模型训练过程

3）本次模型的参数以及超参数经过多重训练后选取（由于 CPU 限制，单次训练需要较长时间，以及训练时作业要求还没有详细公布，所以没有记录），最后选定的训练超参数为 Batch_size = 128, Learning_rate = 0.01, Seed=42, 选用 Cross entropy 作为损失函数。**训练过程为每个 epoch 中测试 40 次模型在验证集上面的效果，在 Test Acc 达到 95%更新优化器的学习率为 0.005；在 0.95 时更新为 0.002；如果 Test Acc 达到 97%+，则终止迭代。**

第一个 Epoch 的训练结果如下：

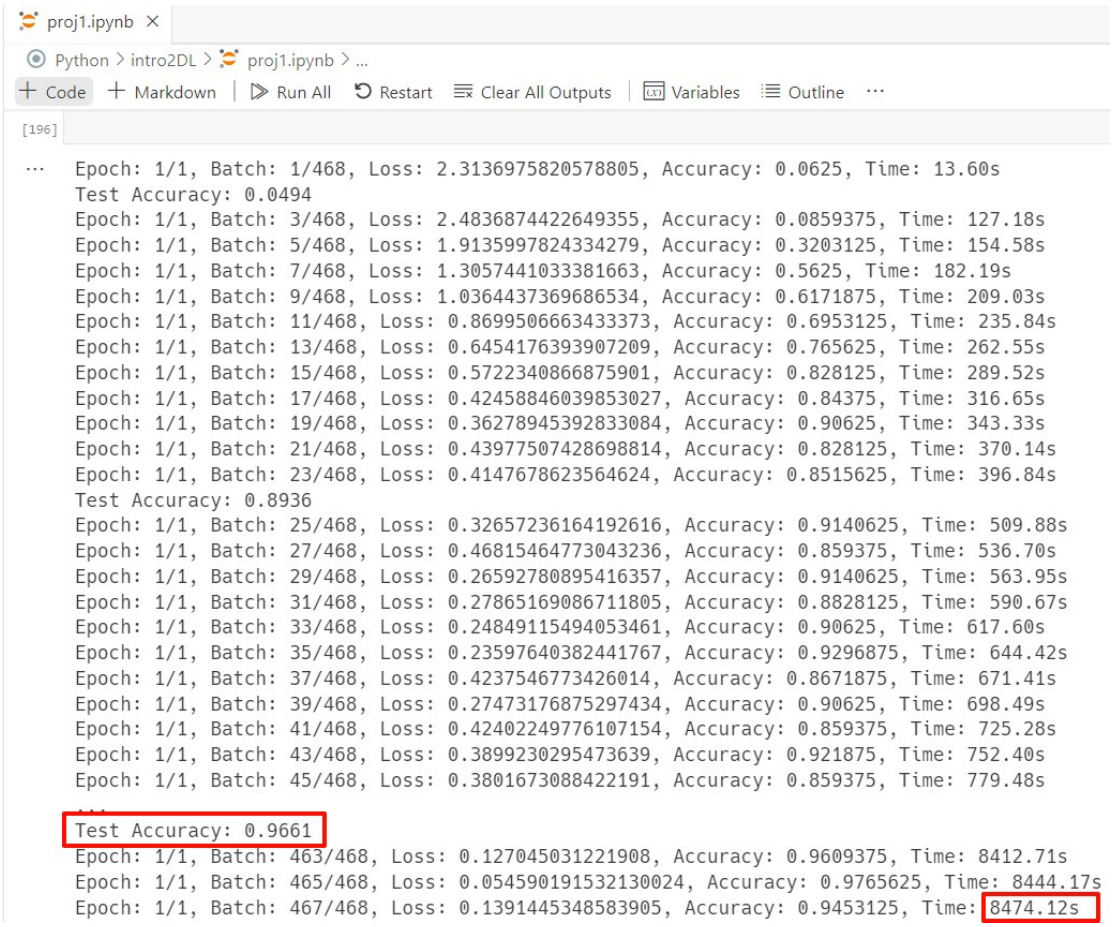

图 3：Epoch 1 的训练结果，由于 CPU 性能原因，训练用时较长

proj1.ipynb

Python > intro2DL > proj1.ipynb > ...

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | ...

```
44                break
[197]
```

```
Epoch: 2/2, Batch: 1/468, Loss: 0.08185436729185819, Accuracy: 0.96875, Time: 8779.01s
Test Accuracy: 0.9662, Time: 8779.01s
Epoch: 2/2, Batch: 3/468, Loss: 0.13201863413307724, Accuracy: 0.9375, Time: 8899.33s
Epoch: 2/2, Batch: 5/468, Loss: 0.07516161501927951, Accuracy: 0.9765625, Time: 8926.64s
Epoch: 2/2, Batch: 7/468, Loss: 0.15616060350279934, Accuracy: 0.9453125, Time: 8953.35s
Epoch: 2/2, Batch: 9/468, Loss: 0.05181389429479412, Accuracy: 0.9921875, Time: 8979.99s
Epoch: 2/2, Batch: 11/468, Loss: 0.12109130705209378, Accuracy: 0.9453125, Time: 9006.56s
Test Accuracy: 0.963, Time: 9006.56s
Epoch: 2/2, Batch: 13/468, Loss: 0.1295823645326285, Accuracy: 0.96875, Time: 9120.63s
Epoch: 2/2, Batch: 15/468, Loss: 0.1883956871892536, Accuracy: 0.9453125, Time: 9147.07s
Epoch: 2/2, Batch: 17/468, Loss: 0.10609803772479776, Accuracy: 0.9609375, Time: 9173.62s
Epoch: 2/2, Batch: 19/468, Loss: 0.24652182546870613, Accuracy: 0.9453125, Time: 9200.12s
Epoch: 2/2, Batch: 21/468, Loss: 0.19148604629855204, Accuracy: 0.9296875, Time: 9226.68s
Epoch: 2/2, Batch: 23/468, Loss: 0.0399525545210815, Accuracy: 0.9765625, Time: 9253.20s
Test Accuracy: 0.9655, Time: 9253.20s
Epoch: 2/2, Batch: 25/468, Loss: 0.1651279733882175, Accuracy: 0.953125, Time: 9366.70s
Epoch: 2/2, Batch: 27/468, Loss: 0.09382879261511676, Accuracy: 0.9765625, Time: 9393.34s
Epoch: 2/2, Batch: 29/468, Loss: 0.045654205271425524, Accuracy: 1.0, Time: 9420.36s
Epoch: 2/2, Batch: 31/468, Loss: 0.12866857595999606, Accuracy: 0.9609375, Time: 9448.91s
Epoch: 2/2, Batch: 33/468, Loss: 0.0739675372233628, Accuracy: 0.9765625, Time: 9479.58s
Test Accuracy: 0.9651, Time: 9479.58s
Epoch: 2/2, Batch: 35/468, Loss: 0.158803578806009, Accuracy: 0.953125, Time: 9610.78s
Epoch: 2/2, Batch: 37/468, Loss: 0.14469159555137723, Accuracy: 0.953125, Time: 9643.11s
Epoch: 2/2, Batch: 39/468, Loss: 0.18366349650136105, Accuracy: 0.9453125, Time: 9674.18s
Epoch: 2/2, Batch: 41/468, Loss: 0.1727778035860404, Accuracy: 0.96875, Time: 9704.82s
...
Epoch: 2/2, Batch: 153/468, Loss: 0.12895990467881377, Accuracy: 0.96875, Time: 12144.10s
Epoch: 2/2, Batch: 155/468, Loss: 0.044295835170931185, Accuracy: 0.9921875, Time: 12170.08s
Test Accuracy: 0.9701, Time: 12170.08s
Converged!
```

图 4：Epoch 2 的训练结果，在 155/468 个 Batch 时候，模型在验证集上的效果达到 97%，因此终止训练

4. 模型训练结果

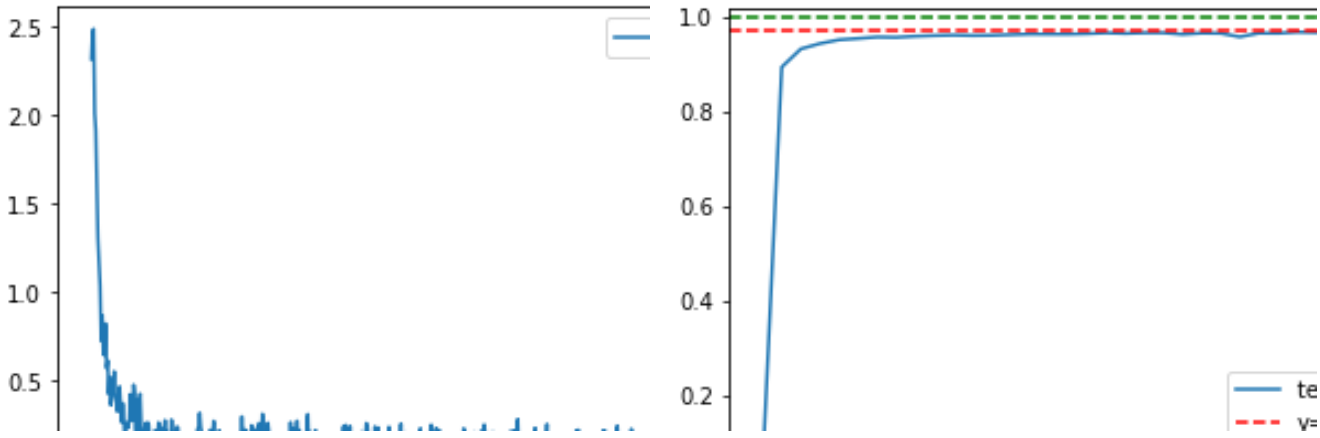

图 5：模型在训练过程中不同指标的变化 左）Train Loss； 右）Test Acc

可以看出，模型在训练初期就开始快速收敛，不仅 Training Loss 在快速下降，Test Acc 也迅速上升。但是随着在达到第一个 Epoch 的一半的时候，模型进入了振荡收敛的时候，开始艰难的收敛过程，这一部分的振荡有模型自身结构的原因（结构过过于简单）；也有此时优化器学习率过高的原因。但是在第二个 Epoch 时候，模型最终达到了 97%的准确率。

5. 验证集中错误样本分析

把整个验证集数据输入进去，得到 299 个错误样本，我们随机选取其中一部分的错误样本：

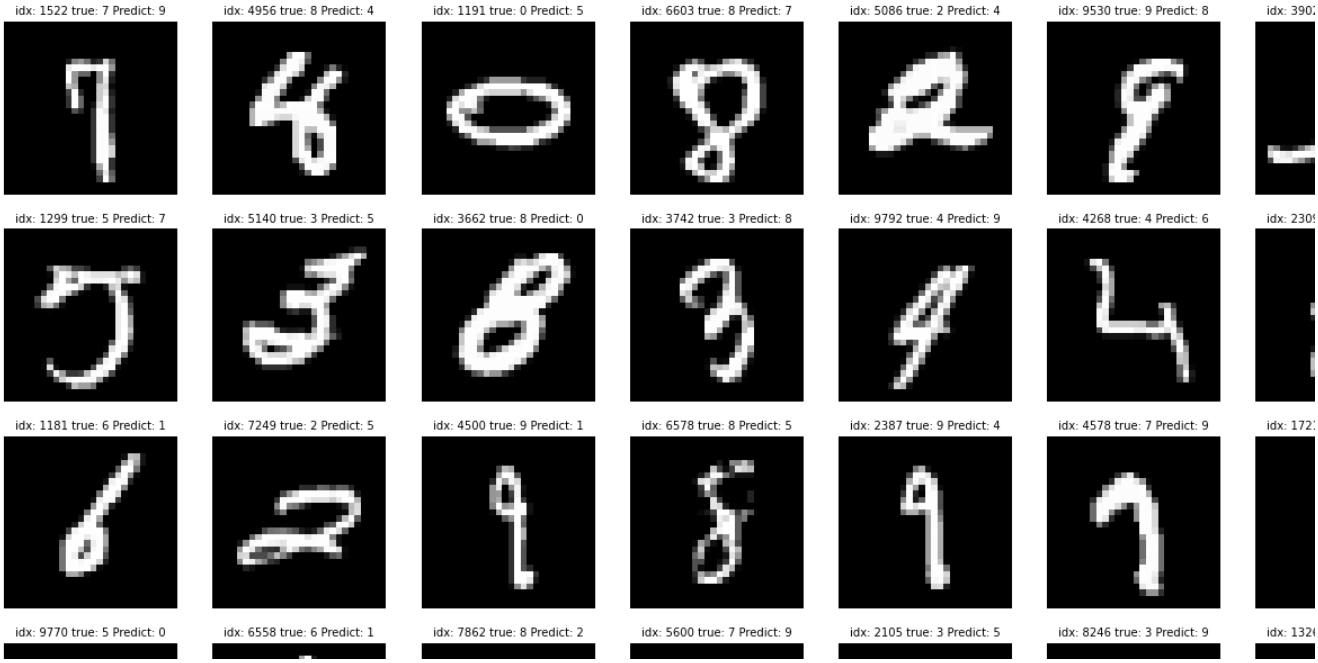

图 6：模型在验证集中错误分类样本

从这些样本直观来看，部分样本是由于形状与预测的数字的特征过于相似，比如 id 1522, 1984, 9792 等，还有一部分则是模型自身原因，比如 idx 4956, 1191 等。考虑到我们只用大约 2w 个参数就实现了 97%的准确率，这个结果是可以接受的。

6. 特征提取可视化

我们随机输入一张图片，通过将每层的结果可视化出来，可以一定程度一窥模型的特征提取能力。

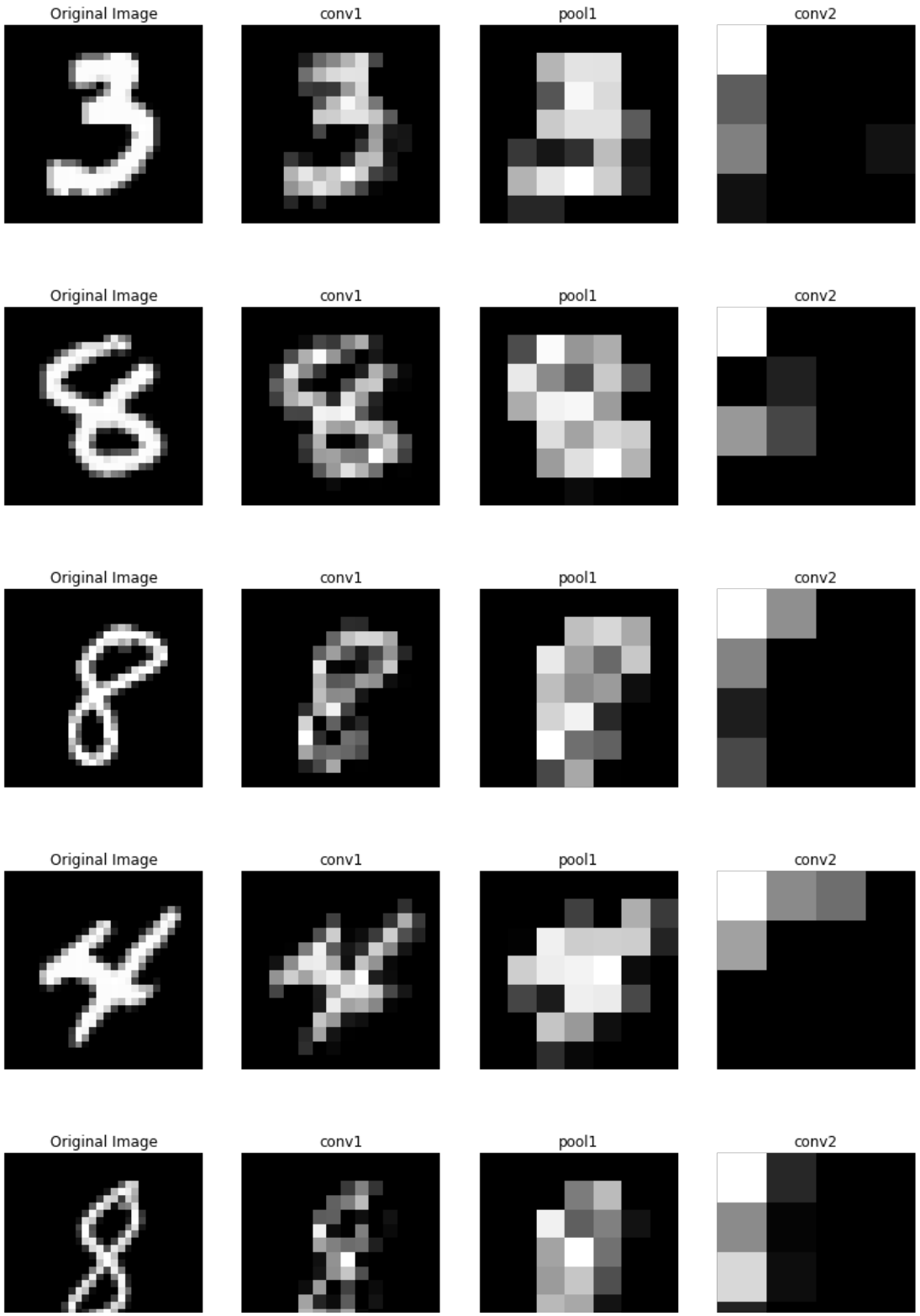

图 7：随机读取图片，然后可视化每层的输出。注意这里只可视化了其中一个 Channel，所以后面的可视化仅为上层输入的局部特征提取

可以看出，模型的大致特征提取逻辑是，从整体到局部，将特征拆分成一个个小细节后，通过线性层实现分类。

六、实验总结：

本次实验仅通过 Numpy，以类 Pytorch 代码风格结构的方式实现了 CNN 的构建，包括各种 CNN 网络层的构建，前向与反向传播的设计，以及 Adam 优化器的实现。最终使用 9 层共 20994 个参数的 CNN，经过大约 1.25 个 Epoch 的训练，成功在验证集上实现了 97.01%的正确率。后续还实现了错误样本分析，以及特征提取可视化，完成了基本任务和部分可选任务。

本次实验的缺陷有：

1. 仅使用 CPU 版本的 Numpy，未优化训练方式（比如使用 CUDA 版本的 Numpy: Cupy 来加快训练）
2. 仅记录了一种网络结构的结果，在前期探究参数的时候的训练结果未记录下来，缺乏对比。
3. 其它提升正确率的层比如 BatchNorm， LeakyReLu，Dropout 等并没有部署进去，模型仍有改进空间。

报告评分：