深度学习导论HW1

PB22151796-莫环欣

本次的两个实验都比较经典，在实验一中实现了一个简易的`FNN`用于波士顿房价预测回归任务；在实验二中则实现了一个也比较简易的`CNN`用于用于`MNIST`手写数字识别分类任务。两处代码结构基本相同，不同之处主要在于网络结构与评价指标（回归任务不讨论准确率），同时都支持通过传参修改结构以完成要求中的测试任务（方便画图）

- 包环境
  - `python`:3.10.10
  - `scikit-learn`:1.6.1
  - `torch`:2.6.0
  - `matplotlib`:3.10.1
  - `pandas`:2.2.3
  - `numpy`:2.0.2

作业 1-1

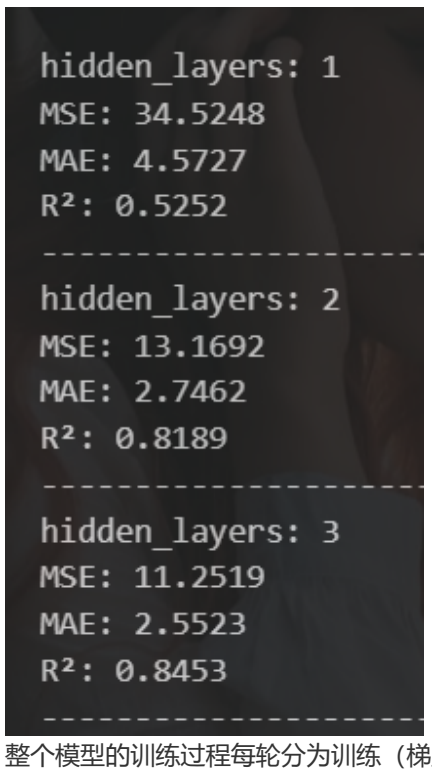
实验步骤

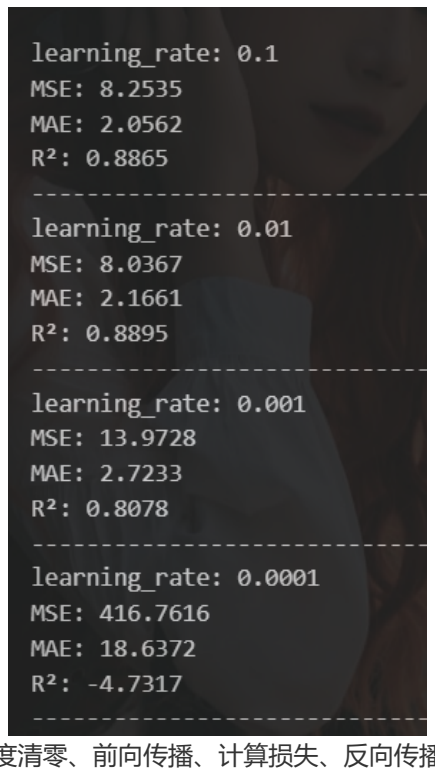
首先通过`sklearn`获取到数据集，并划分为六份（训练集四份、验证集和测试集各一份），随后对数据进行标准化并装载到加载器中。

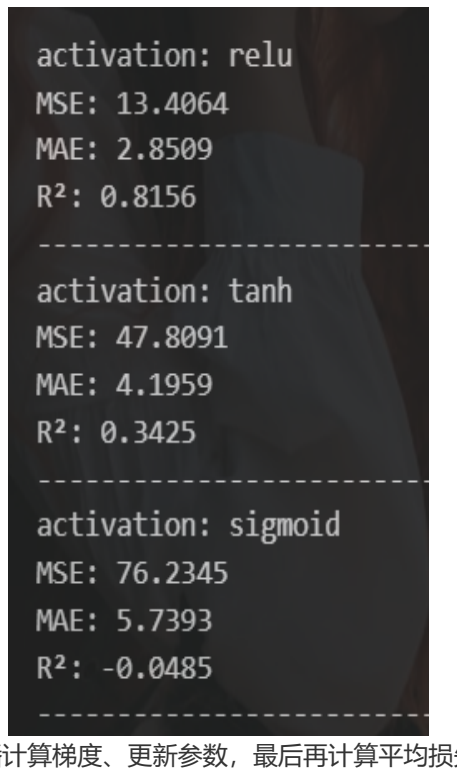
对于`FNN`，这里为其设定了一些默认参数，其中输入层维度与特征数匹配，隐藏层数目和激活函数类型参数方便后续的两个测试；整体结构比较简单，每层`Linear`后都跟着一层激活函数，最后再接上一层`Linear`将输出映射到唯一的一个维度（即上面的房价`MEDV`）

```
def __init__(self, input_dim=13, hidden_layers=1, activation="relu")
```

模型的训练上，采用`MSELoss`为损失函数，这里测试出来的`Adam`优化器的结果比较抽象，于是硬编码`SGD`了





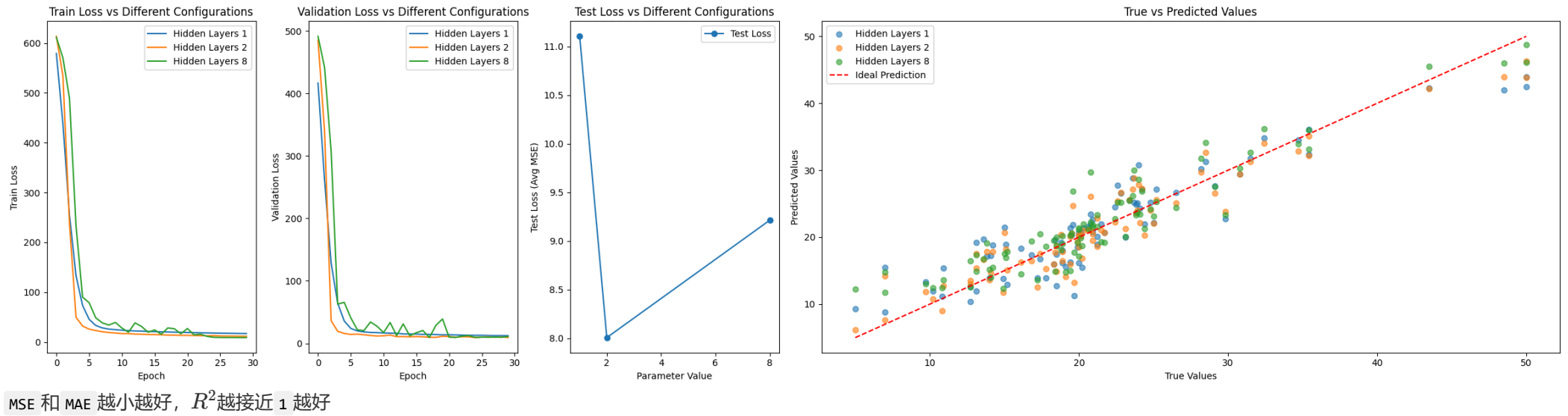


整个模型的训练过程每轮分为训练（梯度清零、前向传播、计算损失、反向传播计算梯度、更新参数，最后再计算平均损失并返回）、验证（带入参数进行预测，并获取`mse`）、调整学习率（借助`ReduceLROnPlateau`学习率调度器，损失一段时间未改善时降低学习率）三步，最后返回训练与验证的损失。

测试部分跟验证部分的逻辑类似，只是将数据集换成了测试集

网络深度的影响

固定其它参数后，对`1`层、`2`层、`8`层隐藏层分别进行了测试



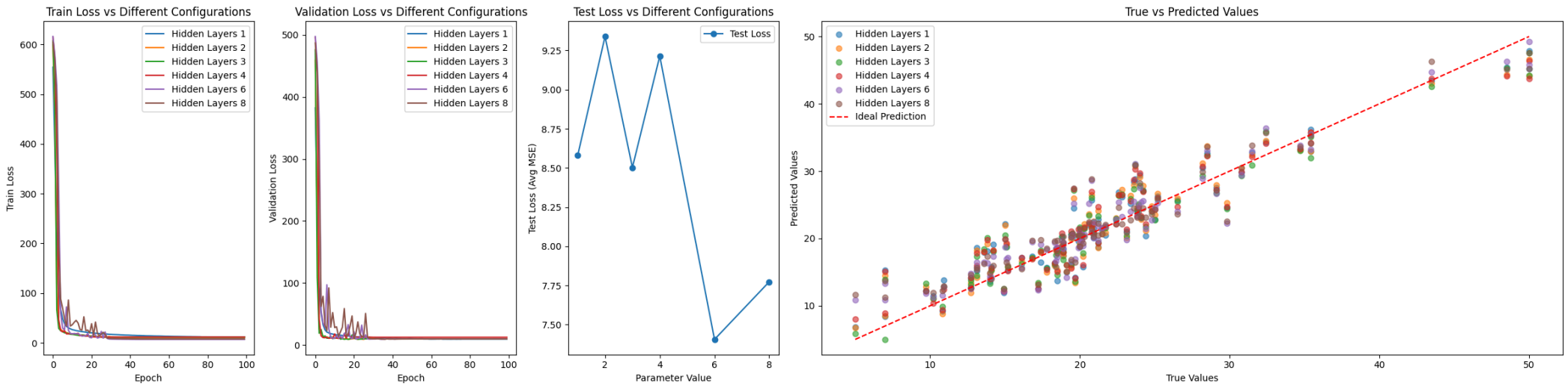
`MSE`和`MAE`越小越好，R^2越接近`1`越好

hidden_layers	MSE	MAE	R²
1	11.1033	2.5860	0.8473

hidden_layers	MSE	MAE	R^2
2	8.0066	2.1820	0.8899
8	9.2127	2.3669	0.8733

通过这里的结果图，可以看到：从一层升到两层时，训练/验证/测试损失均有显著下降；继续增加到八层，其训练/验证损失也有所下降，但在训练与验证过程中出现了较为严重的震荡，最终的测试总损失较两层时有显著回升。

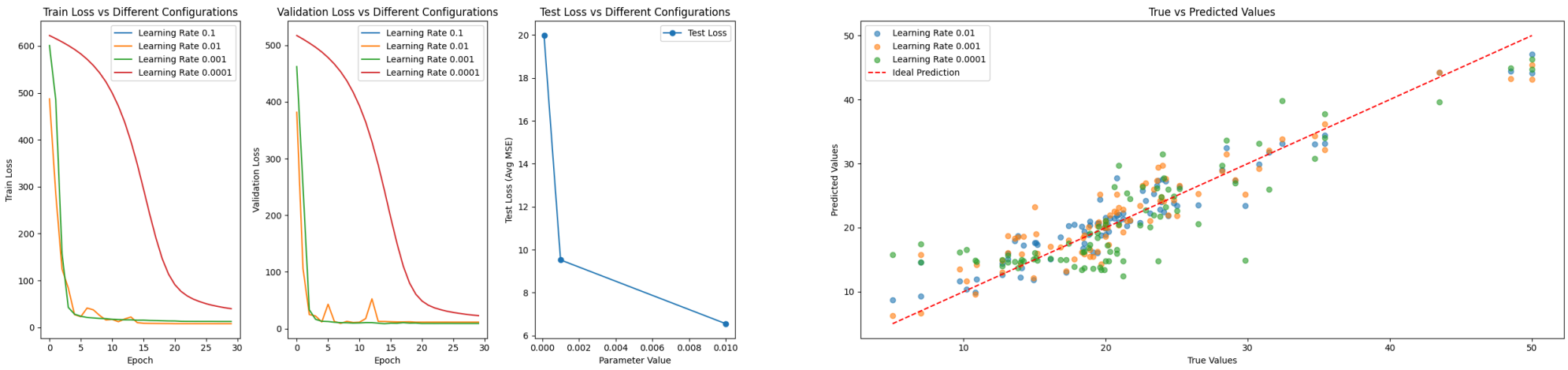
此外，对于八层网络，其训练损失在最后`7`个`epoch`开始低于其它两组，但验证损失的最后两个`epoch`略高一些，同时测试损失也明显的高。基本可以认为其出现了轻微的过拟合现象，但由于每轮保存的是最佳的参数，继续增大训练轮次的影响也不大



学习率的影响

> 为减少其它因素的影响，此处设置两层隐藏层与`relu`激活函数

学习率太高会导致梯度爆炸（例如学习率为`0.1`时）



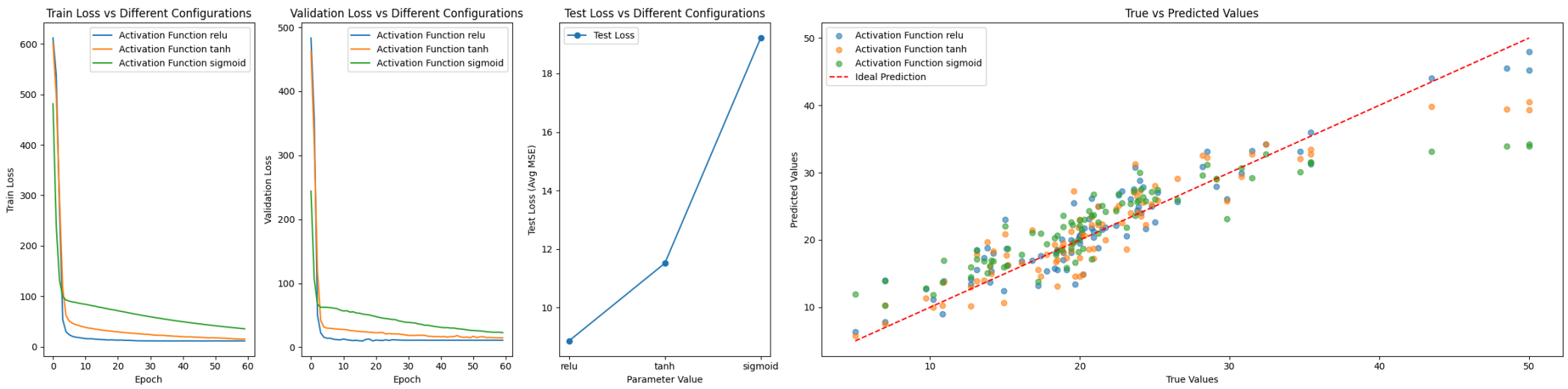
learning_rate	MSE	MAE	R^2
0.01	6.5521	2.0118	0.9099
0.001	9.5214	2.4073	0.8691
0.0001	19.9742	3.4165	0.7253

可以看到在适当范围内，学习率越高收敛速度就越快但越不稳定（震荡幅度较大，不如低学习率时平缓）

如果学习率太大，模型的训练可能直接爆掉（例如学习率为`0.1`时全部输出`nan`），不爆的话也可能引起震荡（`0.01`），但学习率太小的时候收敛速度又太慢（参考上图中学习率为`0.0001`的曲线，收敛速度远远慢于其它两组（`0.1`那组爆掉了不考虑））

激活函数的影响

在默认配置下，对于三种激活函数`ReLU、Tanh、Sigmoid`，训练效率和预测效果依次递减，`relu`十轮不到就收敛了，`tanh`到六十轮时基本收敛，而`sigmoid`离收敛还有很长一段距离。`relu`的鲁棒性较好，外加之前在处理数据的时候只进行了标准化，导致后两个激活函数出现了不同程度的梯度消失问题，另外`sigmoid`还限制了输出范围。



activation	MSE	MAE	R^2
relu	8.8628	2.2904	0.8781

$$relu(x) = \max(0, x)$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

activation	MSE	MAE	R^2
tanh	11.5156	2.5597	0.8416
sigmoid	19.2103	3.1946	0.7358

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

作业 1-2

> **Note**
>
> 分析中若未特别说明，采用的就是对随机`3000`张灰度图进行训练的模型，另外在云平台上使用全量数据集等比划分重新跑了一遍实验，各部分的预测准确率基本稳定在`99%+`，结果附带在代码文件的输出中
>
> 按理说应该先搭建一个能运行但效果较差的裸模型，再逐渐增加需要分析的各部分功能进行分析的，但一开始没意识到（基础版本训练完都稳定`97%`了），等准备分析的时候沉没成本又有点太高了，下次会吸取教训
>
> 下面图像中的`check`同`validation`

模型说明

这里首先随机抽取`5%`的数据（在本地运行），并参考上一个实验的划分将数据集分为`训练:验证:测试=4.8:1.2:1`，

尝试添加了以下数据增强但似乎完全是负优化（也可能是跟其它的没配合好，后面移除了）

```
transforms.RandomRotation(10),  # 随机旋转 ±10 度
transforms.RandomAffine(0, translate=(0.1, 0.1)),  # 随机平移 10% 范围
transforms.ColorJitter(brightness=0.2),  # 亮度扰动（针对灰度图）
```

对于模型结构，默认两个卷积层，第一层设置一个通道（灰度，`28x28`），每个卷积层之下设置池化层防止过拟合并减小图片的尺寸，最后再接到两个全连接层上输出十个分类。激活函数参考`FNN`实验的结果选择了`relu`；最大池化适合提取显著特征，平均池化适合分割图像，于是这里选择使用最大池化`Maxpool2d`

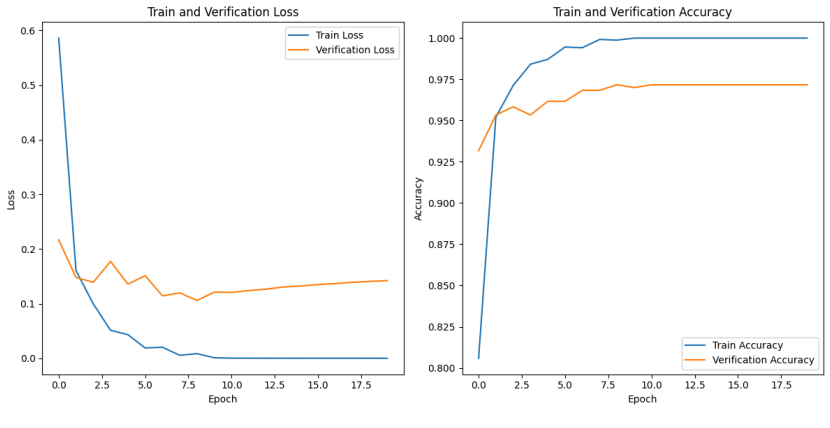
后续增加网络深度就是重复叠加`conv`与`pool`

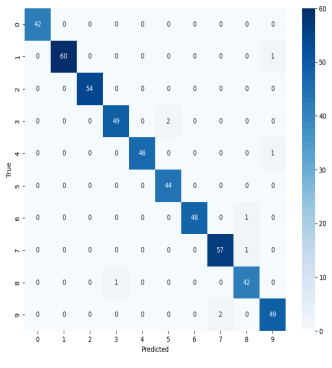
```
conv-->pool-->conv-->pool-->fc-->fc
```

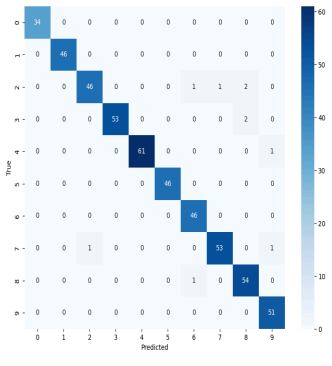
这里模型的训练/验证/测部分整体结构与上面基本相同，不同点主要在于使用了更适合分类问题的交叉熵损失函数和`Adam`优化器

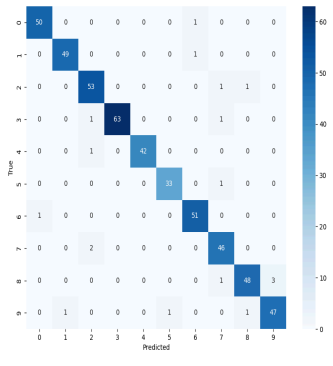
接下来训练了模型，可以看到训练曲线基本没什么问题，不过验证曲线的`loss`有回升趋势（好在验证曲线表现还行）

随后将最佳一版模型在三组不同测试集上进行了评估并作混淆矩阵热力图







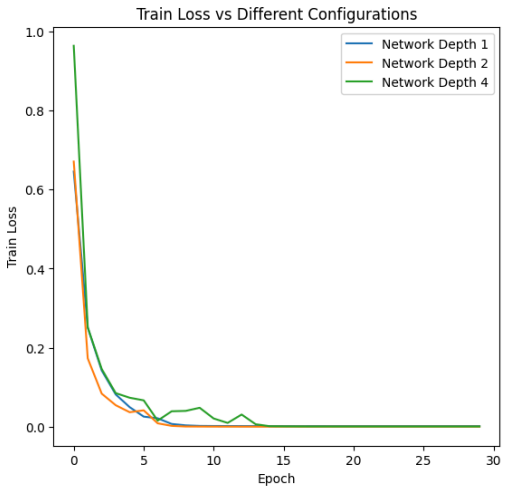


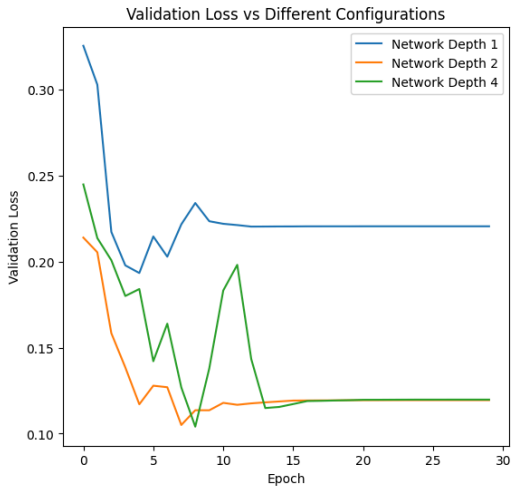
Total acc	0	1	2	3	4	5	6	7	8	9
0.9820	1.0000	1.0000	1.0000	0.9800	1.0000	0.9565	1.0000	0.9661	0.9545	0.9608
0.9800	1.0000	1.0000	0.9787	1.0000	1.0000	1.0000	0.9583	0.9815	0.9310	0.9623
0.9640	0.9804	0.9800	0.9298	1.0000	1.0000	0.9706	0.9623	0.9200	0.9600	0.9400

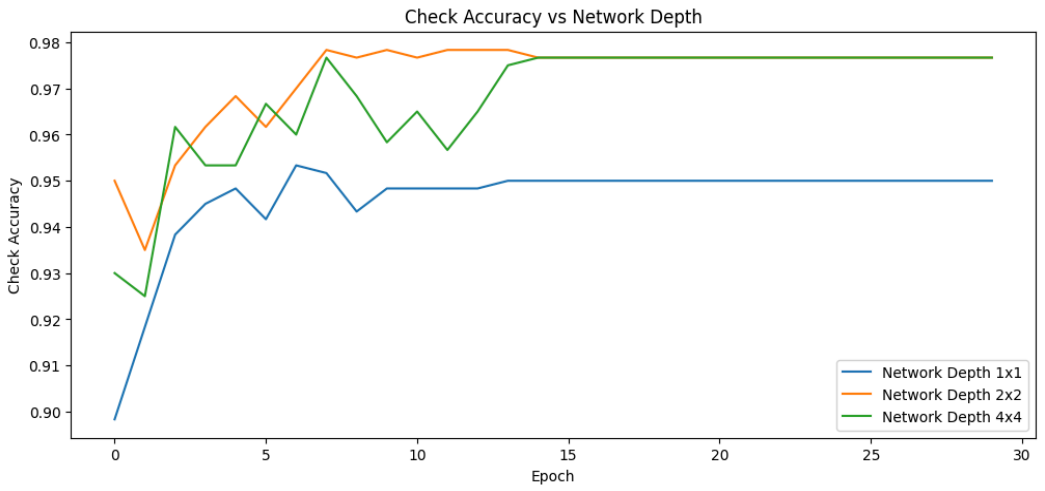
网络深度的影响

这里分别测试了三种深度的网络（`1`、`2`、`4`），发现当卷积层只有一层时，其拟合能力尚可但学习能力不行，对陌生手写数字的分类效果相比其它两组要显著差上一截；另外两组在训练轮数达到`15`轮之后共同收敛到较高值

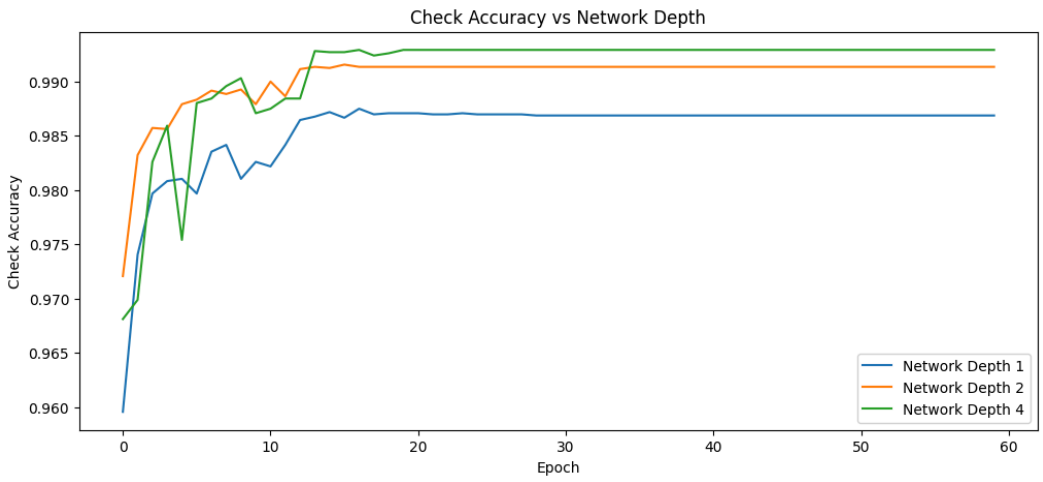
再看其损失曲线，与上面在验证集上的预测准确率曲线走向基本一致。不过注意到当深度为`4`时其损失下降的速度要比深度为`2`时更慢一些，曲线也更陡峭



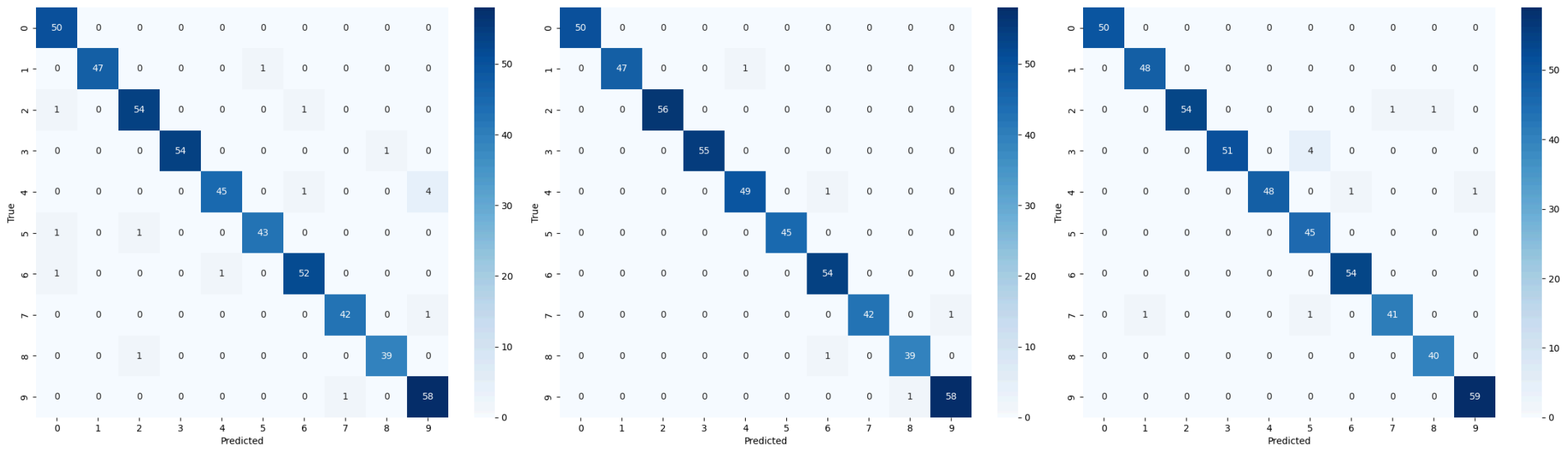




后续增大数据集（全量）与训练轮次发现 4 层卷积层的验证效果会更好一些，上面可能是数据量太少。另外，4 层卷积层在两种数据量的情况下验证准确率在前期的抖动都要更严重



下面的混淆矩阵热力图（由测试集得出）自左向右网络深度逐渐递增



通过对测试准确率的分析也能验证我们上面的观点，2 层与 4 层的准确率基本一致，相比较一层都更高

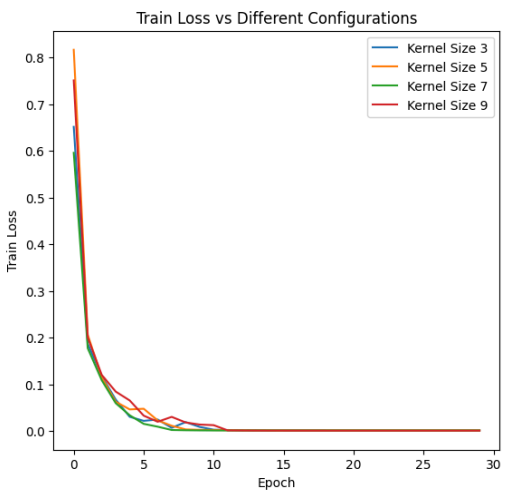
Total acc	0	1	2	3	4	5	6	7	8	9
0.9680	0.9434	1.0000	0.9643	1.0000	0.9783	0.9773	0.9630	0.9767	0.9750	0.9206
0.9900	1.0000	1.0000	1.0000	1.0000	0.9800	1.0000	0.9643	1.0000	0.9750	0.9831
0.9800	1.0000	0.9796	1.0000	1.0000	1.0000	0.9000	0.9818	0.9762	0.9756	0.9833

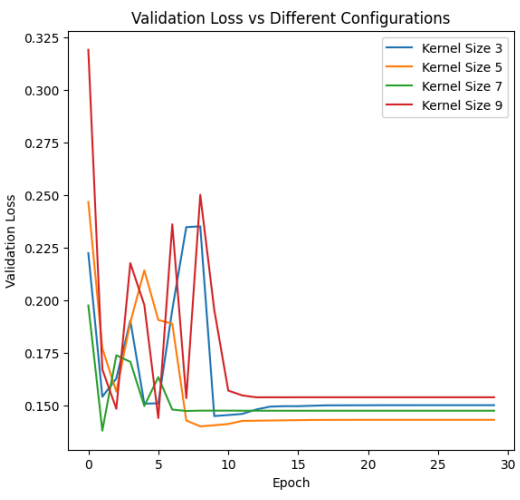
基本可以得出以下结论：

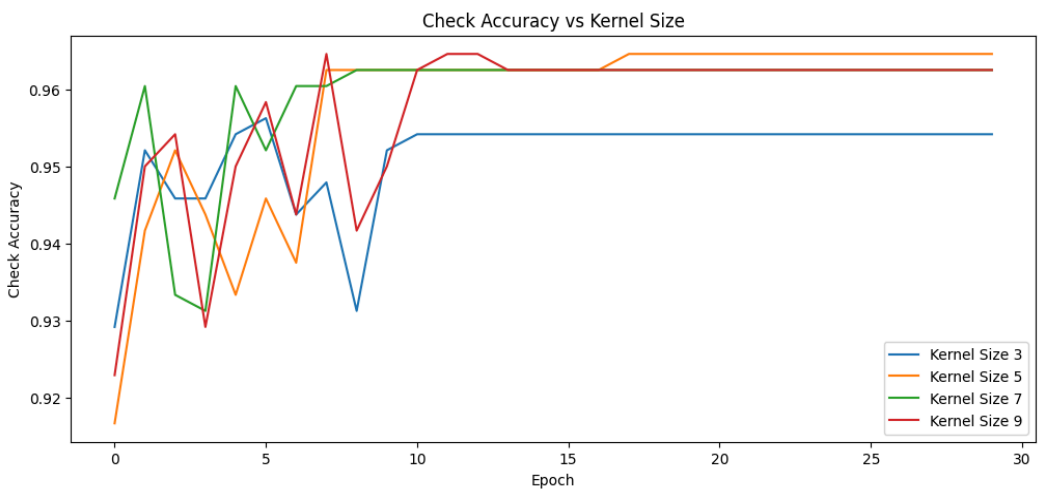
- 深层网络可以小幅度提升分类的准确率，未发现明显的过拟合现象；其需要更多训练时间和计算资源，但对 `MNIST` 的提升有限，性价比较低

卷积核大小影响

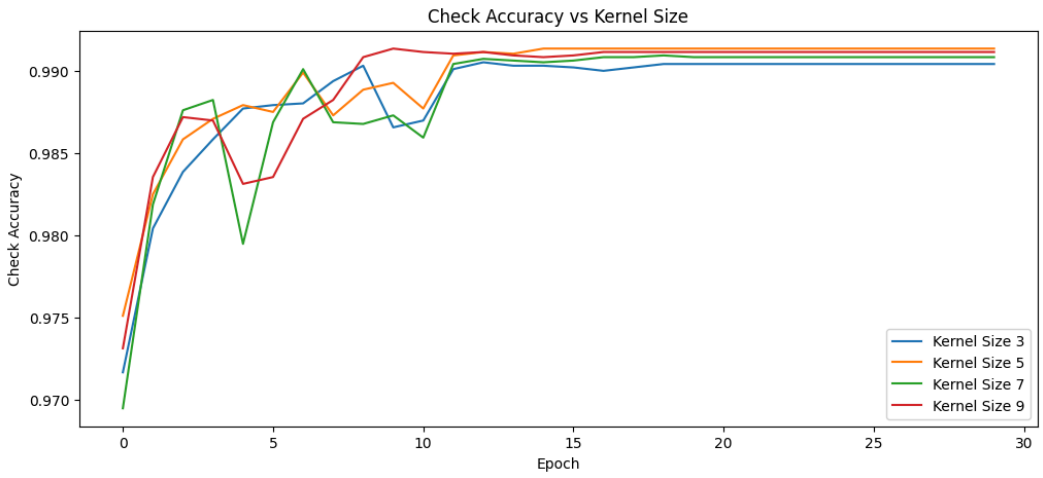
这里可以明显看出从 `3x3` 到 `5x5` 的效果有较大提升，继续增大卷积核在训练上的质量略有下降

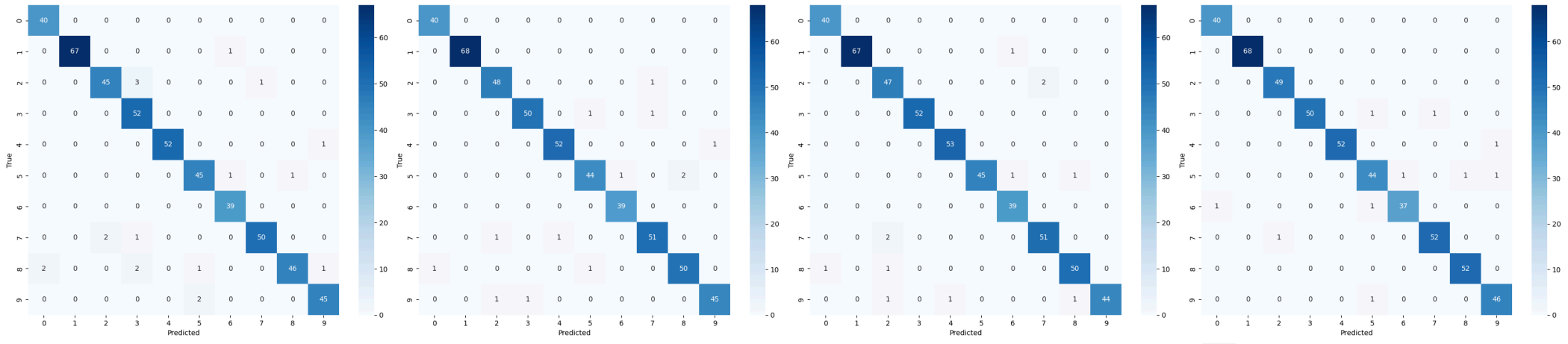






在全量数据集上得出的结果也基本一致，`5x5` 卷积核的效果同样更好



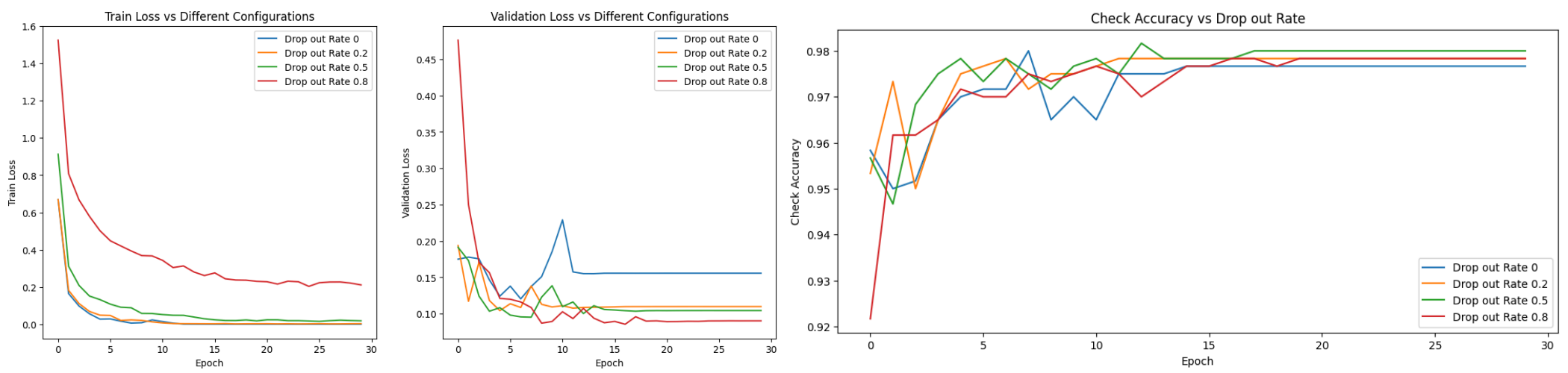


而在测试时实际上后三个卷积核大小带来的差别并不大，可以认为大卷积核能够提升分类准确率，但也应平衡训练效率与质量，在本次任务上选用 `5x5` 大小的卷积核可以兼顾训练效率与质量

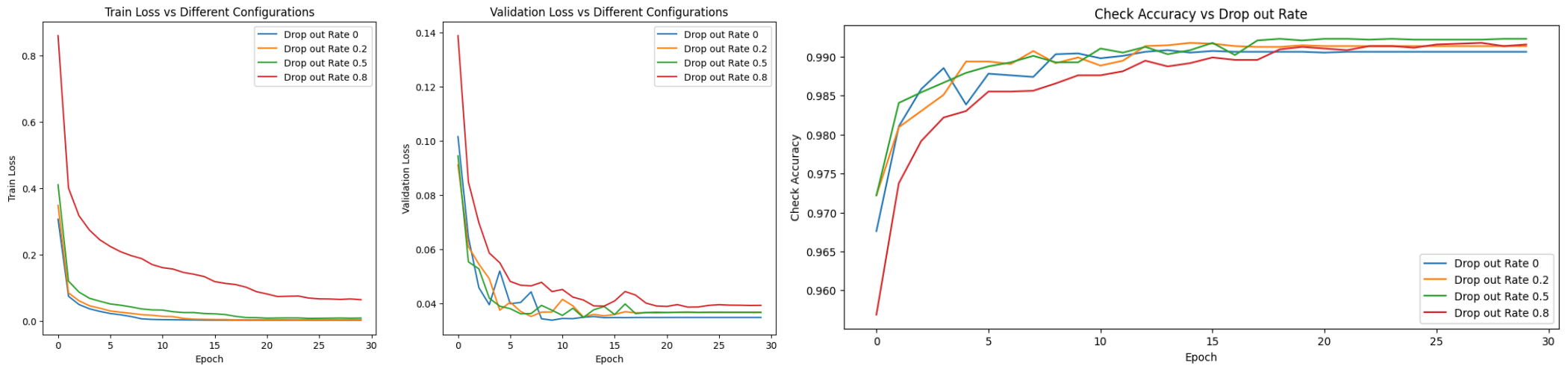
Test acc	0	1	2	3	4	5	6	7	8	9
0.9620	0.9524	1.0000	0.9574	0.8966	1.0000	0.9375	0.9512	0.9804	0.9787	0.9574
0.9740	0.9756	1.0000	0.9600	0.9804	0.9811	0.9565	0.9750	0.9623	0.9615	0.9783
0.9760	0.9756	1.0000	0.9216	1.0000	0.9815	1.0000	0.9512	0.9623	0.9615	1.0000
0.9800	0.9756	1.0000	0.9800	1.0000	1.0000	0.9362	0.9737	0.9811	0.9811	0.9583

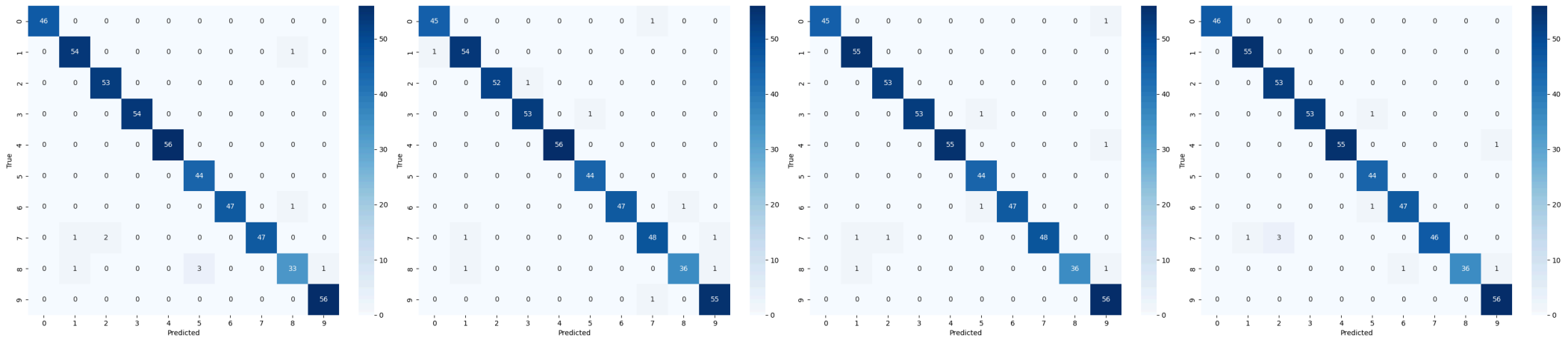
Dropout 的作用

在一定范围内提高时其有助于**提高**预测准确率（提升泛化能力与鲁棒性），但如果设置得太高，虽然其仍有一些提升，但相比其它值又略有不足，并且其初值也相对较低；同时，丢弃率太高使得模型能够使用的信息变得很少、无法学习到足够的特征，严重影响训练收敛速度，**会**导致模型训练不充分。不过由于此次的数据量比较小、任务也比较简单，同时训练保存的都是最佳模型，丢弃率在测试集上的准确率表现差别并不是特别明显（都使用各自的最佳模型进行预测）



对于全量数据集，上面的结论更为明显，高丢弃率情况下训练损失、验证损失的下降速度都明显地慢，验证准确率的上升速度也明显慢了一大截





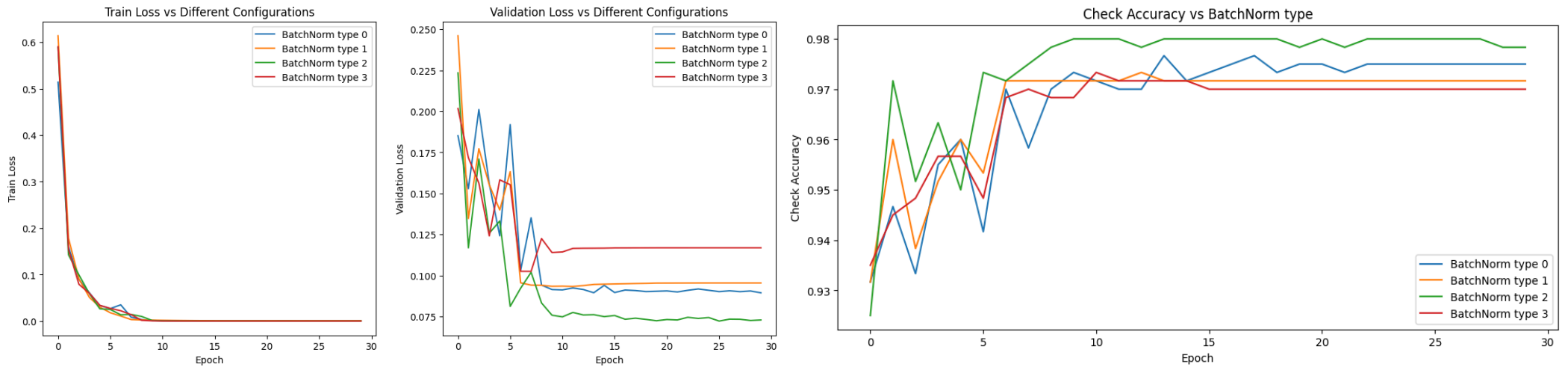
Test acc	0	1	2	3	4	5	6	7	8	9
0.9800	1.0000	0.9643	0.9636	1.0000	1.0000	0.9362	1.0000	1.0000	0.9429	0.9825
0.9800	0.9783	0.9643	1.0000	0.9815	1.0000	0.9778	1.0000	0.9600	0.9730	0.9649
0.9840	1.0000	0.9649	0.9815	1.0000	1.0000	0.9565	1.0000	1.0000	1.0000	0.9492
0.9820	1.0000	0.9821	0.9464	1.0000	1.0000	0.9565	0.9792	1.0000	1.0000	0.9655

BatchNorm 的影响

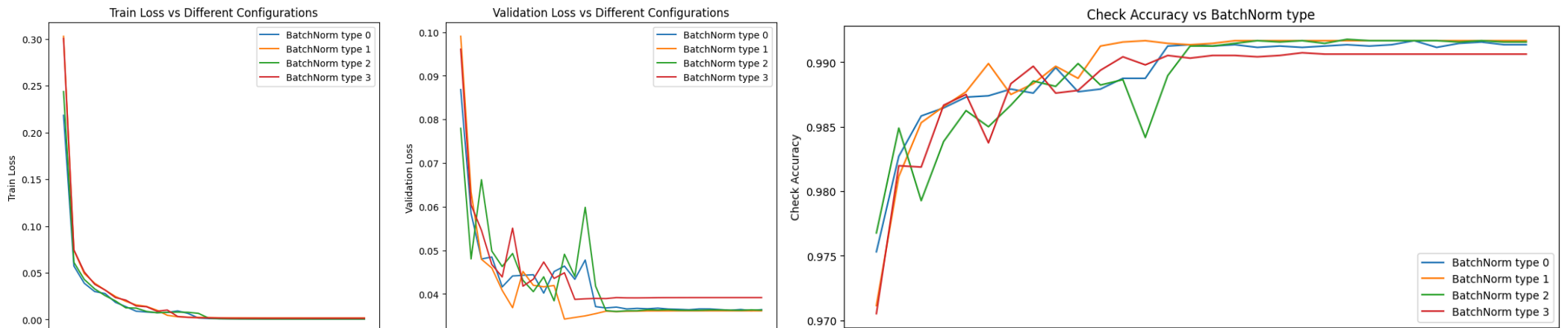
> 这里 BatchNorm type 为 0 和 2 的时候为添加了归一化层，另两种为同参数但无归一化的情况

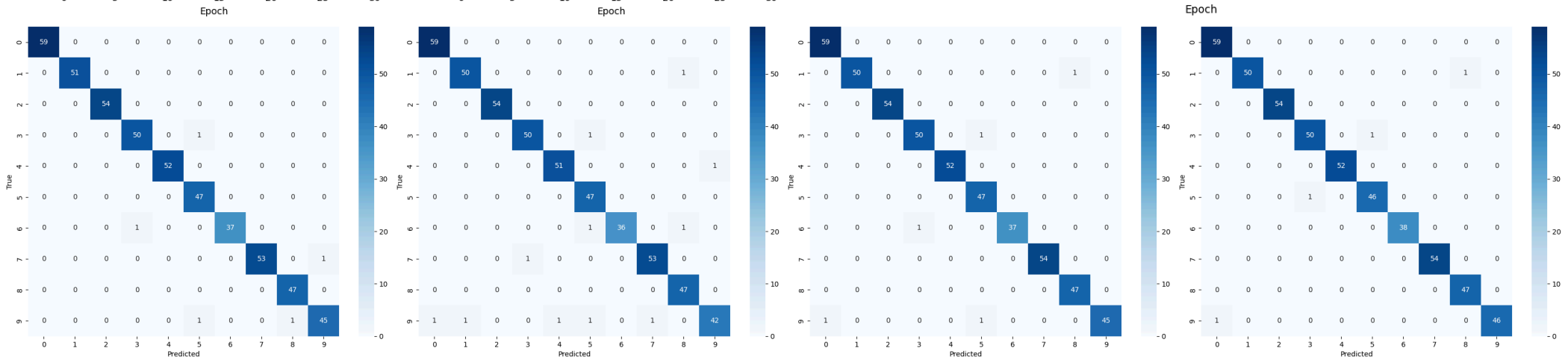
归一化层可以减少内部协变量偏移、提高梯度的稳定性、减少过拟合并提升泛化能力。由于前面的输入数据已经经过了标准化，这里归一化层的效果稍微减弱了一些，但仍能看出差距

可以看到，添加归一化层之后验证集分类准确率有显著上升，验证损失也明显更低，表明模型的泛化能力得到了增强



在全量数据集上，增加归一化层之后损失下降的速度更快了，能够加速收敛





Test acc	0	1	2	3	4	5	6	7	8	9
0.9900	1.0000	1.0000	1.0000	0.9804	1.0000	0.9592	1.0000	1.0000	0.9792	0.9783
0.9780	0.9833	0.9804	1.0000	0.9804	0.9808	0.9400	1.0000	0.9815	0.9592	0.9767
0.9900	0.9833	1.0000	1.0000	0.9804	1.0000	0.9592	1.0000	1.0000	0.9792	1.0000
0.9920	0.9833	1.0000	1.0000	0.9804	1.0000	0.9787	1.0000	1.0000	0.9792	1.0000