Project 1

姓名：陈春源 学号：20307140019 班级：人工智能（大数据班）

本次 Project 使用 python 完成，主要功能仅使用 numpy 等 python 基本库。

文件结构（主要内容）：

```
|  digits.mat
|  report.pdf
|
├─best_param.............................最佳参数
|      best_conv_model.pickle            CNN模型最佳参数
|      best_linear_model.pickle          线性模型最佳参数
|
├─handwrite.............................自制16x16手写数字图像
|      handwrite_0.png
|      handwrite_1.png
|      handwrite_2.png
|      handwrite_3.png
|      handwrite_4.png
|      handwrite_5.png
|      handwrite_6.png
|      handwrite_7.png
|      handwrite_8.png
|      handwrite_9.png
|
├─layer_func.............................layer及功能函数
|     AveragePooling.py
|     Conv2d.py
|     CrossEntropyLoss.py
|     gauss_noise.py
|     LinearLayer.py
|     Relu.py
|     Sigmoid.py
|     Softmax.py
|
├─model..................................模型
|     Classifier.py
|     Classifier_conv.py
|
├─runner.................................实验及模型训练
|      1_8.py
|      9.py
|      Conv2D_Model.py
|      handwrite_test.py.................训练最佳模型在自制图像上的测试
|      Linear_Model.py
```

具体功能：

[1.2.x] 为对应题目的实现，具体用法在“规定题目完成情况”板块中介绍。

Layer

线性层 LinearLayer [1.2.1]

实现了前向、后向过程

（可选参数）实现了偏置 b [1.2.6]；实现了 drop out [1.2.7]； （取代动量法）实现了 Adam 参数更新 [1.2.2]；实现了参数 W 的 L_2 正则化 [1.2.4]

激活函数 Relu

实现了 Relu 激活函数： $y = max(0, x)$ 的前向、后向过程

概率归一化 Softmax [1.2.5]

实现了归一化概率分布 Softmax： $p(y_i) = \frac{exp(z_i)}{\sum_{j=1}^{n} exp(z_j)}$ 的前向、后向过程，为了避免溢出做了平移处理： $z_i = z_i - max_n(z_j)$

交叉熵损失函数 CrossEntropyLoss [1.2.5]

实现了交叉熵损失函数 $Loss = \frac{1}{N} \sum_{i=1}^{n} -y_i log(\hat{y}_i)$ 的前向、后向过程

（可选参数）在训练时对可学习参数进行 Adam 参数更新；对线性层进行参数 W 的 L_2 正则化

2D卷积层 Conv2d [1.2.10]

实现了2D卷积的前向、后向过程；

（可选参数）实现了卷积核大小、步长、padding自定义；实现了多通道卷积，可自定义输入输出通道数；实现了 Adam 参数更新

均值池化层 AveragePooling

实现了均值池化前向、后向过程

（可选参数）实现了卷积核大小、步长

数据处理

批归一化

二值化处理

其他功能

批训练，向量化、矩阵化计算过程 [1.2.3]

提前停止训练过程 [1.2.4]

模型训练时可选 fine-tune 对最后一层（线性层）进行微调 [1.2.8]

为训练集加入高斯噪声，扩展训练集；通过验证集确定超参数后，将验证集合并入训练集进行训练 [1.2.9]

规定题目完成情况

实验代码见 runner/1_8.py 和 runner/9.py，由于实验过程中设计了频繁的模型参数修改，实验代码有些繁琐和省缺

1.2.1

在初始化线性模型时，通过 hidden_num 参数可以设置隐藏层神经元个数

```
class Classifier(object):
    def __init__(self, in_num, hidden_num, out_num, lr=0.001):
```

1.2.2

说明：由于实现了 Adam 参数学习策略，所以没有选择实现原始的 momentum 参数学习策略

在模型训练时，通过设置 adam=True 使用 Adam 参数学习策略（adam=False 使用 SGD 参数学习策略）

```
def train(self, batch_size, epoch_num, reg=False, lamda=0.01, if_eval=True, X_eval=None, y_eval=None, drop=True, shuffle=True, fine_tune=False, save_path=None, adam=True):
```

1.2.3

在模型训练时，通过设置 batch_size 参数可以启用批训练，训练时进行向量化、矩阵化计算

```
def train(self, batch_size, epoch_num, reg=False, lamda=0.01, if_eval=True, X_eval=None, y_eval=None, drop=True, shuffle=True, fine_tune=False, save_path=None, adam=True):
```

1.2.4

在模型训练时，通过设置 reg=True 启用 L_2 正则化；通过设置 lamda 参数设置正则化系数

```
def train(self, batch_size, epoch_num, reg=False, lamda=0.01, if_eval=True, X_eval=None, y_eval=None, drop=True, shuffle=True, fine_tune=False, save_path=None, adam=True):
```

通过手动修改模型中的代码可以实现提前停止

```
if i and self.eval_epoch_loss[i] >= self.eval_epoch_loss[i - 1]:  # 提前停止
    # break
    continue
```

1.2.5

Softmax.py 和 CrossEntropyLoss.py 实现了 Softmax 层和 CrossENtropyLoss 层，在模型构建中均使用了 Softmax + CrossEntropyLoss 的结构。

1.2.6

初始化线性层时，通过设置 bias=True 可启用偏置项

```
class LinearLayer(object):
    def __init__(self, in_num, out_num, bias=True, dropout=False, drop_prob=0.5, name=None):
```

1.2.7

初始化线性层时，通过设置 dropout=True 启用 drop out 功能；通过设置 drop_prob 参数设置 drop 概率

```
class LinearLayer(object):
    def __init__(self, in_num, out_num, bias=True, dropout=False, drop_prob=0.5, name=None):
```

1.2.8

在模型训练时，通过设置 fine_tune=True 启用 fine_tune，实现冻结前置层参数，仅对最后一层进行参数微调更新

```
def train(self, batch_size, epoch_num, reg=False, lamda=0.01, if_eval=True, X_eval=None, y_eval=None, drop=True, shuffle=True, fine_tune=False, save_path=None, adam=True):
```

1.2.9

通过 gauss_noise() 方法可以对训练集添加高斯噪声，实现数据集增广

```
def gauss_noise(X, y, mean=0, sigma=0.5):
    X_noise = X + np.random.normal(mean, sigma, X.shape)
    y_noise = y.copy()
    return X_noise, y_noise
```

1.2.10

Conv2d 实现了多通道卷积层的几乎全部功能

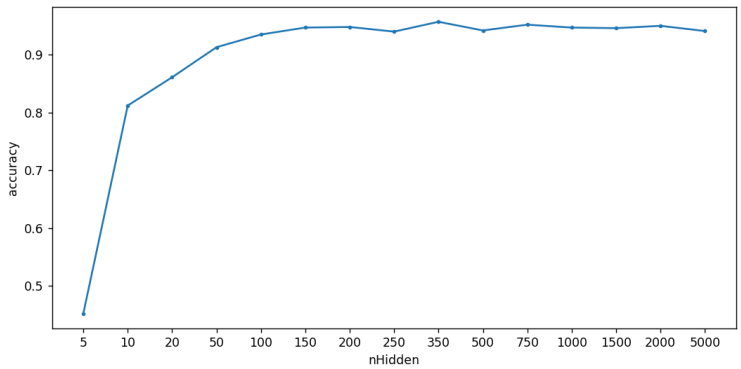
```
class Conv2d(object):
    def __init__(self, kernel_size=3, stride=1, in_chanel=1, out_chanel=3, padding=True, kernel=None):
```

具体实验（主要部分）：

未经特殊说明，均使用向量化、矩阵化运算和单隐藏层线性模型进行实验，每轮训练对训练集进行 shuffle。

1. 改变隐藏层神经元数量 [1.2.1]

控制训练批大小为 100，训练 3 轮，调节隐藏层神经元数量，观察测试集准确率。



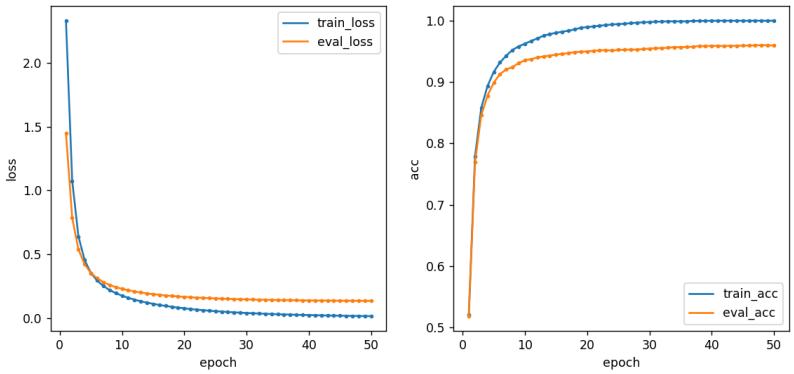
可以发现，在控制训练批大小和轮次数相同的情况下，随着隐藏层神经元数量增加，模型的预测准确率在提升；当隐藏层神经元数量达到一定值时，训练效果没有明显变化。这是因为隐藏层神经元数量的增加导致了模型参数的增加，更多的模型参数意味着更强的学习能力，但同时会带来计算的复杂度提升和模型的参数冗余（事实上可以通过参数更少的简单模型经过更多轮次的训练达到同样的效果）。

2. 利用动量法更新参数 [1.2.2]

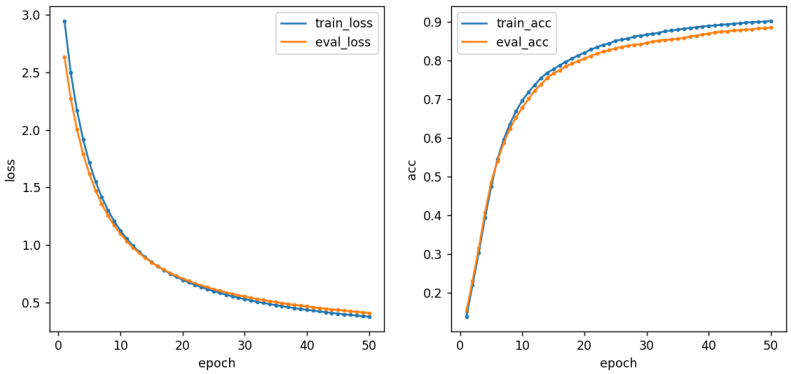
控制训练批大小为 500，训练 100 轮，分别使用 Adam 和 SGD 参数更新策略（学习率分别为 0.01 和 1.0），观察学习过程中 loss 曲线和观察测试集准确率。

此处的 SGD 方法是 mini-batch 参数更新。

Adam acc=0.963



SGD acc=0.867

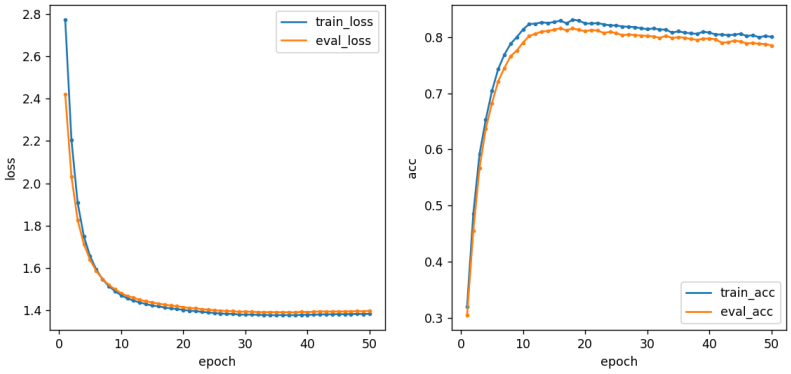


可以发现，在控制训练批大小、轮次数相同和学习率适当的情况下，由于引入了动量，Adam 可以比传统的 SGD 找到更优的参数更新方向和更新步长，其收敛速度更快，效果也优于 SGD 方法。

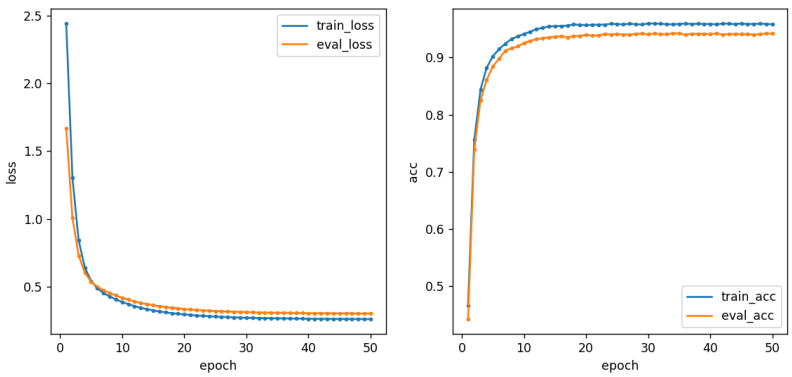
3. 正则化约束 [1.2.4]

控制训练批大小为 500，训练 100 轮，学习率 0.01，分别使用 L_2 正则化（lamda分别为 1.0，0.1和 0.01） 和不使用正则化，观察学习过程中 loss 曲线和观察测试集准确率。

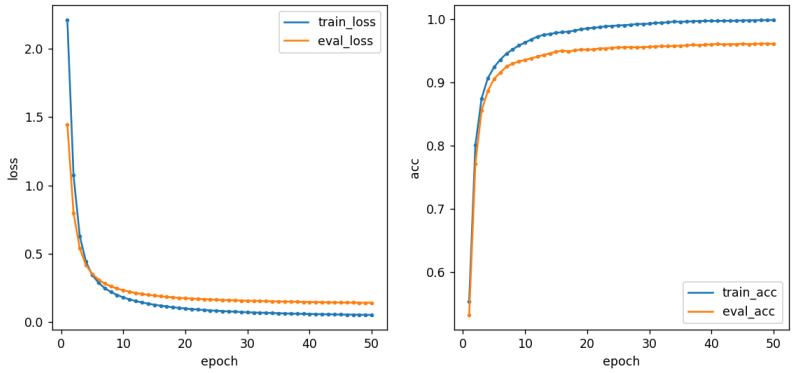
L2 regulation lamda=1.0 acc=0.813



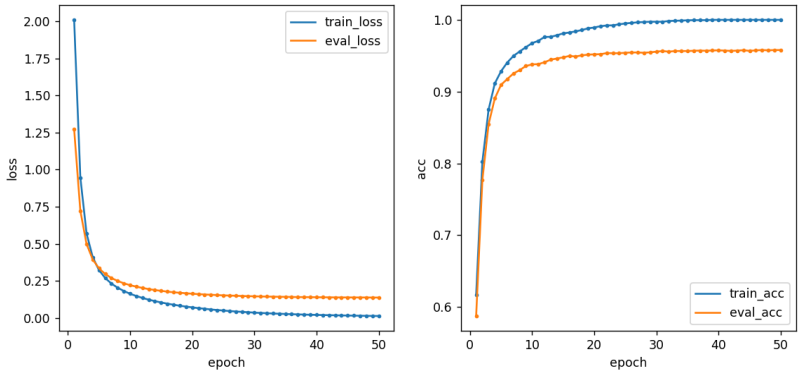
L2 regulation lamda=0.1 acc=0.948



L2 regulation lamda=0.01 acc=0.969



No regulation acc=0.963

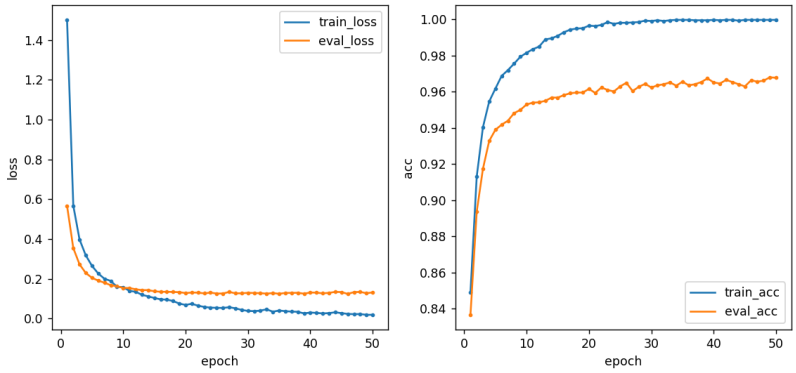


可以发现，正则化系数为 1.0 时，模型在训练后期出现过拟合，效果不如不使用正则化；而正则化系数为 0.1 和 0.01 时，模型的表现则与不使用正则化相当。正则化实际上约束了可学习参数的下降范围，过大的正则化系数可能会导致参数无法更新到较好的位置，因此需要谨慎选择正则化系数。实际上还可以通过梯度截断等方法控制参数的更新范围，由于时间原因没有选择实现。

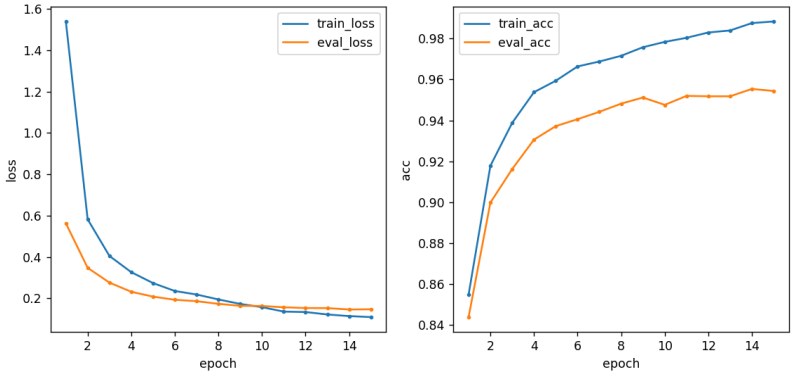
4. early stopping [1.2.4]

控制训练批大小为 100，训练 50 轮，学习率 0.01，分别不启用和启用 early stopping，观察学习过程中 loss 曲线和观察测试集准确率。

not early stop acc=0.971



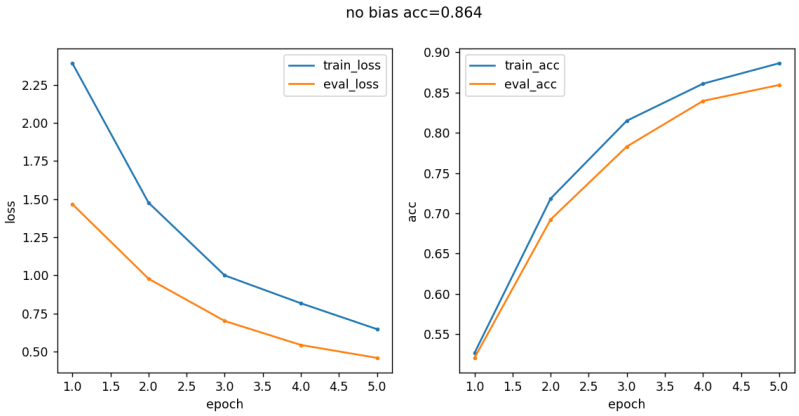
early stop acc=0.967

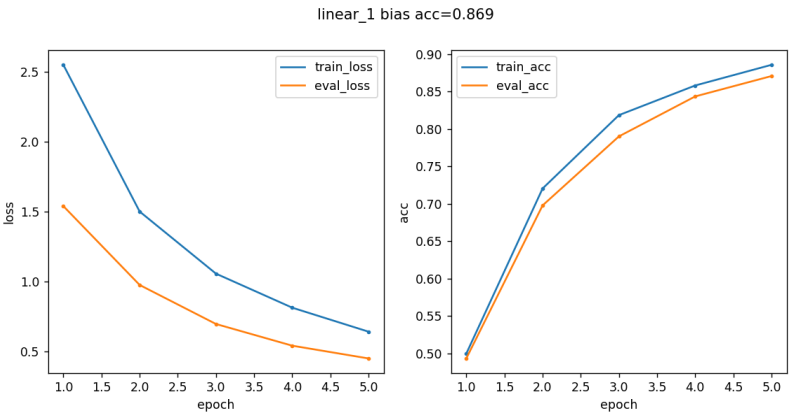


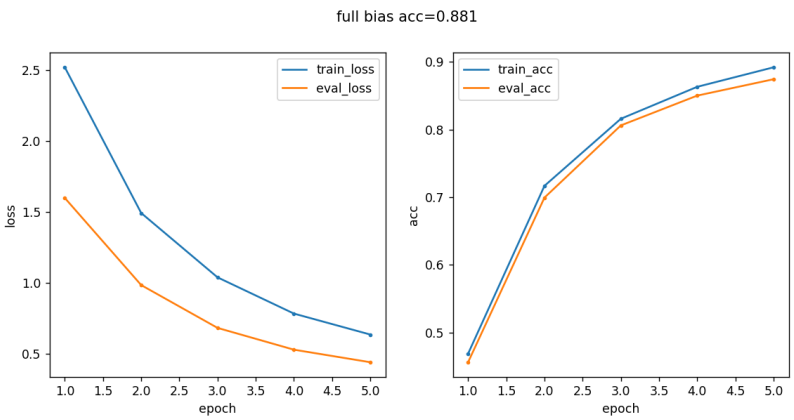
可以发现，在不启用 early stopping 时，模型完成了 50 轮训练；而启用 early stopping 时，模型仅进行了 15 轮训练便停止，但在测试集上的准确率略低于不启用 early stopping 的结果。实验结果表明，early stopping 可以有效减少无谓的大量训练（可能大量的训练轮次也无法使 loss 明显下降，并获得明显的准确率提升）从而节约时间和计算成本，且可以有效地防止模型过拟合；但有时也可能导致模型在不好的局部最优点时停止更新，难以获得更好的训练效果。

5. 添加偏置 [1.2.6]

控制训练批大小为 500，训练 5 轮，学习率 0.01，分别不加偏置，为第一层添加偏置，为所有层添加偏置，观察学习过程中 loss 曲线和观察测试集准确率。





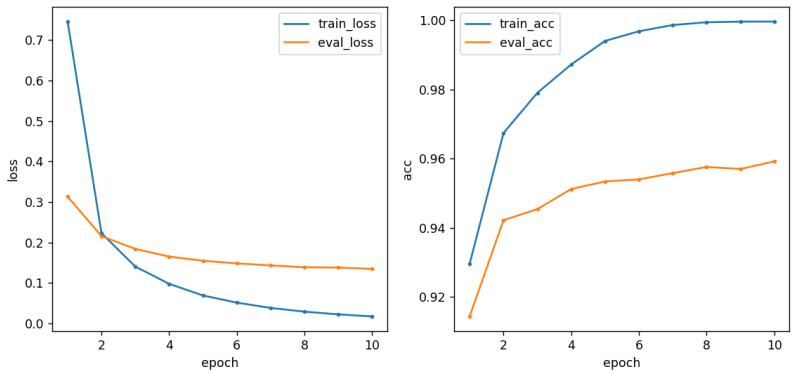


可以发现，添加偏置后，模型在测试集上的表现略有提升，说明为线性层添加偏置可以增强模型的学习能力，但由于该数据集和任务相对简单，不能体现得十分明显。偏置项可以使神经元更加灵活，加速网络拟合。

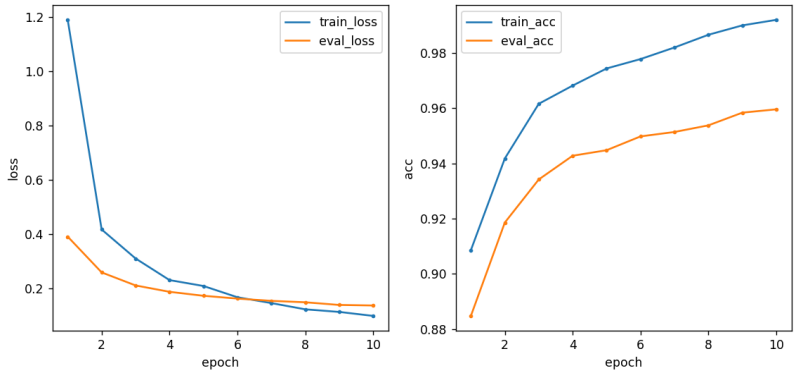
6. drop out [1.2.7]

控制训练批大小为 500，训练 100 轮，学习率 0.01，分别使用 drop out（drop_prob 分别为 0.2，0.5 和 0.8） 和不使用 drop out，观察学习过程中 loss 曲线和观察测试集准确率。

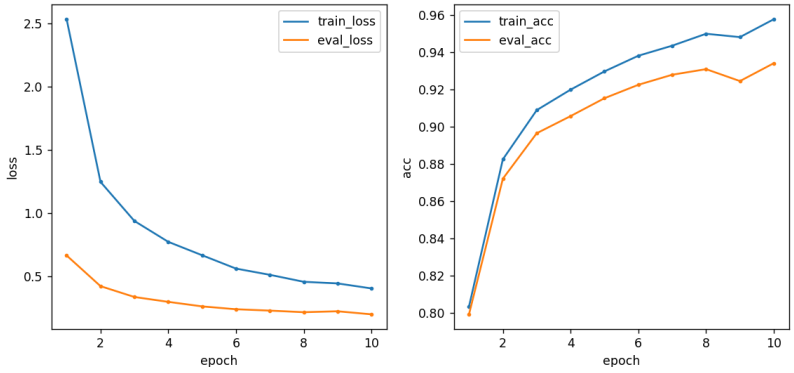
drop_prob = 0.0 acc = 0.967



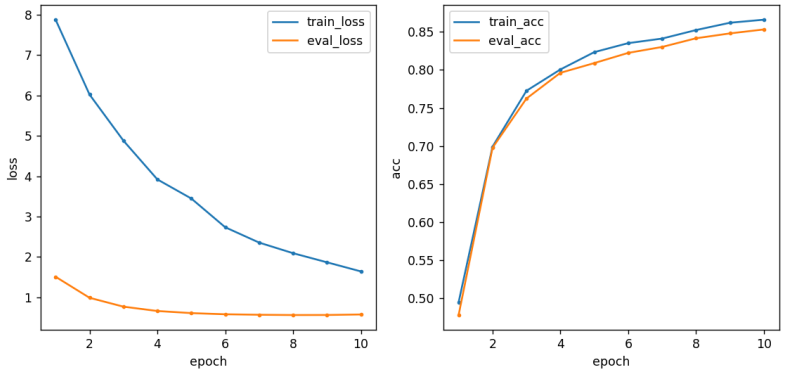
drop_prob = 0.2 acc = 0.959



drop_prob = 0.5 acc = 0.944



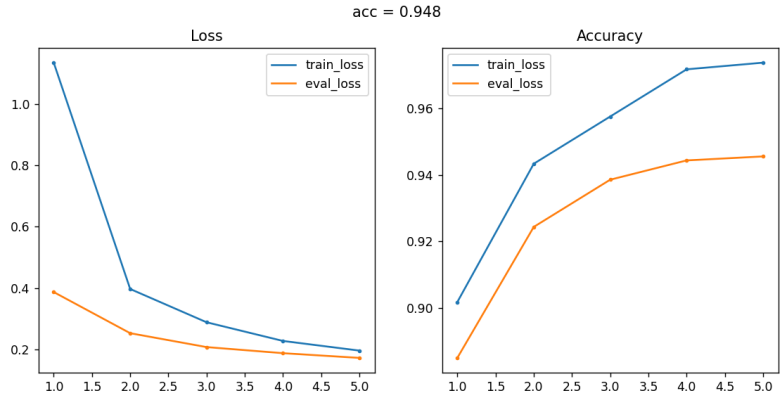
drop_prob = 0.8 acc = 0.841



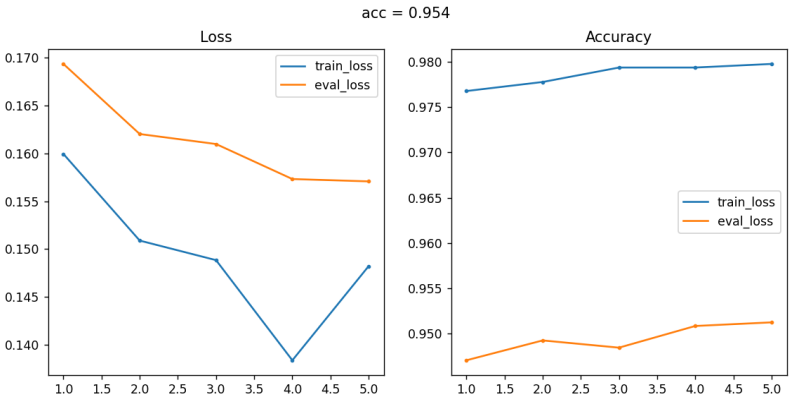
可以发现，当 drop_prob 为 0.2 和 0.5 时，模型的表现则与不使用正则化相当；而当 drop_prob 为 0.8 时，模型能力大幅下降，这是由于在训练时丢掉的权重过多，模型学习能力大幅下降。适当选取 drop_prob 可以增强模型的泛化性。

7. fine-tune [1.2.8]

使用线性模型设置 batch_size=50，epoch=5 进行与训练。



在预训练基础上再进行 batch_size=50，epoch=5 的 fine-tune。

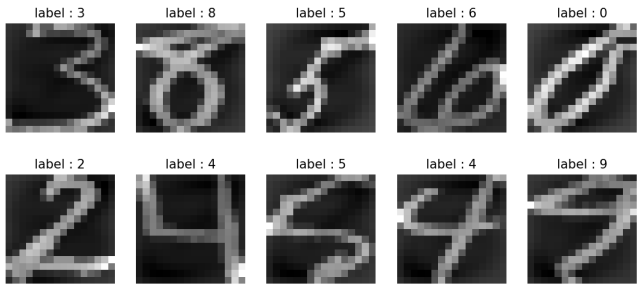


可以发现，经过 fine-tune 后，模型效果进一步增强，在测试集上的表现进一步提升。

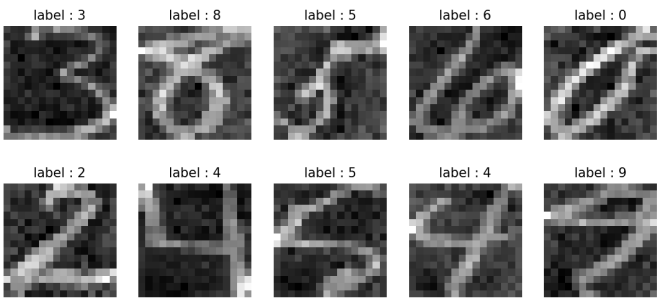
8. 增加训练集 [1.2.9]

为归一化后的图像叠加高斯噪声。

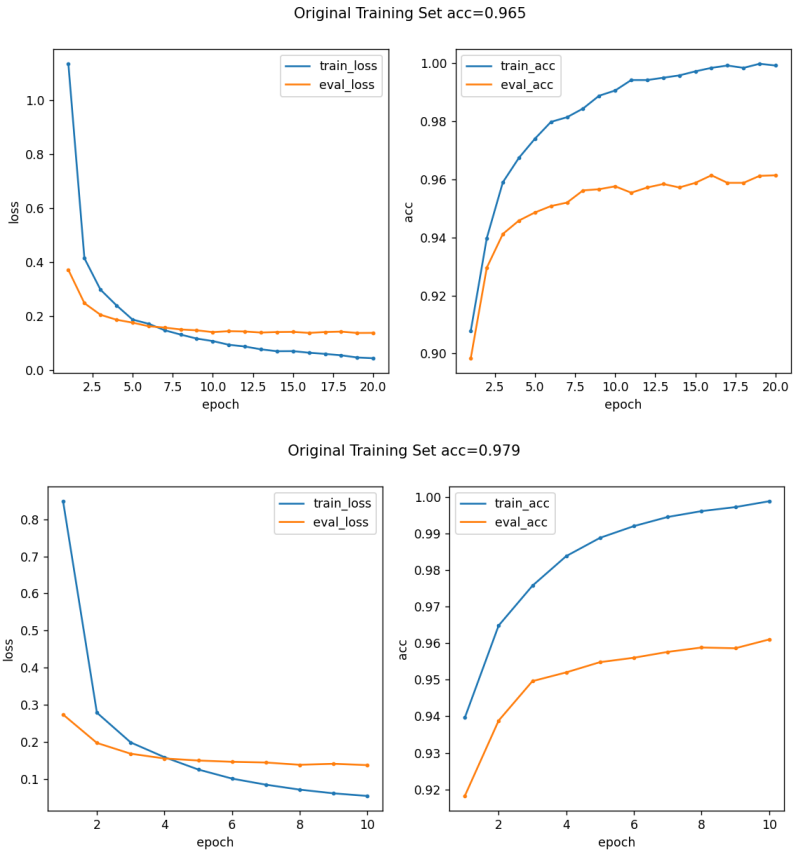
原始训练集图像：



加入高斯噪声后的训练集图像：



为了控制总训练 batch 数相同，设置原始训练集训练参数 batch_size = 50，epoch = 20；加噪声训练集训练参数 batch_size = 50，epoch = 10

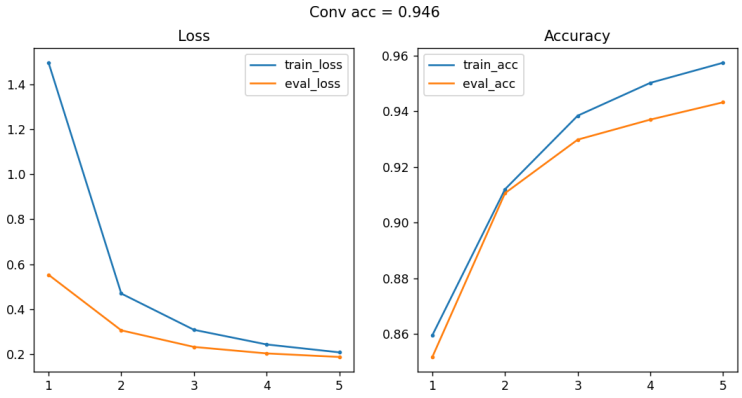


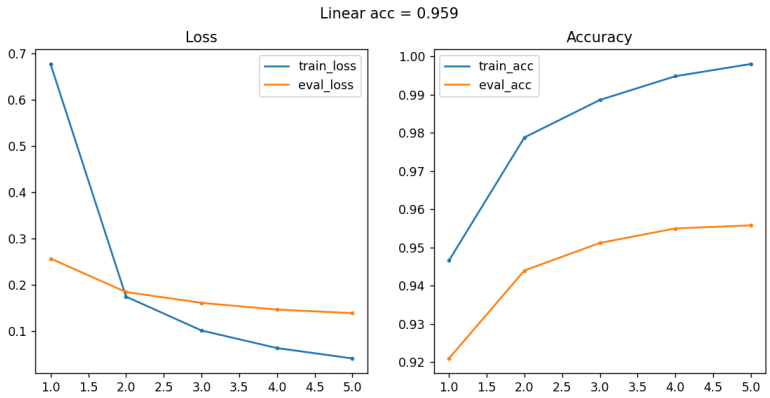
可以发现，加入噪声后的训练集的模型在测试集上表现更好，说明噪声图像的加入增加了模型的泛化能力，可以适当抵抗一些干扰。

由于数据集中的图片较小（16 x 16），且目标为数字，占据了相对满的图像区域，因此旋转、随机裁剪等传统方法不太适用于数据集扩展。

9. 使用卷积 [1.2.10]

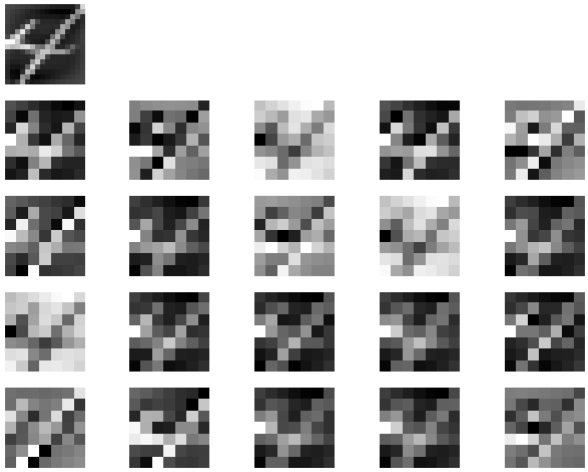
将线性模型的第一层改为多通道卷积，控制 batch_size = 25，epoch = 5，lr = 0.01，观察学习过程中 loss 曲线和观察测试集准确率。





卷积的效果和线性模型相当。

查看通过卷积提取出的 20 个特征：（左上为原图）



发现卷积层可以从输入图像中有效地提取出一些轮廓、形状、颜色信息，用于后续的分类任务。

7. 其他说明

对于 [1.2.3]：由于 python 实现的全部操作均基于向量化、矩阵化计算，故不做对比展示，仅做分析。batch 更新能够使模型能够在固定的时间内进行更多训练，且合适的 batch_size 可以使模型一次接受更多输入，学习到更广泛的特征，增强模型的泛化性和学习稳定性。

对于 [1.2.5]：由于 python 实现的模型全部基于 Softmax + CrossEntropyLoss 结构，没有实现 MSELoss（因为知道效果不好，且多分类模型几乎均采用 Softmax + CrossEntropyLoss 结构），不便进行对比展示，仅做分析。如果只使用 MSELoss，由于没有进行 Softmax 概率分布归一化，会导致 Loss 很大，且参数学习困难；使用 Softmax + CrossEntropyLoss 结构能够将概率分布归一化，且比 MSELoss 更好地计算两个概率分布之间的距离，更利于参数学习。

模型选择

尝试了不同层的模型组网（包括 layer 选择，激活函数（Relu、Sigmoid）选择等）

尝试了不同的数据处理手段：批归一化、二值化处理等

尝试了不同的数据增广手段：为图像添加高斯噪声、重新划分训练集和验证集

尝试了不同的防止过拟合手段：正则化、drop out、early stop、交叉验证

最终选择了以下两个模型

线性模型

模型选择：

Linear(256 --> 128, bias, drop_prob=0.2) ==> Relu ==> Linear(128 --> 10, bias, drop_prob=0.2) ==> Softmax ==> CrossEntropyLoss

训练集选择：

为训练集 (5000) 和验证集 (5000) 增加高斯噪声 (sigma=0.4) 合并后，选取验证集中的 20% (2000) 作为新的验证集，其余部分 (18000) 作为新的训练集投入训练

训练过程：

batch_size = 50, epoch = 20, lr = 0.01, **reg**=True, lamda=0.001

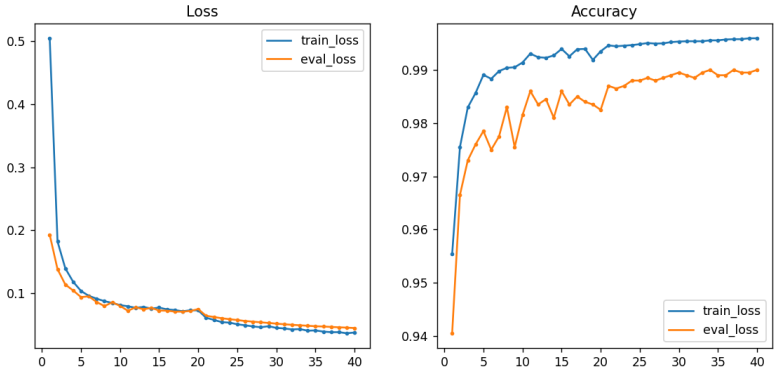
batch_size = 50, epoch = 20, lr = 0.001, **fine_tune**=True

训练过程中保存 eval_loss 最小的参数作为最佳模型参数

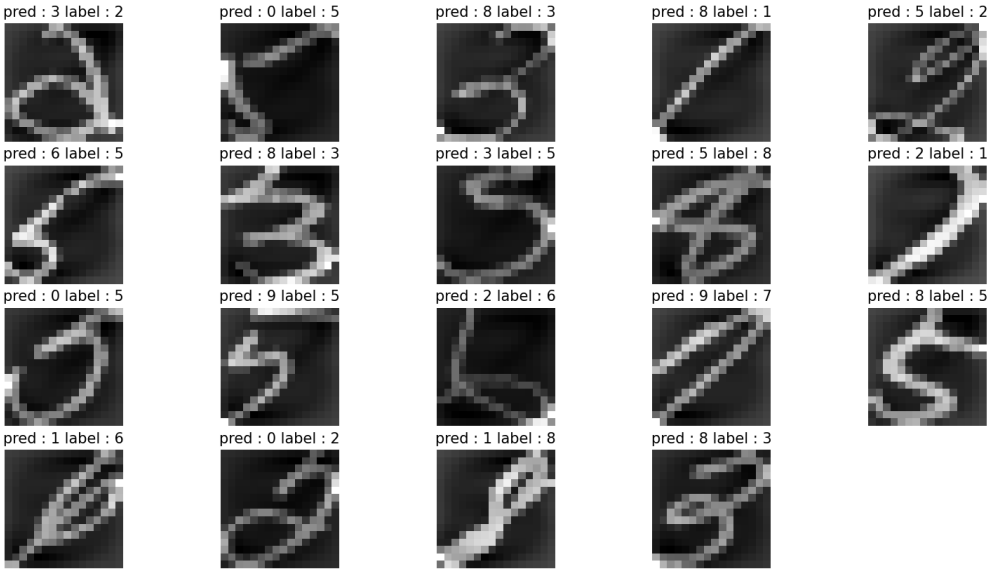
模型结果：

在测试集上最高正确率：981 / 1000

训练过程 Loss 和 Accuracy 曲线：



错误分类的所有图片：



最佳模型参数保存地址：

best_param/best_linear_model.pickle

复现方法：

运行 runner/Linear_Model.py

分析：

> 对数据集进行了增广和重新划分。
> 第一次训练进行正则化约束，第二次训练进行 `fine-tune`，两次训练均使用 `drop oup`，增强了模型的泛化能力。
> 可以发现在测试集上预测错误的图像确实不易区分，对线性模型来说难以识别，譬如上图中2行1列,3行1列,4行3列的图像，人类也不好轻易识别。
> 由于该任务比较简单，线性模型在该任务上能达到较好的表现。

CNN 模型

模型选择：

Conv2D(1x16x16 --> 10x16x16, kernel_size=3, stride=1, padding=True) ==> Relu

==> AveragePooling(10x16x16 --> 10x8x8, kernel_size=2, stride=2) ==> Linear(640 --> 256, drop_prob=0.2)

==> Linear(256 --> 10, bias) ==> Softmax ==> CrossEntropyLoss

训练集选择：

为训练集 (5000) 和验证集 (5000) 增加高斯噪声 (sigma=0.4) 合并后，选取验证集中的 20% (2000) 作为新的验证集，其余部分 (18000) 作为新的训练集投入训练

训练过程：

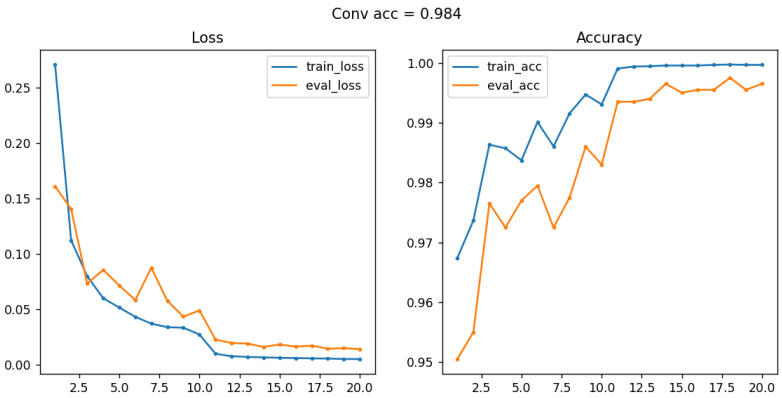
batch_size = 25, epoch = 20, lr = 0.01

batch_size = 25, epoch = 20, lr = 0.001

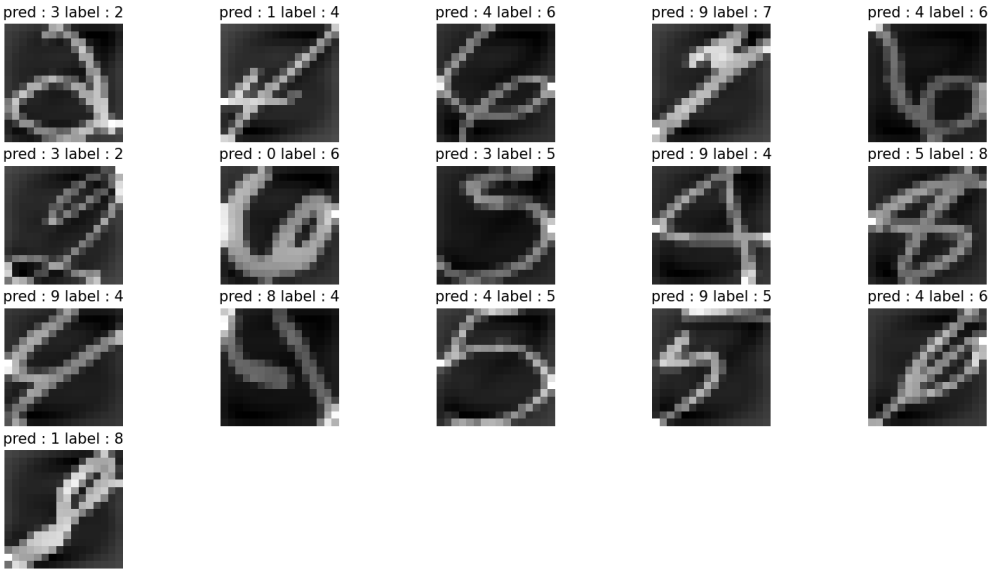
模型结果

在测试集上最高正确率：984 / 1000

训练过程 Loss 和 Accuracy 曲线：



错误分类的所有图片：



最佳模型参数保存地址：

best_param/best_conv_model.pickle

复现方法：

运行 runner/Conv2D_Model.py

分析：

对数据集进行了增广和重新划分。
使用了均值池化等手段，增强了模型的泛化性；调整了学习率，使模型能充分得到训练。
可以发现在测试集上预测错误的图像在形态上有一些混淆特征，不易被卷积网络识别。
卷积模型能够更好的提取图像的信息，比线性模型能力更强，表现更好，在复杂的图像分类任务上更能体现优势。