Cifar-10 with MLP and CNN Report

展子航 计21 2022010745

基础要求

Explain how `self.training` work. Why should training and testing be different?

- 在我的模块实现中，`self.training` 标识了模型是否处于被训练的状态。被神经网络的 `train` 置为 `true`，被 `eval` 置为 `false`，在训练过程中：
  - `self.training` 影响了 `BatchNorm` 行为的分布特征（mean 和 var）更新的计算。
  - `self.training` 影响了是否执行 `DropOut` （测试时不能丢特征）
- 训练和测试的本质任务是不同的：训练是调整模型本身的推理过程；测试是对模型的分类表现进行评价。因此两者自然需要不同。

Initial Experiences

我的两种实验使用的参数组如下（一共是 27 种组合）：

```
batch_sizes = [64, 128, 256]
learning_rates = [1e-2, 1e-3, 1e-4]
drop_rates = [0.3, 0.5, 0.7]
num_epochs = 50

input_dim = 3072
hidden_dim = 1024
class_num = 10

self.sequence_model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            BatchNorm1d(hidden_dim),
            nn.ReLU(),
            Dropout(drop_rate),
            nn.Linear(hidden_dim, class_num)
        )
```

MLP

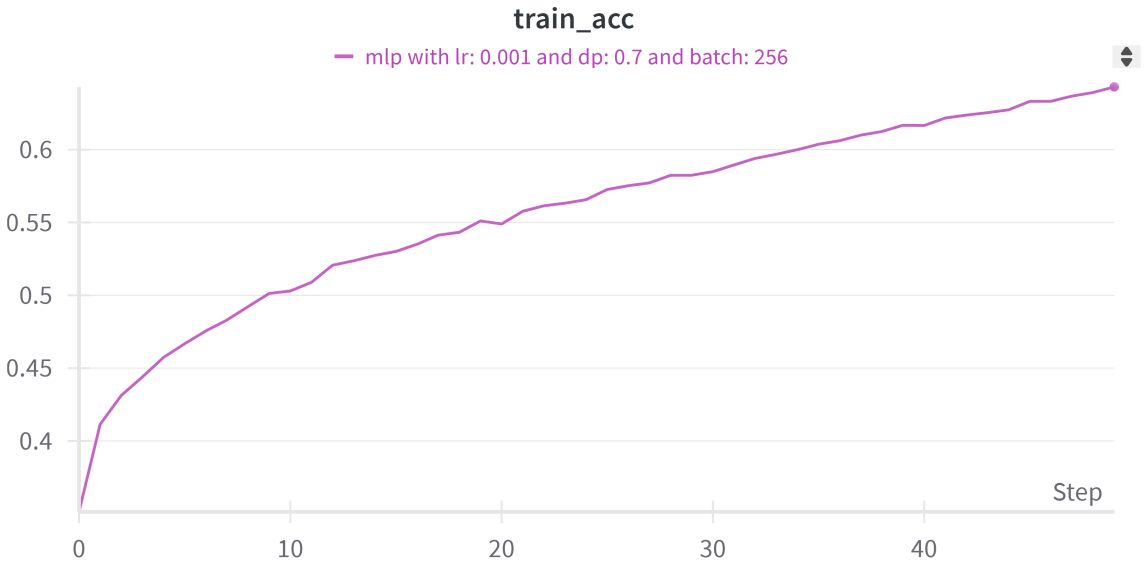
在 `train` 和 `valid` 表现最好的参数组为：

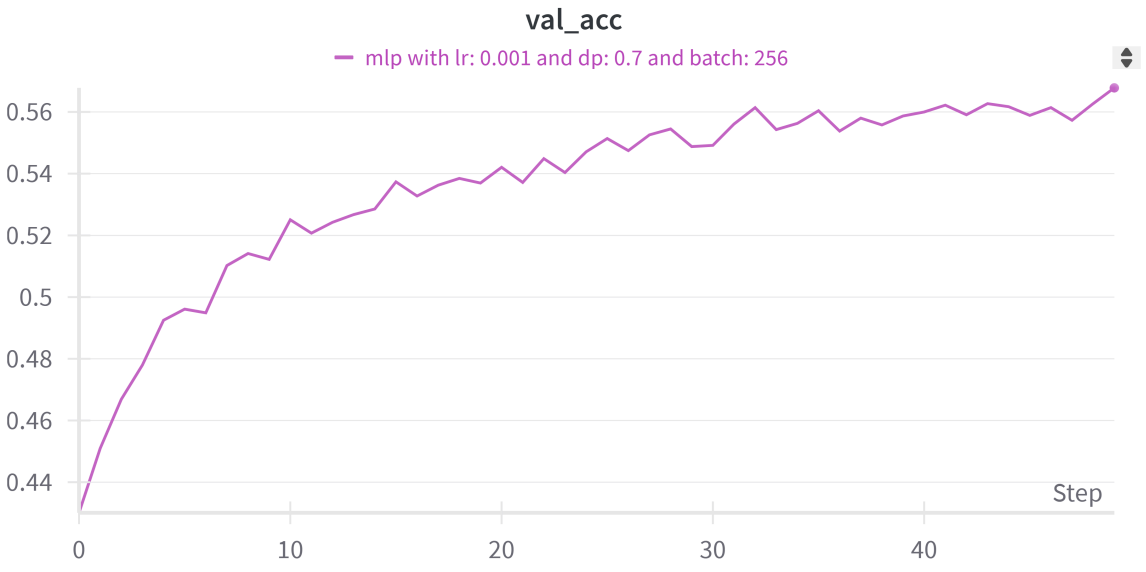
```
batch_sizes = 256
learning_rates = 1e-3
drop_rates = 0.7
num_epochs = 50
```

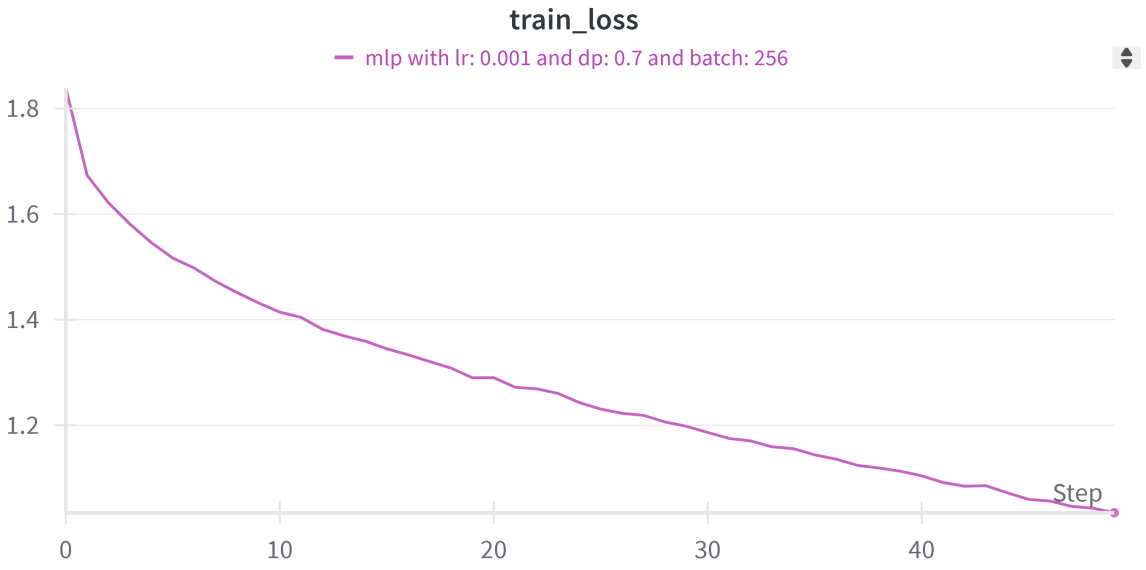
结果如下：

```
train_acc = 0.64308
val_acc = 0.56781
train_loss = 1.03412
val_loss = 1.32162
```

图样如下：







CNN

- 在 `train` 和 `valid` 表现最好的参数组为：

```
batch_sizes = 256
learning_rates = 1e-3
drop_rates = 0.3
num_epochs = 50

convblock(in_channel, out channel) = [
            nn.Conv2d
            (in_channels=in_channel,
             out_channels=out_channel,
             kernel_size=3,
             stride=1,
             padding=1),
            BatchNorm2d(out_channel),
            nn.ReLU(),
            Dropout(drop_rate),
            nn.MaxPool2d(kernel_size=2, stride=2),
        ]
```

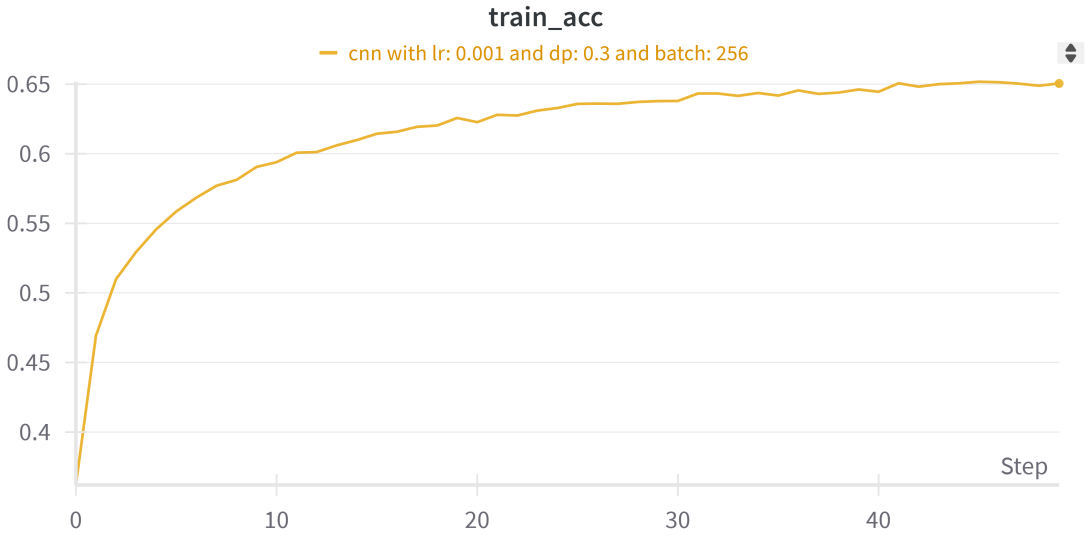
```
class_num = 10

cnn.sequential = {
    convblock(3, 16)
    convblock(16, 32)
    nn.Flatten(),
    nn.Linear(32 * 8 * 8, class_num)
}
```

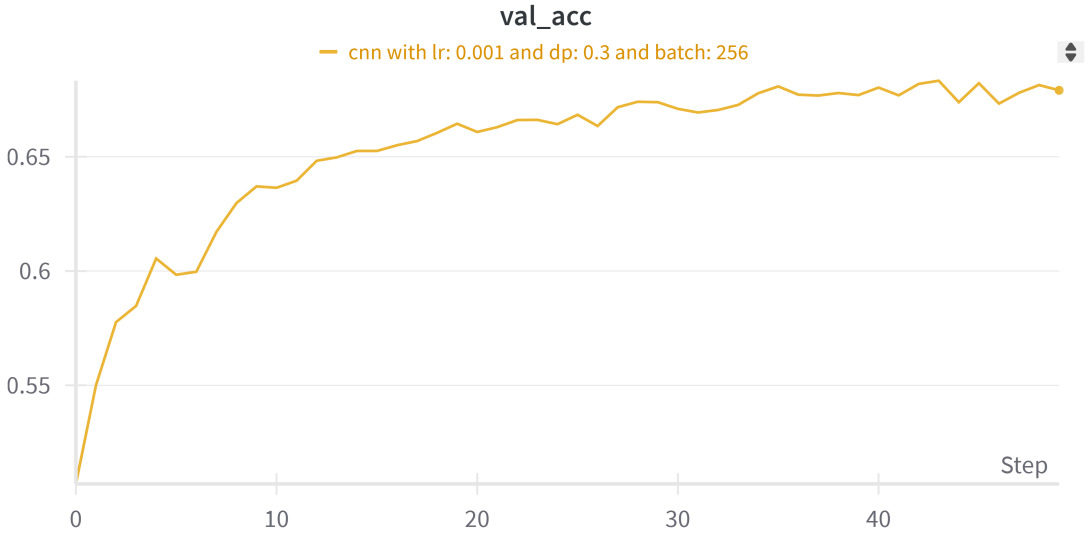
结果如下：

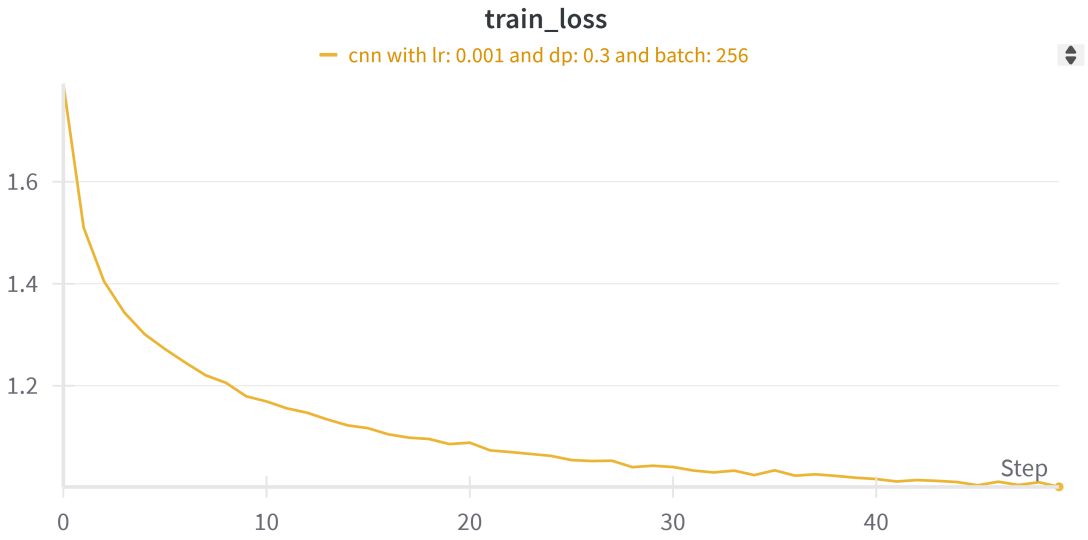
```
train_acc = 0.68219
val_acc = 0.65067
train_loss = 1.00157
val_loss = 0.93607
```

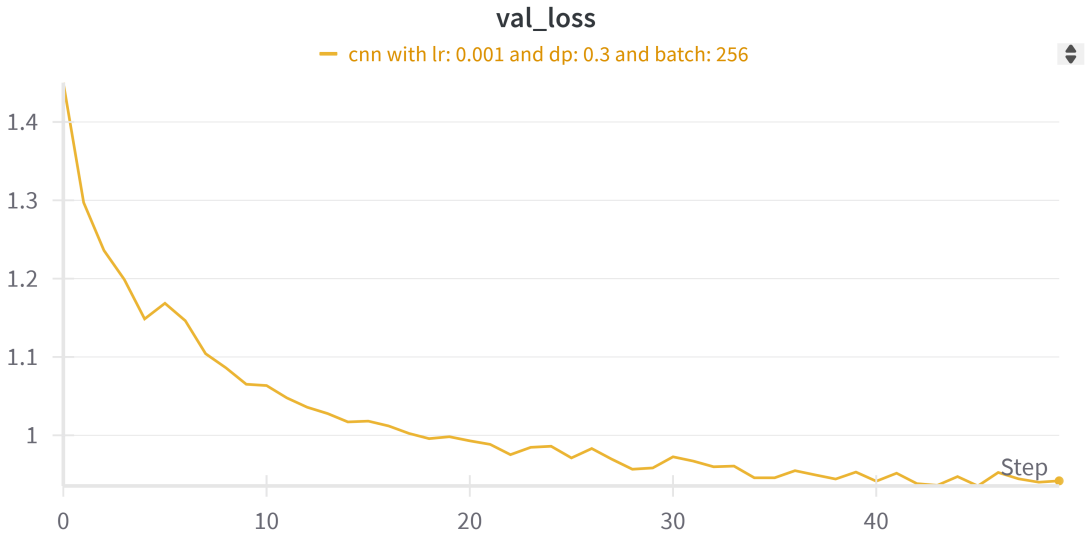
可以看出 CNN 效果明显好很多。

图样如下：









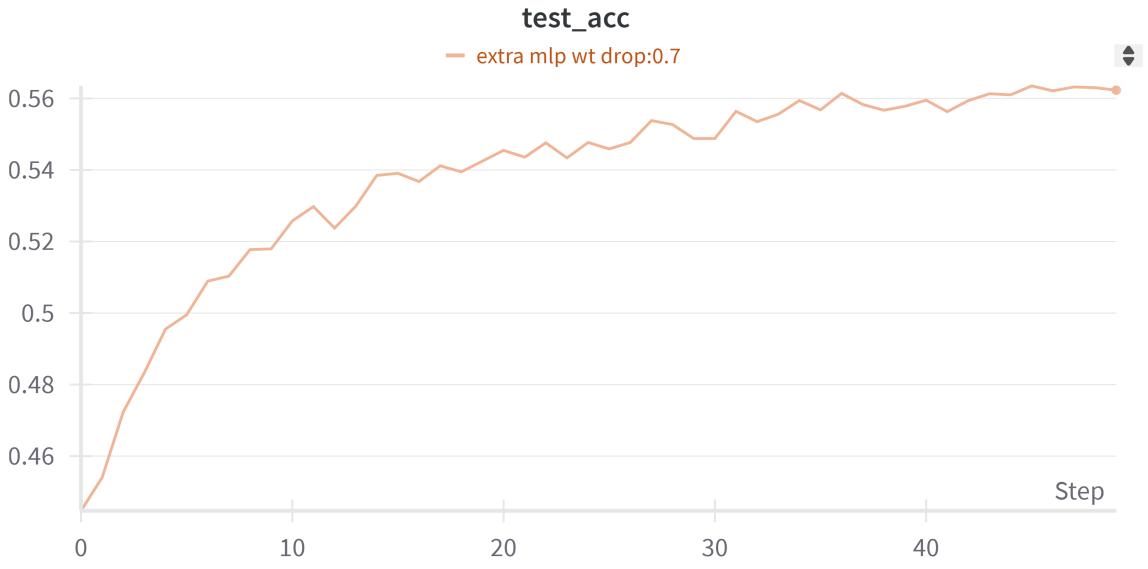
Explain why training loss and validation loss are different. How does the difference help you tuning hyper-parameters?

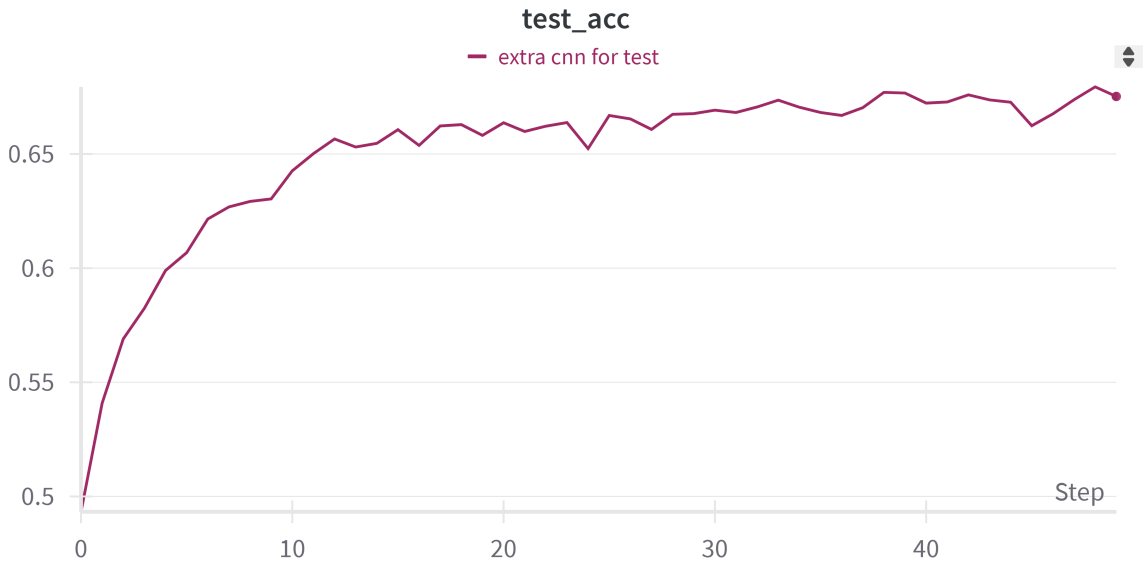
1. 二者的数据集本身就不同，算出来的loss不一样也很正常。
    - 前期 train loss 可能会稍高一些，这与计算过程相关。因为其算的是整个训练的平均值，而 valid loss 是一轮最后算的。
    - 后期 valid loss 会稍高一些，这是因为**模型训练时会根据 train 的方向进行训练**，而其中有些特有的特征与 valid 相似度并不高（也即是过拟合）。
2. 二者的差距可以从以下方面帮助模型超参数调整：
    - overfit：如果 valid loss 显著高于 train loss，则出现了严重的过拟合，此时考虑调高 `dropout` 或者降低 `num_epochs` 等方式降低过拟合。
    - underfit: 如果 valid loss 显著低于 train loss，则可能 train 不完全，此时是欠拟合状态，此时考虑调低 `dropout` （个人推荐）或者提升 `num_epochs` （我不推荐，效率太低）等方式修正。

Report the final accuracy for testing. Compare the performance of MLP and CNN and briefly explain the reason of the different performance.

结果如下：

```
mlp_test_acc = 0.5623
cnn_test_acc = 0.6767
```





显然，CNN 的效果要优于 MLP，这还是只训练了 50 epoch 的情况下。原因如下：

- CNN 的二维卷积运算能关注空间上相近的点的特征提取（这是计算方式决定的）；
- CNN 的池化操作更能提取出部分空间的相对特征；
- 反观 MLP 的一维像素运算，由于其运算是 Linear 延展的，会丢失第 2 维空间上的空间相似度，自然效果也会更差。

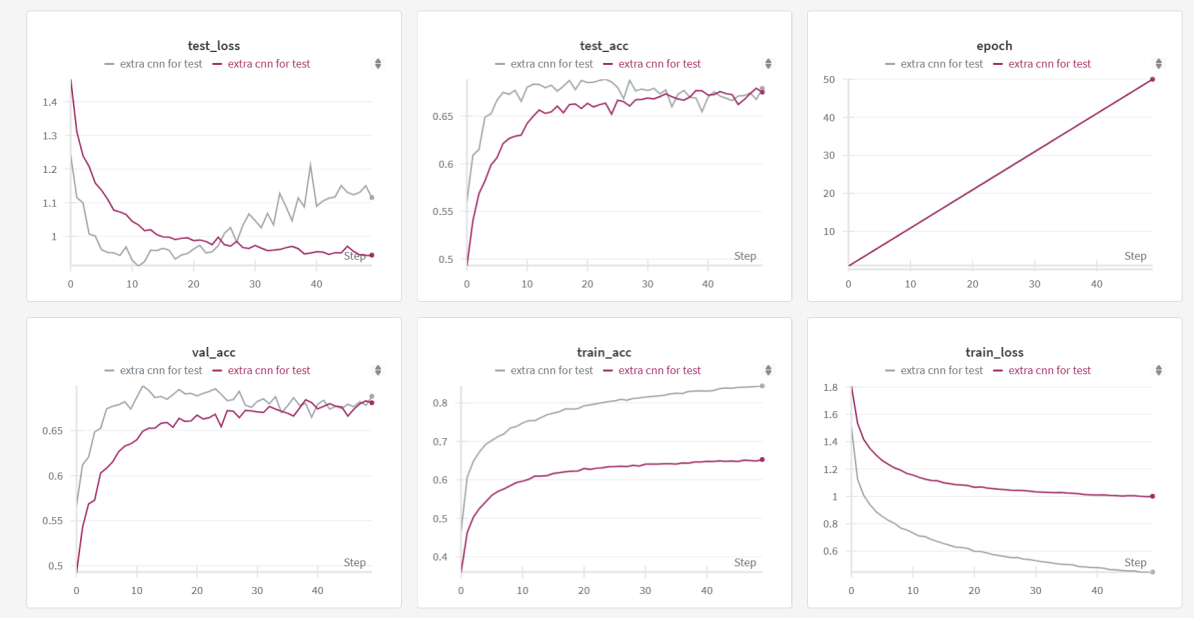
Construct MLP and CNN without dropout, and discuss the effects of dropout.

我使用了最佳表现实验的参数作为对比。

MLP

可以看到在没有 DropOut 的情况下过拟合极其严重，test_acc 差距巨大。

CNN



在没有 DropOut 的情况下，虽然从 loss 能够发现过拟合的情况，但是 test 的效果却要略高一些。

DropOut 的影响

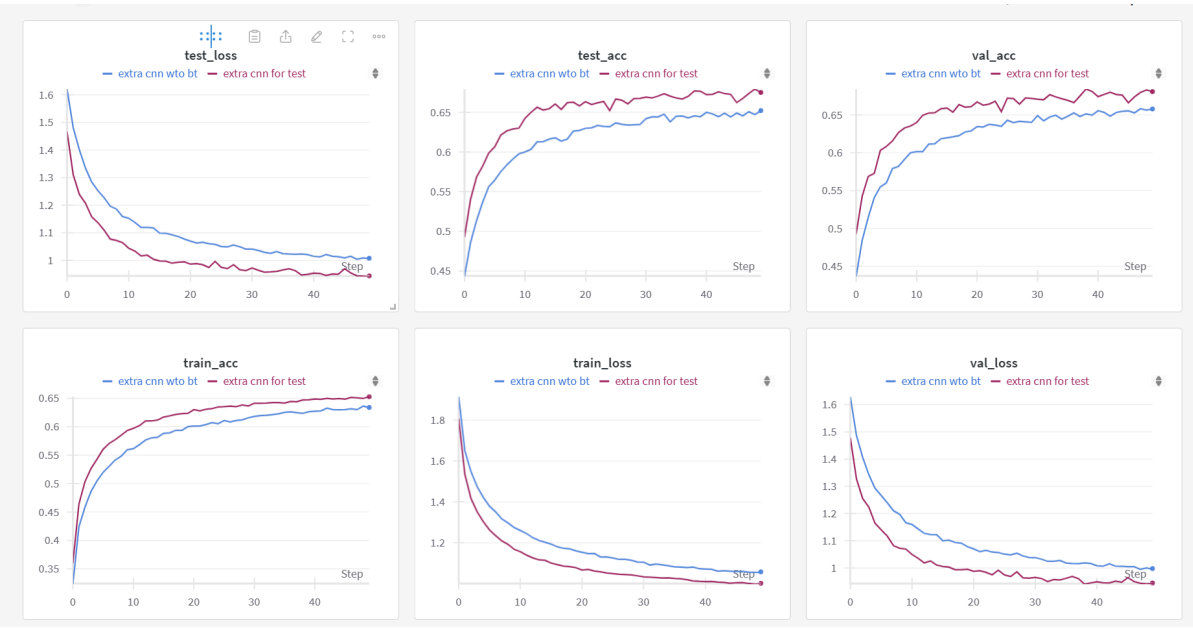
- 毫无疑问的是，DropOut 的存在可以有效防止过拟合，减小训练过程中三种数据集效果的差异。
- DropOut 的存在可以让训练曲线具有更好的收敛性。
- 用DropOut训练的神经网络中的每个隐藏单元必须学会与随机选择的其他单元样本一起工作，这样会使得每个隐藏单元更加的健壮，并使得他们尽量自己可以创造有用的功能，而不是由于训练集的路径依赖，导致特殊特征的出现使得神经网络的参数单元出现粘合和依附的情况。

Construct MLP and CNN without batch normalization, and discuss the effects of batch normalization.

依然使用了最佳表现实验的参数作为对比。

MLP

CNN



可以看到两者在去除 BatchNorm 之后整体效果都有打折扣，可以说明这个方法是一个相对健壮的优化。

BatchNorm 的影响

- 在没有 BatchNorm 的情况下，网络的初始化对训练结果有很大影响。BatchNorm 通过规范化每个 mini-batch 的输入，减少了这种依赖，使得数据的分布更加稳定，这样可以使用更高的学习率而不会导致训练过程的不稳定。
- BatchNorm 有助于防止在训练过程中出现的梯度消失和梯度爆炸问题，因为它可以保持每层输入的均值和方差在一定范围内，使得反向传播时梯度不会太小或太大。

额外要求

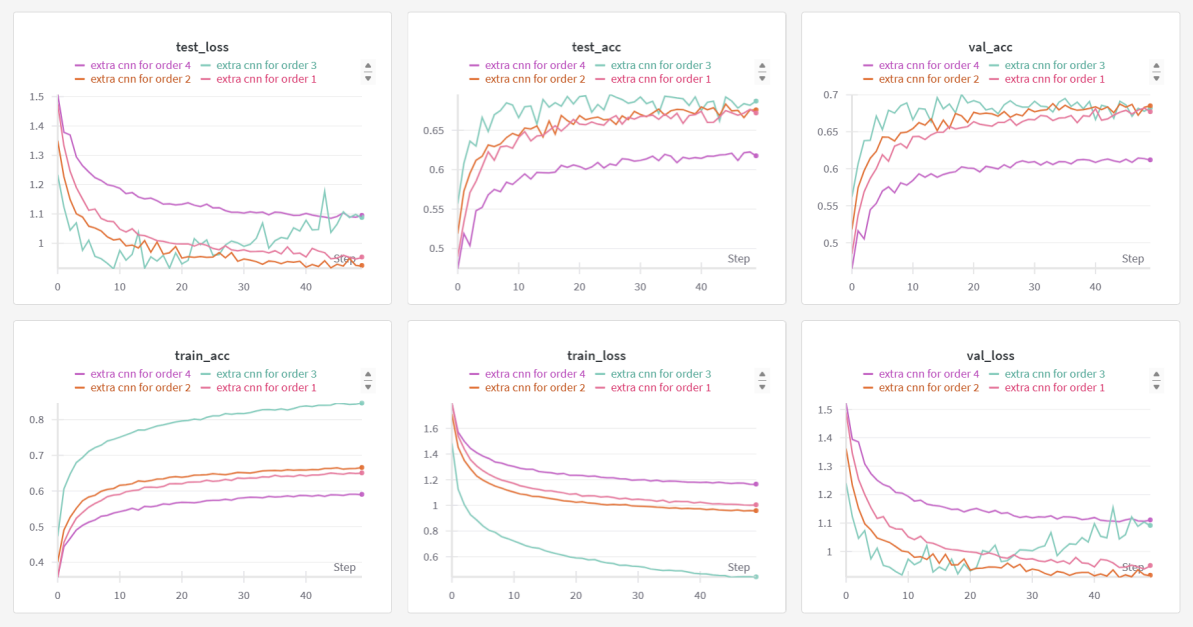
Consider changing the orders of different network blocks in CNN， Show the results with different orders and explain why.

我们同样使用最好的参数。

其中一些我认为相对合理的结构编号如下：

```
        operations = [
            nn.Conv2d
            (in_channels=in_channel,
             out_channels=out_channel,
             kernel_size=3,
             stride=1,
             padding=1),
            BatchNorm2d(out_channel),
            nn.ReLU(),
            Dropout(drop_rate),
            nn.MaxPool2d(kernel_size=2, stride=2),
        ]
        ordered_operations = []
        if oper_seed == 1: # 正常顺序
            ordered_operations = operations
        elif oper_seed == 2: # 调换 relu 和 norm 的位置
            ordered_operations = [operations[0], operations[2], operations[1],
operations[3], operations[4]]
        elif oper_seed == 3: # 去掉 dropout
            ordered_operations = [operations[0], operations[1], operations[2],
operations[4]]
        elif oper_seed == 4: # 调换 maxpool 到最前面
            ordered_operations = [operations[4], operations[0], operations[1],
operations[2], operations[3]]
        else:
            ordered_operations = operations
```

以下是四种结构的训练反馈：



结合图上来看：

- 去除 DropOut 由于上文已经分析过，就不再分析，而剩下三种结构的收敛性都较好。
- 先激活再 BN 的结构 2 效果居然比默认结构 1 要略好一些，我的理解是：虽然先激活之后实际上改变了数据的分布，但是其对于该数据集过滤掉了一些对于网络优化效果不佳的负结果，并且让方差更小，从而影响权重和偏置的更新。
- 而结构 4 先池化的效果不出预料地很差，因为池化会丢弃一些图片初始拥有的特征，使得一开始图片就是有损的结构，这很大可能会导致一些混淆的情况出现，因此效果变差。

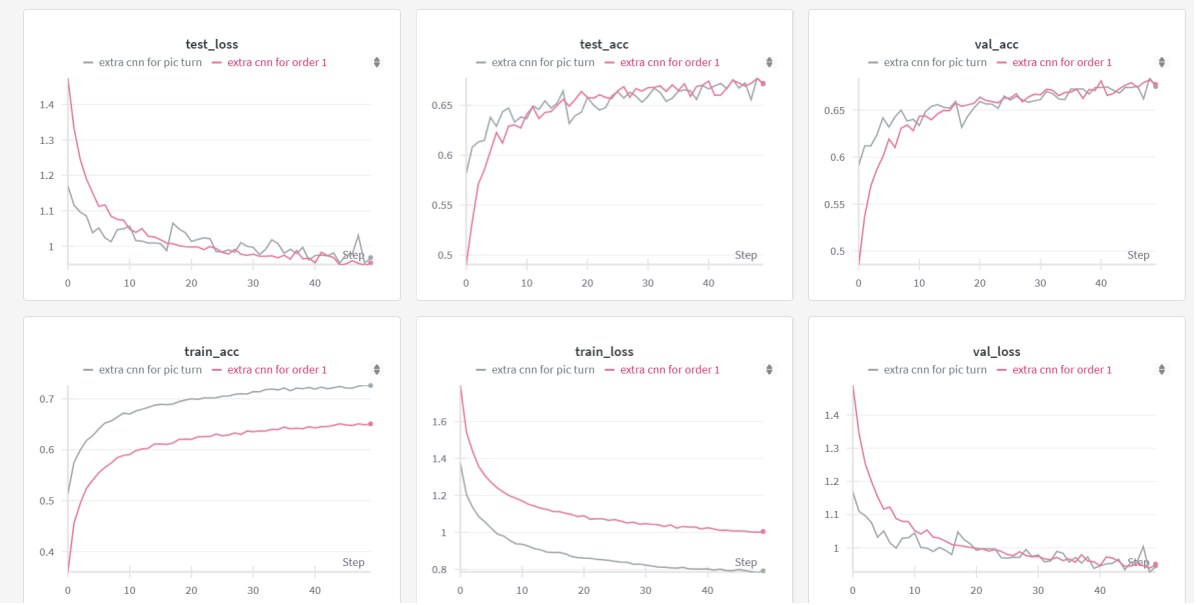
Try additional data augmentation tricks

我对 CNN 使用的方法是：对训练集中的图片进行随机翻转。翻转并不影响物体种类。

我将 `cnn.main` 中的 `shuffle` 进行了如下改动：

```
def shuffle(X, y, shuffle_parts):
    ......
    ......
    for k in range(shuffle_parts):
        np.random.shuffle(shuffled_range)
        for i in range(chunk_size):
            if (i % 5 == 0): # 1/5 的图片被左右翻转
                X_buffer[i] = X[k * chunk_size + shuffled_range[i]][:, :, ::-1]
            elif (i % 5 == 1): # 1/5 的图片被上下翻转
                X_buffer[i] = X[k * chunk_size + shuffled_range[i]][:, ::-1, :]
            else:
                X_buffer[i] = X[k * chunk_size + shuffled_range[i]]
    ......
    ......
    return X, y
```

并且调整了训练参数，将学习率调整至 `2e-3`，Dp 调整至 `0`，Batch 调整至 `128`：



可以看到 train 和 val 的提升都比较明显，test 有略微提升，但是总体差距不太大。

我的理解是，通过在训练过程中引入翻转的图像，不仅人为地理想下增加了数据集的大小，同时也鼓励模型学习到对图像方向不变的特征。这意味着即使测试图像在水平方向上有所不同，模型也能够正确识别。