深度学习 第一次作业

1 BackPropagation

本来在LaTeX上推导了一遍，但回头看的时候太繁琐了，所以在此简要地记录重点。详细推导请见 DL_hw1.pdf

1.1 BN

$$\begin{aligned}\frac{\partial \boldsymbol{y}^i}{\partial \boldsymbol{\gamma}} &= diag(\hat{\boldsymbol{x}}^{\boldsymbol{i}}) \\ \frac{\partial \boldsymbol{y}^i}{\partial \boldsymbol{\beta}} &= \mathbb{I}\end{aligned} \tag{1}$$

1.2 Softmax

$$\begin{aligned}\text{softmax}(x_i) &= \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} \\ \boldsymbol{y} = \text{softmax}(\boldsymbol{x}) &= [\text{softmax}(x_1), \dots, \text{softmax}(x_n)]^\top = [y_1, \dots, y_j]^\top \\ \frac{\partial y_i}{\partial x_j} &= \frac{-\exp(x_i)\exp(x_j)}{[\sum_{k=1}^n \exp(x_k)]^2} = -y_i y_j, \quad i \neq j \\ \frac{\partial y_i}{\partial x_i} &= \frac{\exp(x_i)\{[\sum_{k=1}^n \exp(x_k)] - \exp(x_j)\}}{[\sum_{k=1}^n \exp(x_k)]^2} = y_i(1-y_i)\end{aligned} \tag{2}$$

1.3 Feed Forwad

$$\begin{aligned}\boldsymbol{x}_{1A} &= \boldsymbol{\theta}_{1A}\boldsymbol{x} + \boldsymbol{b}_{1A} \\ \boldsymbol{x}_{DP} &= \boldsymbol{M} \odot \sin(\boldsymbol{x}_{1A}) \\ \hat{\boldsymbol{y}}_A = \boldsymbol{x}_{2A} &= \boldsymbol{\theta}_{2A}\boldsymbol{x}_{DP} + \boldsymbol{b}_{2A}\end{aligned}$$

$$\begin{aligned}\boldsymbol{x}_{1B} &= \boldsymbol{\theta}_{1B}\boldsymbol{x} \\ \boldsymbol{\mu} &= \frac{1}{m}\sum_{i=1}^m \boldsymbol{x}_{1B}^i \\ \boldsymbol{x}_{BN} &= \boldsymbol{x}_{1B} - \boldsymbol{\mu} + \boldsymbol{b}_{1B} \\ \boldsymbol{x}_C &= \text{ReLU}(\boldsymbol{x}_{BN}) \oplus \boldsymbol{x}_{2A} \\ \boldsymbol{x}_{2B} &= \boldsymbol{\theta}_{2B}\boldsymbol{x}_C + \boldsymbol{b}_{2B} \\ \hat{\boldsymbol{y}}_B &= \text{softmax}(\boldsymbol{x}_{2B})\end{aligned} \tag{3}$$

1.4 Gradients

首先，可以推导出这些结论：

对于一个全连接网络，记参数为 $\boldsymbol{\theta}, \boldsymbol{b}$，输入输出 $\boldsymbol{x}, \boldsymbol{y}$ ，满足

$$\boldsymbol{y} = \boldsymbol{\theta}\boldsymbol{x} + \boldsymbol{b} \tag{4}$$

它的反向梯度传播满足

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \boldsymbol{x}^\top \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} &= \boldsymbol{\theta}^\top \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}
\end{aligned} \tag{5}
$$

然后对于两个 `loss` ，分别有

$$
\begin{aligned}
\frac{\partial \text{softmax}(\boldsymbol{x})}{\partial \boldsymbol{x}} &= \text{diagonal}(\text{softmax}(\boldsymbol{x})) - \text{softmax}(\boldsymbol{x})\text{softmax}(\boldsymbol{x})^\top \\
\frac{\partial \text{CrossEntropy}(\boldsymbol{y}_B, \hat{\boldsymbol{y}}_B)}{\partial \hat{\boldsymbol{y}}_B} &= \boldsymbol{y}_B \oslash \hat{\boldsymbol{y}}_B
\end{aligned} \tag{6}
$$

其中，$\oslash$ 为 Hadamard division，即两个相同形状矩阵逐元素除法。

利用这些信息，进行推导可以得到：

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{2B}} &= \frac{1}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{B}}^{\boldsymbol{i}}) {\boldsymbol{x}_C^i}^\top \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_{2B}} &= \frac{1}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{B}}^{\boldsymbol{i}})
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{1B}} &= \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{1B}^i} {\boldsymbol{x}_i}^\top \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{1B}^i} &= (1 - \frac{1}{m}) \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{BN}^i} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{BN}^i} &= \text{ReLU}'(\boldsymbol{x}_{BN}^i) \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_C^i} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_C^i} &= \frac{1}{m} \boldsymbol{\theta}_{2B}^\top (\hat{\boldsymbol{y}}_B^i - \boldsymbol{y}_B^i)
\end{aligned} \tag{8}
$$

$$
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{1B}} = \frac{1}{m} \sum_{i=1}^{m} [(1 - \frac{1}{m}) \text{ReLU}'(\boldsymbol{x}_{BN}^i) \boldsymbol{\theta}_{2B}^\top (\hat{\boldsymbol{y}}_B^i - \boldsymbol{y}_B^i)] {\boldsymbol{x}_i}^\top
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{2A}} &= \sum_{i=1}^{m} [\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{2A}^i}] {\boldsymbol{x}_{DP}^i}^\top \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{2A}^i} &= \frac{1}{m} [2(\hat{\boldsymbol{y}}_{\boldsymbol{A}}^{\boldsymbol{i}} - \boldsymbol{y}_A^i) - \boldsymbol{\theta}_{2B}^\top (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_B^i)] \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{2A}} &= \frac{1}{m} \sum_{i=1}^{m} [2(\hat{\boldsymbol{y}}_{\boldsymbol{A}}^{\boldsymbol{i}} - \boldsymbol{y}_A^i) - \boldsymbol{\theta}_{2B}^\top (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_B^i)] {\boldsymbol{x}_{DP}^i}^\top \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_{2A}} &= \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{2A}^i} = \frac{1}{m} \sum_{i=1}^{m} [2(\hat{\boldsymbol{y}}_{\boldsymbol{A}}^{\boldsymbol{i}} - \boldsymbol{y}_A^i) - \boldsymbol{\theta}_{2B}^\top (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_B^i)]
\end{aligned} \tag{9}
$$

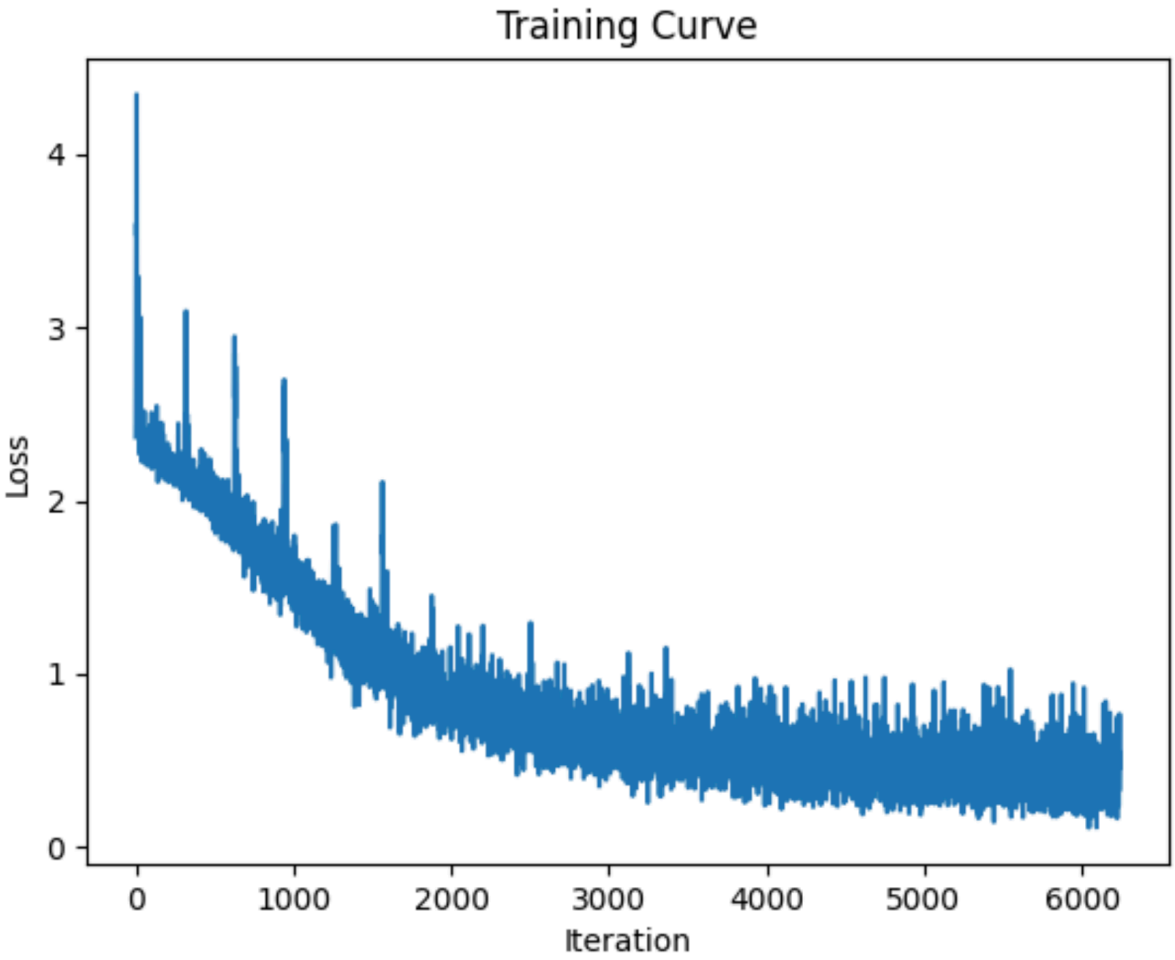
$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{1A}} &= \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{1A}^{i}} \boldsymbol{x}^{i\top} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{1A}^{i}} &= \cos(\boldsymbol{x}_{1A}^{i}) \odot \boldsymbol{M} \odot \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{DP}^{i}} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{DP}^{i}} &= \boldsymbol{\theta}_{2A}^{\top} \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{2A}^{i}} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{2A}^{i}} &= \frac{1}{m} 2(\hat{\boldsymbol{y}}_{\boldsymbol{A}}^{\boldsymbol{i}} - \boldsymbol{y}_{A}^{i}) - \frac{1}{m} \boldsymbol{\theta}_{2B}^{\top} (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_{B}^{i}) \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{1A}} &= \frac{1}{m} \sum_{i=1}^{m} \{\cos(\boldsymbol{x}_{1A}^{i}) \odot \boldsymbol{M} \odot \{\boldsymbol{\theta}_{2A}^{\top} [2(\hat{\boldsymbol{y}}_{\boldsymbol{A}}^{\boldsymbol{i}} - \boldsymbol{y}_{A}^{i}) - \boldsymbol{\theta}_{2B}^{\top} (\hat{\boldsymbol{y}}_{\boldsymbol{B}}^{\boldsymbol{i}} - \boldsymbol{y}_{B}^{i})]\}\}
\end{aligned}
\tag{10}
$$

2 Section 2.1 代码补全

2.1 默认参数

代码已经补全在相应的文件中。以下是默认参数的训练曲线，可以观察到loss是下降的，但震荡的厉害，推测原因是 `batch_size` 较小。

具体实现细节，我在 `softmax` 实现中，算 `exp` 时先整体减去了向量的最大元素，在算 `log` 前进行了 `numpy.clip`，避免出现 `log(0)` 的问题，增加数值稳定性。



2.2 调参

经过大力调参，我最终选取参数设置。注意，我发现原有的 `SGD` 并不能很好的拟合，所以我实现了简单的带 `momentum` 的SGD，详见代码。

```
lr: 1e-1
epoch: 60
momentum: 0.9
hidden_dim: 128
batch_size: 128
```

Model	train acc	test acc	train loss
MLP-128	0.9173	0.903	0.221947

3 Section 2.2 CNN训练

3.1 Task A 运行现有代码

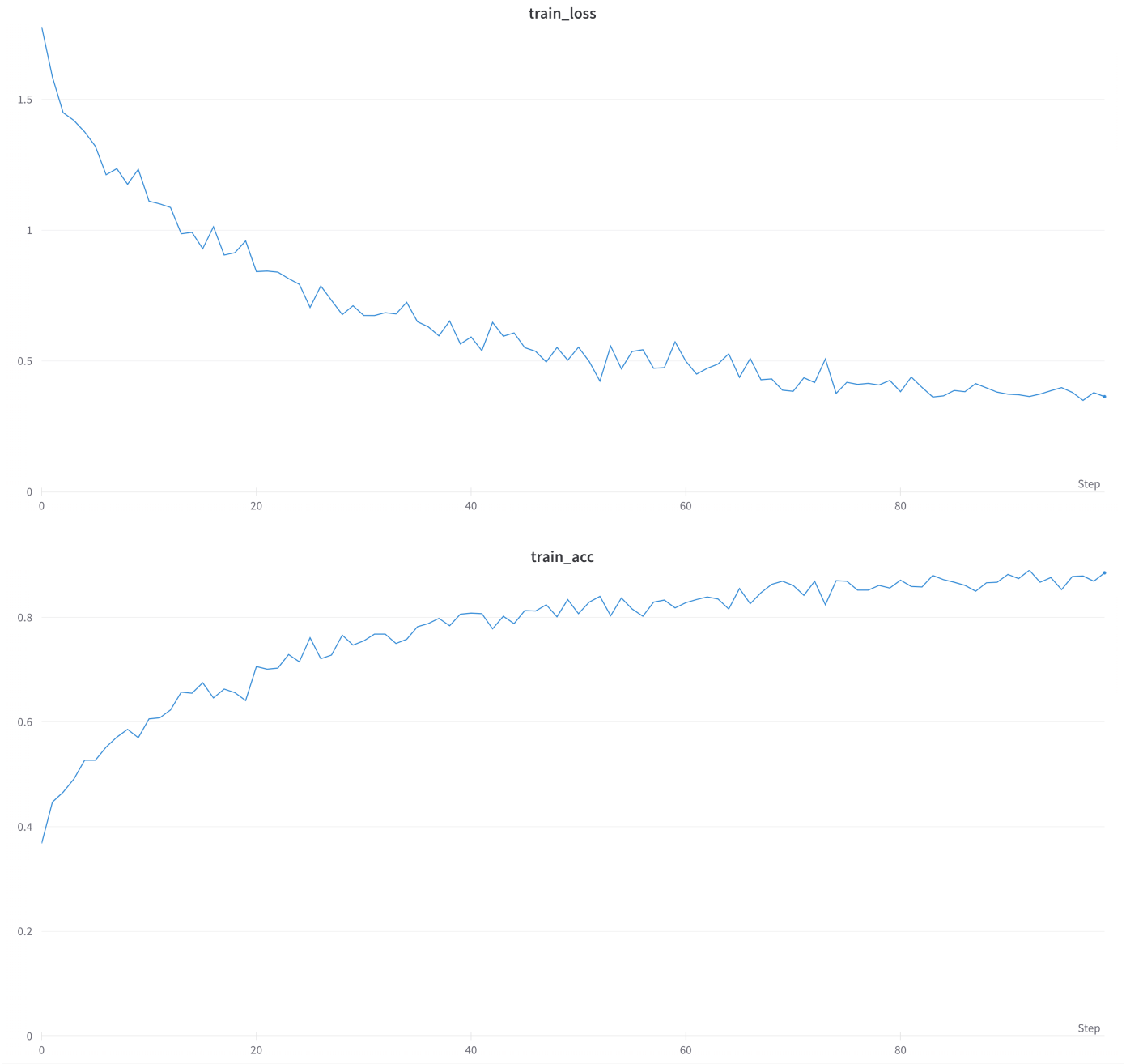
为了获得极致的性能，我没有按照默认设定，而是“稍微”调了下参，看看基线最强的性能。最终，我在以下超参数条件下运行了实验。

```
model: torchvision.resnet-18
num_classes: 10
input_size: 224
batch_size: 36
num_epochs: 100
lr: 1e-3
Optimizer: AdamW
beta1: 0.9
beta2: 0.99
momentum: 0.9
weight_decay: 0.01
cuda: 10.2 Quadro RTX 24GiB
```

得到了如下结果

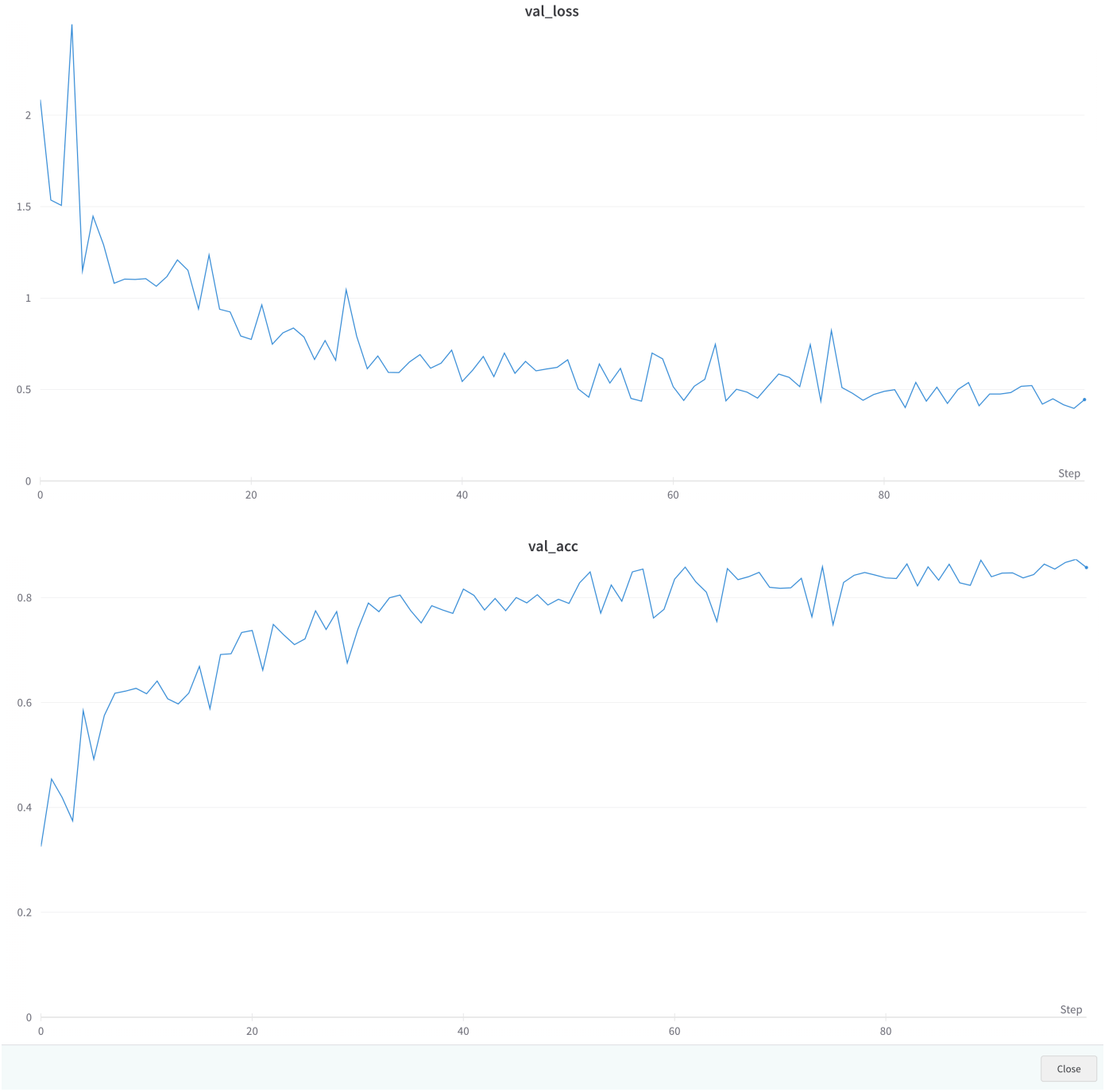
模型	train loss	train acc	val loss	Val acc
ResNet-18	0.3891	0.8810	0.4504	0.8498

训练时曲线图如下所示


train_loss
1.5
1
0.5
0
Step
0
20
40
60
80

train_acc
0.8
0.6
0.4
0.2
0
Step
0
20
40
60
80

Close



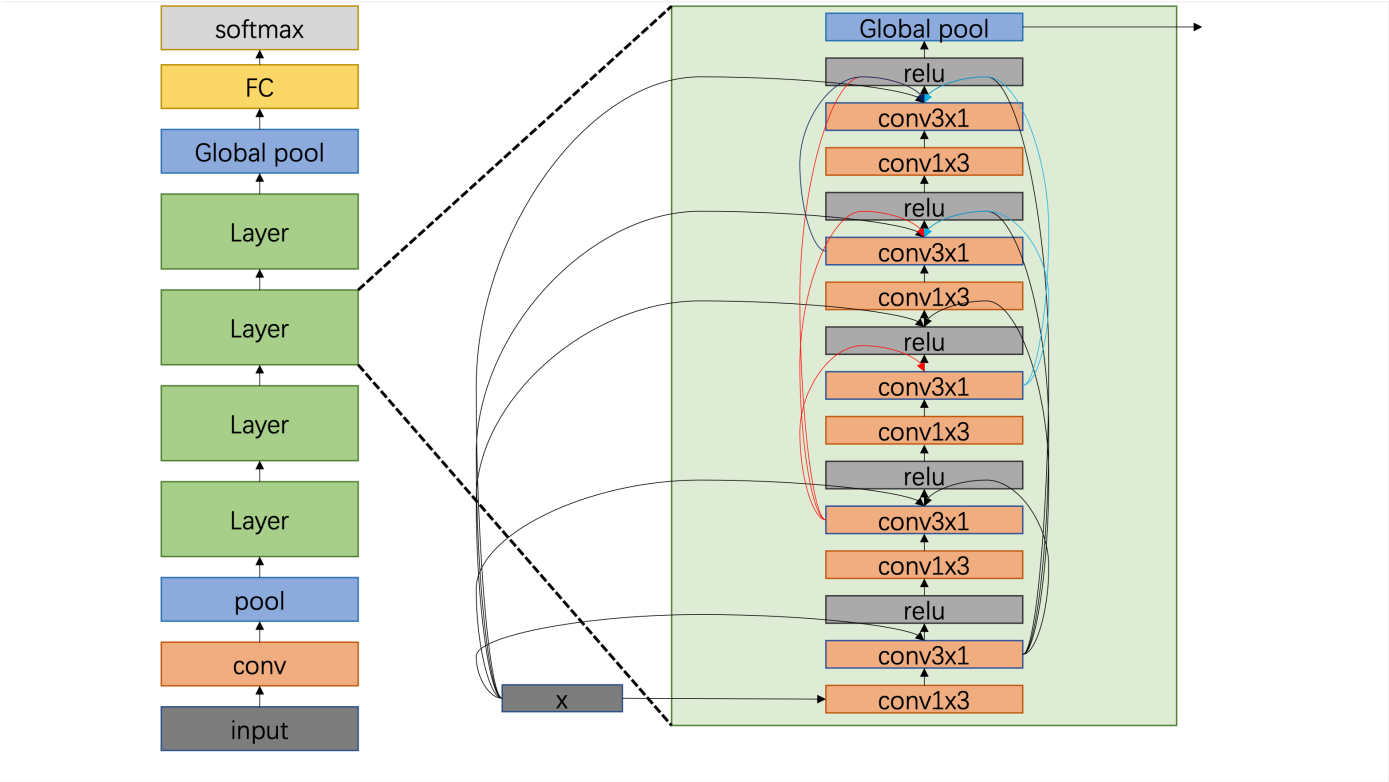
3.2 Task B 设计新模型

以下是我设计的模型结构。在我设计的模型中，我参考了`Naive GooLeNet`的思路，把一个`conv3x3`转化为两个`conv1x3`和`conv3x1`在垂直方向上的组合，减少了参数，从效果上来讲还提升了性能。其次，我也参考了`DenseNet`的思路，将多跳`FeedForward`的方法实现在普通的`Resnet basic block`中。

在具体的代码实现过程中，我首先用`torch.nn.Conv2d`模块进行堆叠，但是一开始手动实现出来的`resnet18`比`torchvision`库中提供的`resnet`性能差了不少。于是，我进行了库代码阅读，学习了它面向对象的基本框架。我发现`ResNet`在初始化参数的时候，有许多魔法（例如`nn.init.kaiming_`，又比如`global pooling`与单层`fc`的设想，以及单个`ResNet layer`内`BasicBlock`的数量与`downpooling`的位置等）在坚持`ResNet`大框架不动摇的前提下，对`BasicBlock`和`BottleNeck`进

行了修改，实现了不少有益的变动。最终，我的模型架构如下图所示。

一个 `Layer` 由多个 `BasicBlock` 组成，这也是超参数：四个 `Layer` 的 block 数量，以 `resnet-18` 为例，记为 `[2,2,2,2]`。

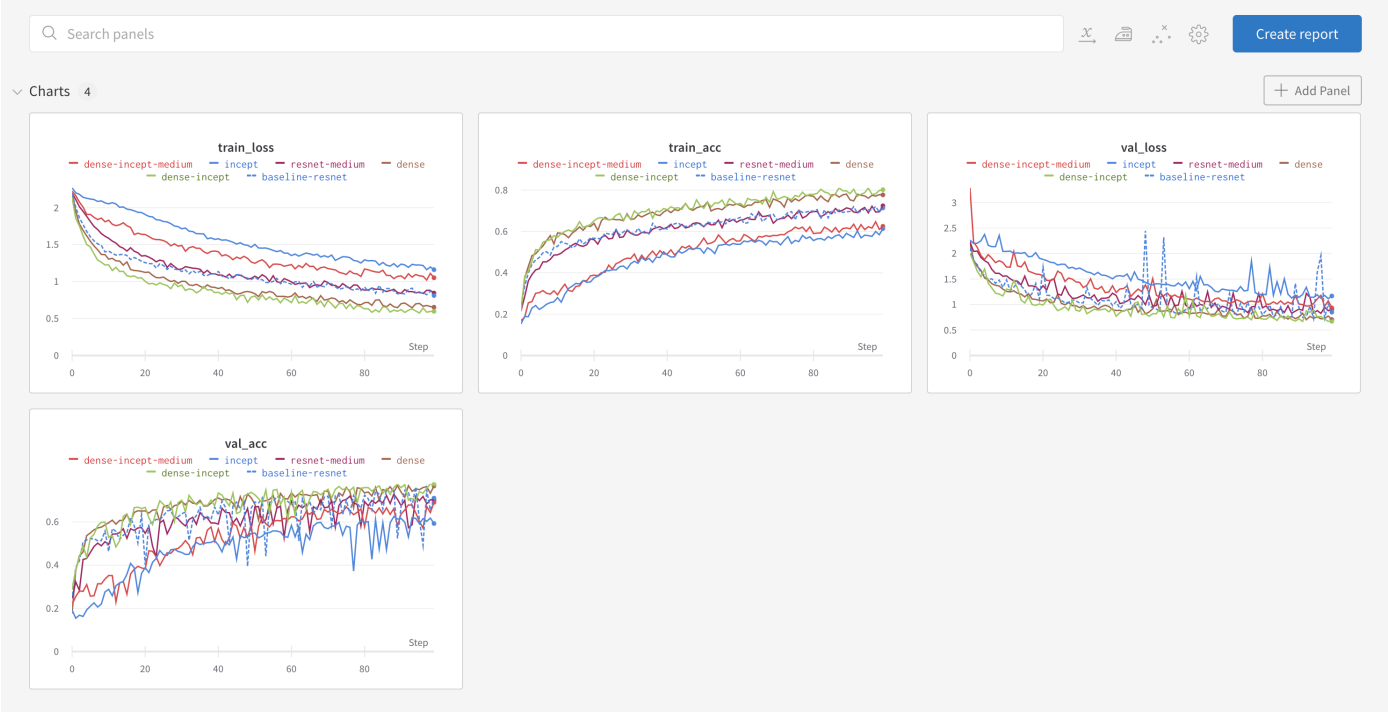


以下是训练超参数

```
model: torchvision.resnet-18
num_classes: 10
input_size: 224
batch_size: 36
num_epochs: 100
lr: 1e-2
Optimizer: SGD
weight_decay: 0.01
cuda: 10.2 Quadro RTX 24GiB
```

为了探究新增结构对性能的影响，我做了以下的逐步消融。其中，默认模型四层layer的参数为 `[2,2,2,2]`，**Medium** 的layer参数为 `[2,6,4,3]`。结果如下图、下表。

模型	val acc	val loss	train acc	train loss
ResNet-18	0.7098	0.8504	0.714	0.8124
ResNet-medium	0.7011	0.9093	0.724	0.845
DenseBlock	0.7661	0.7055	0.777	0.6545
InceptBlock	0.5924	1.168	0.614	1.162
DenseInceptBlock	**0.7741**	**0.6721**	**0.802**	**0.5983**
DenseInceptBlock-medium	0.6904	0.9342	0.625	1.052



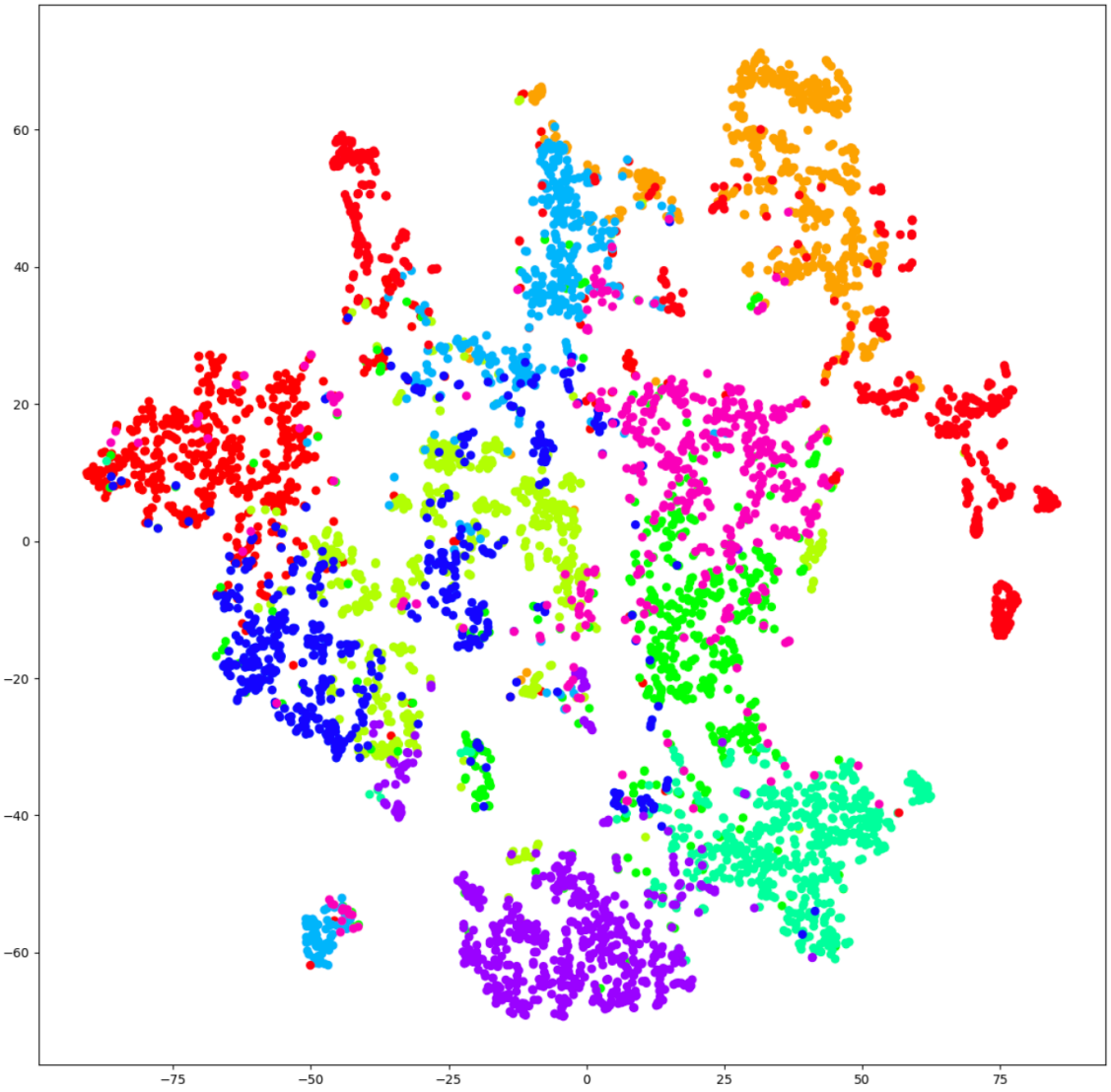
从结果中可以看出，去掉 resnet 的 feedforward 换成 incept 的结构，性能会下降不少，说明单独的 incept 结构比不上feedforward。比较 dense 的多跳 feedforwad 与 resnet ，可以发现性能有一定的提升，说明 densenet 的多跳 feedforward 比单跳有增益。再结合 dense 与 denseincept ，可以发现在 dense 的基础上加入 incept 的结构，性能进一步提升， incept 对模型有性能增益。

最后，比较 **medium** 与 baseline 的性能，可以发现 层数越多性能没有提升，看曲线发现其收敛速度显著变慢，说明参数量（层数）增加后，模型变得更难收敛，在同样的迭代步数内性能没有提升。

3.3 分类头前feature可视化

如图所示，对所有测试集内的图片，输入模型，将 `x.flatten(1)` 后的 feature 向量经过 t-SNE 降维至2个维度，按照类别同色绘制点图如下。绘制脚本见 `CNN/test_model.py` 。我选取的 checkpoint 是 `DenseInceptBlock` 的参数。

从图中可以看出，不同类别的图像在经过模型后，特征出现了一定程度的聚类现象，这说明模型能够提取到相关特征，在分类头前将其区分。它的意义是进行了简单的消融，模型的能力大部分是前置的CNN提供的，而不全是分类头。



3.4 数据增强与学习率调整

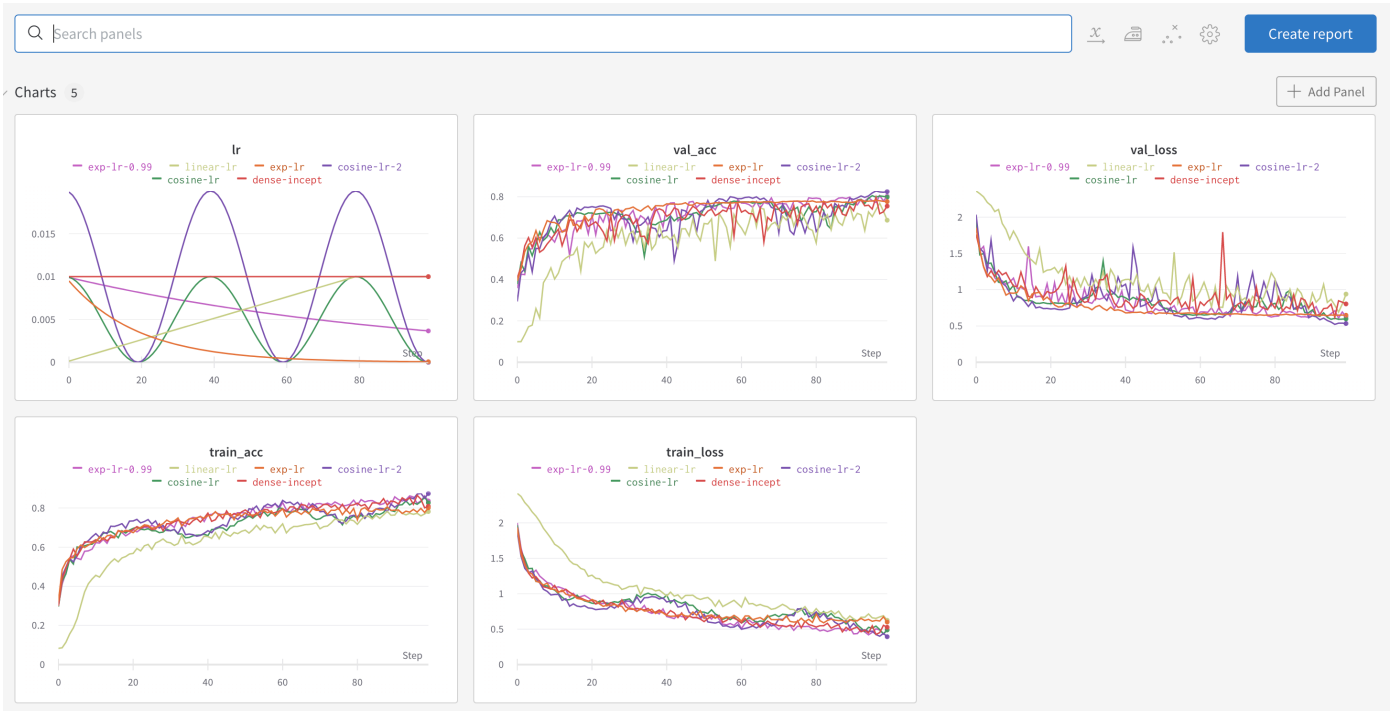
3.4.1 学习率调整

我尝试了三种学习率调整方法，分别是 `CosineAnnealingLR`， `LinearLR`，`ExponentialLR`，是通过调用 `torch.optim.lr_scheduler` 实现的，其中：

- `CosineAnnealingLR`：$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_m in)(1 + \cos(\frac{T_{cur}}{T_{max}}))$，`T_max`=`num_epochs // 5`
- `LinearLR`：$\eta_t = [start + \frac{(\eta_{max} - start)}{iters} * epoch] * \mathbb{I}_{epoch \leq iters} + \eta_{max}\mathbb{I}_{epoch > iters}$，`start`=`1e-3`，`iters`=`80`
- `ExponentialLR`：$\eta_t = \gamma^{epoch}$，`gamma=0.99`

初始学习率调整为 `1e-2`，在 `Cosine` 中尝试了 `\eta_max = 2e-2` 的方式，记为 cosine-double。各个方法的效果如下。

Method	val acc	val loss	train acc	train loss
None	0.7546	0.804	0.809	0.5271
Cosine	0.7981	0.5977	0.827	0.4892
Cosine-double	**0.8246**	**0.534**	**0.874**	**0.3937**
Exp	0.7765	0.6481	0.801	0.6006
LinearLR	0.6865	0.9397	0.782	0.626



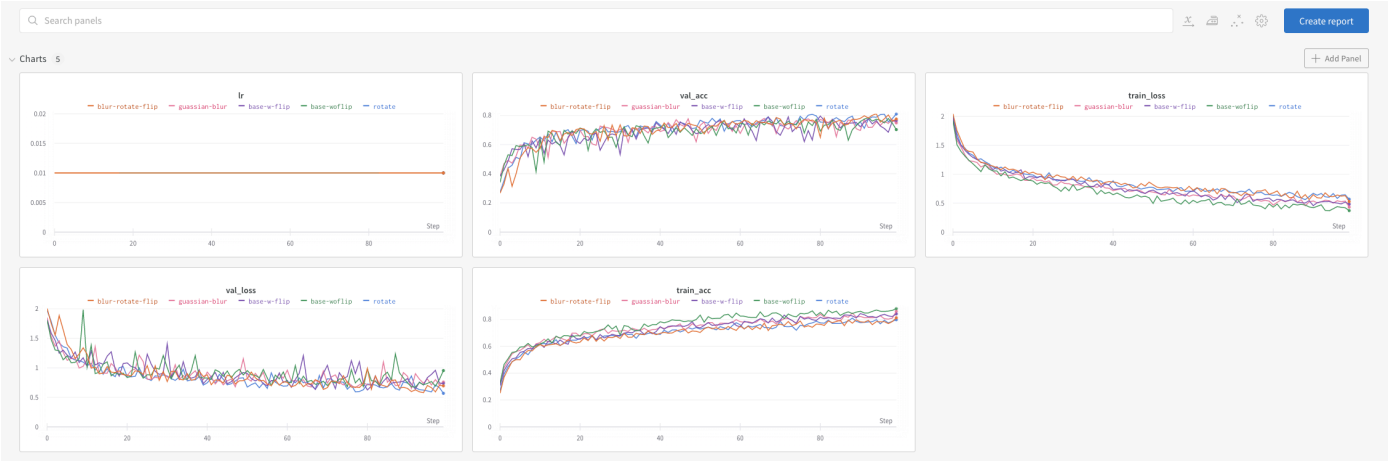
从表中，**cosine-double** 方法拥有四个指标上最好的性能，而它学习率在训练过程中的均值与 **None** 方法一样都是 `1e-2`，这说明了在接近收敛的时候，动态调节学习率步长有助于模型收敛到更好的最优点上去。类似地，cosine 与 exponential 都能够达到比 None 更好的测试集准确率。

3.4.2 数据增强

在默认的实现数据处理中，已经加入了 `RandomResizedCrop` 与 `RandomHorizontalFlip` 两种处理手段了。`RandomHorizontalFlip` 是左右镜面对称，在地图图像处理中，直觉上是非常有用的手段，因此在后续添加其他数据增强手段中，我在它的基础上继续应用其他处理方法。

- `RandomRotation`：将图片旋转一个角度，直觉上增加了地图的观察角度，类似于旋转kernel角度扩大感受野
- `GaussianBlur`：对图片内加入高斯模糊，使得模型在训练时能够对抗一定的噪声

method	Val acc	Val loss	Train acc	Train loss
base w/o flip	0.7033	0.9536	0.882	0.3725
Base	0.7657	0.7385	0.842	0.4833
Rotate	**0.8106**	**0.5686**	**0.799**	**0.5676**
Gaussian Blur	0.7498	0.7584	0.862	0.433
rotate+blur	0.7757	0.6952	0.813	0.5279



从实验结果中可以看出，对实验效果性能提升最大的数据增强手段是 `Flip` 和 `Rotate`，在 Val acc 上分别实现了 +0.0624，+0.0449 的提升，整体超过了 10%。而高斯噪声没有提升反而降低的原因，大概是数据训练集本身质量一般，图片模糊有较多噪声，因此这种数据增强方式并没有带来性能提升。

3.4.3 结论

最后，我把 `CosineAnnealingLR` 与 `Flip` 和 `Rotate` 结合起来训练模型，得到如下结果。

method	Val acc	Val loss	Train acc	Train loss
Final model	0.8206	0.5359	0.814	0.4833