HW5 Report

【备注】在打包文件夹中，也提交了六个保存训练模型的权重偏置等的 `.pth` 文件，可以在 函数 `def test(..)` 的 `checkpoint = torch.load('checkpoint_epoch_50_fcnn_new.pth',map_location=device)` 语句调用不同参数组合，能够复现报告中"结果对比"部分最终在test集上的正确率。分别是：

1. `checkpoint_epoch_50_fcnn_new.pth` —— fcnn, adamw + step, 也是报告中fcnn下正确率=60.7%的参数组合
2. `checkpoint_epoch_50_fcnn_new_step.pth` ——fcnn, adamw + step
3. `checkpoint_epoch_50_fcnn_new_sgd.pth` ——fcnn, sgd + cosine
4. `checkpoint_epoch_50_fcnn_new_sgd_step.pth` ——fcnn, sgd + step
5. `checkpoint_epoch_50_cnn.pth` ——cnn，sgd +step
6. `checkpoint_epoch_50_linear.pth` ——linear, adamw + cosine

模型结构

LinearClassifier

```
self.linear = nn.Linear(in_channels=3072, out_channels=10)
```

FCNN

选择有5个隐藏层的结构，以relu作为激活函数，并加入batch norm：

```
def __init__(self, in_channels: int, hidden_channels: int, out_channels: int):
    super().__init__()
    hidden = [512, 256, 128, 64,32]
    self.relu = nn.ReLU()
    self.bn1 = nn.BatchNorm1d(hidden[0])
    self.bn2 = nn.BatchNorm1d(hidden[1])
    self.bn3 = nn.BatchNorm1d(hidden[2])
    self.bn4 = nn.BatchNorm1d(hidden[3])
    self.bn5 = nn.BatchNorm1d(hidden[4])

    self.fc1 = nn.Linear(in_channels, hidden[0])
    self.fc2= nn.Linear(hidden[0], hidden[1])
```

```
        self.fc3= nn.Linear(hidden[1], hidden[2])
        self.fc4= nn.Linear(hidden[2], hidden[3])
        self.fc5= nn.Linear(hidden[3], hidden[4])
        self.fc6= nn.Linear(hidden[4], out_channels)

    def forward(self, x: torch.Tensor):
        x=self.fc1(x)
        x=self.bn1(x)
        x=self.relu(x)

        x=self.fc2(x)
        x=self.bn2(x)
        x=self.relu(x)

        x=self.fc3(x)
        x=self.bn3(x)
        x=self.relu(x)

        x=self.fc4(x)
        x=self.bn4(x)
        x=self.relu(x)

        x=self.fc5(x)
        x=self.bn5(x)
        x=self.relu(x)
        x=self.fc6(x)
        return x
```

训练过程

`train`函数中，在初始数据处理阶段，都加入归一化处理，并对训练集的数据进行数据增强——随机调整图片的亮度、对比度、饱和度和色调；随机旋转；随机平移

```
transform_train = transforms.Compose([
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.RandomRotation(15),
    transforms.RandomAffine(degrees=0, translate=(0.12, 0.12)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

然后，在训练阶段进行多轮重复，每轮包括：

- 遍历训练集，前向传播、计算损失、反向传播、优化参数，累计loss和准确率。
- 根据scheduler调整学习率
- 在测试集上评估模型，计算loss和准确率，记录到TensorBoard。

训练结束时保存模型和优化器等的checkpoint

最终选择参数组合：

```
#def train(...)
epoch=50
batch_size=128

#optimizer
optimizer = torch.optim.AdamW(model.parameters(), lr=0.007)

#scheduler
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50, eta_min=0.0)
```

这样组合下的正确率可达60.7%：

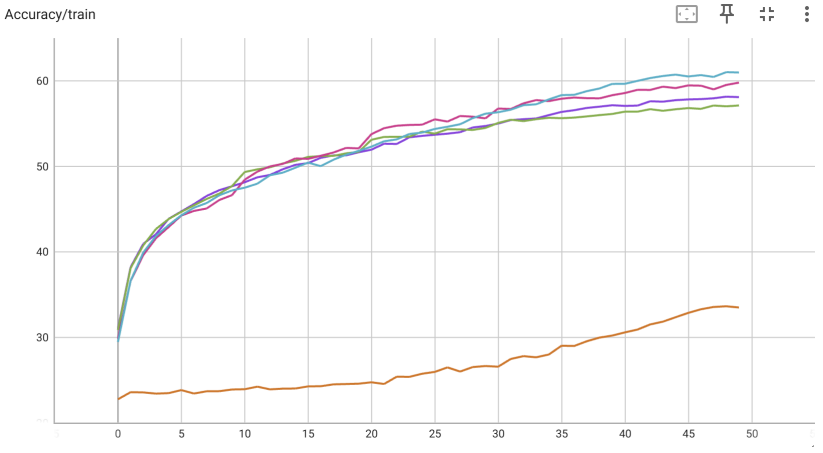
```
root@autodl-container-26904cacf3-5fb78e5c:~/autodl-tmp# python main.py --run=test --model=fcnn --optimiz
er=adamw --scheduler=cosine
Final Test Loss: 1.1162, Final Test Accuracy: 60.7200
```

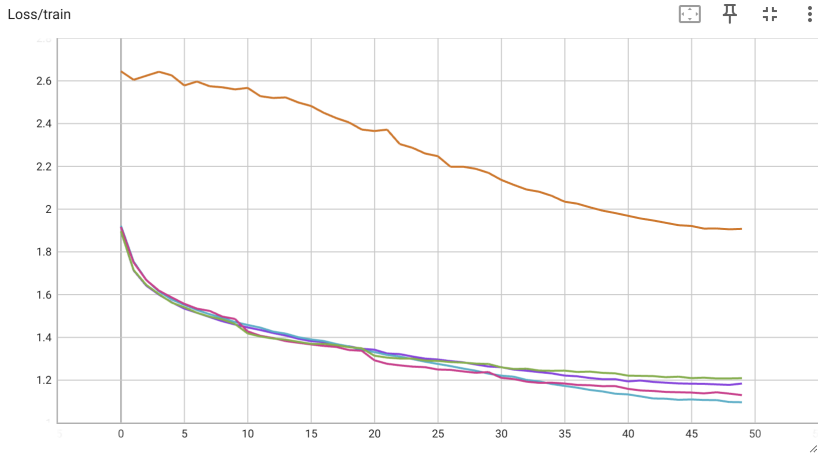
结果对比

曲线对应颜色和最终的测试集正确率（第50轮，epoch=49）：

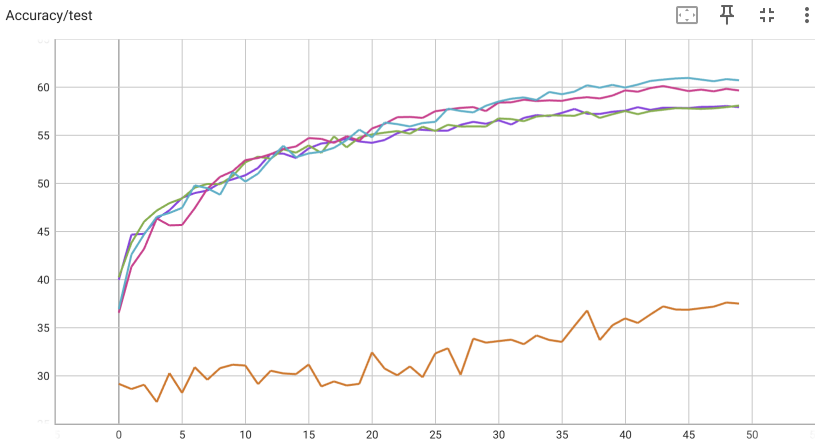
Run ↑	Value	Step
fcnn_adamw_cosine	60.72	49
fcnn_adamw_step	59.66	49
fcnn_sgd_cosine	57.93	49
fcnn_sgd_step	58.09	49
linear_adamw_cosine	37.51	49

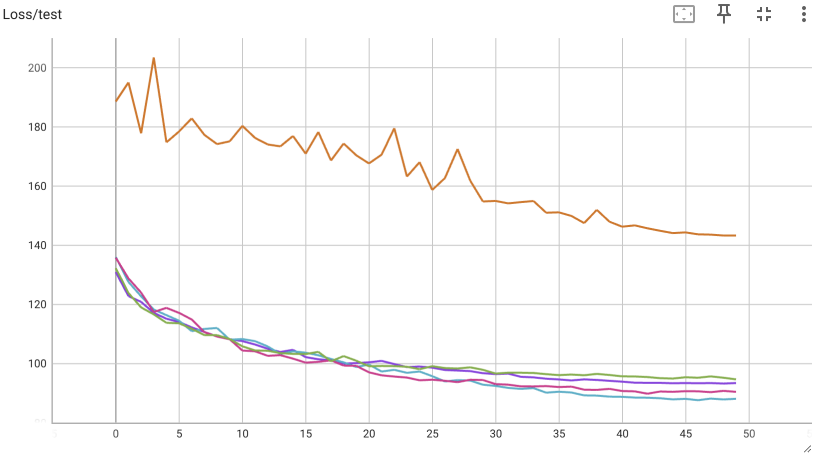
在训练集上的正确率、loss：





`def train(...)`中每轮训练时在测试集上的正确率、loss：

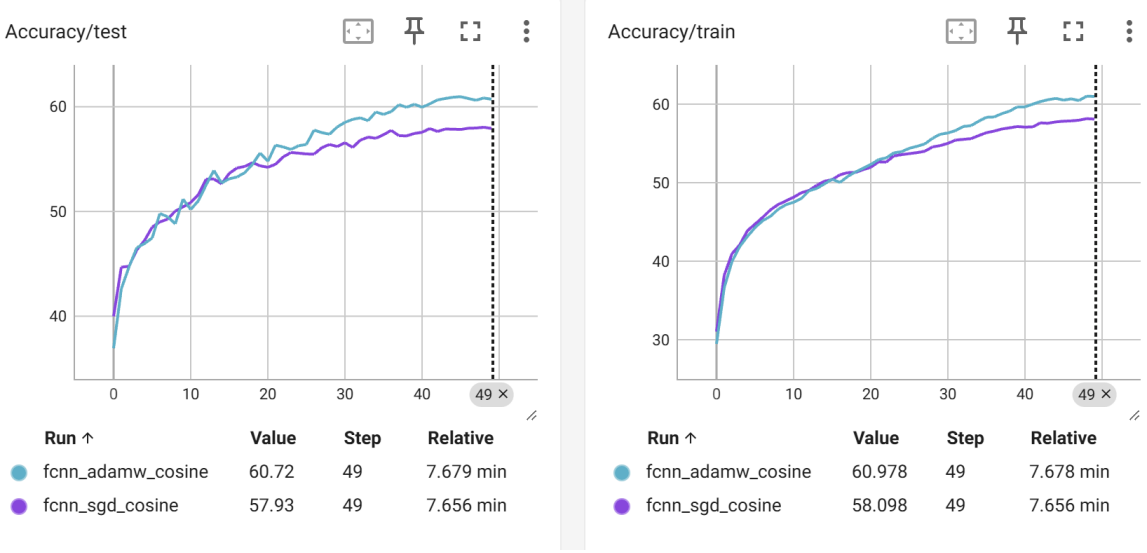


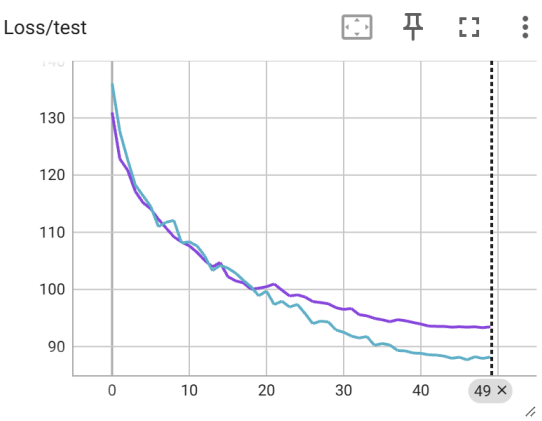


明显看出，linearclassifier（选择adamw+cosine）在收敛速度、正确率、loss大小的表现明显不如

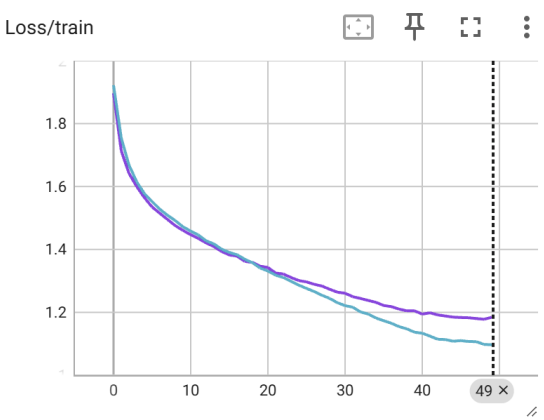
1. Optimizer: AdamW and SGD

AdamW + cosine 与 sgd + cosine在test/train下的正确率、loss对比：



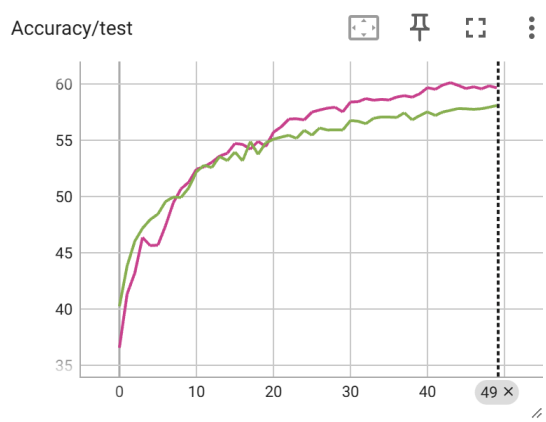


Run ↑	Value	Step	Relative
fcnn_adamw_cosine	88.1574	49	7.679 min
fcnn_sgd_cosine	93.4645	49	7.656 min

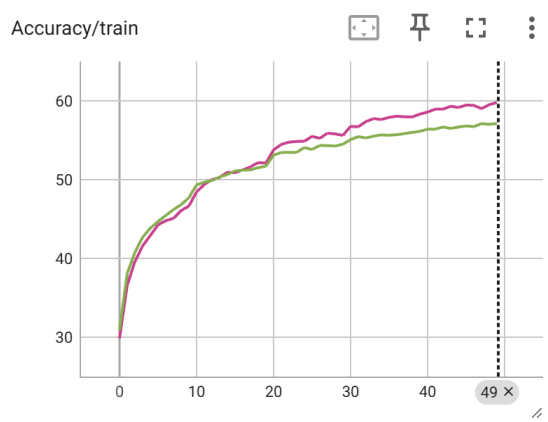


Run ↑	Value	Step	Relative
fcnn_adamw_cosine	1.0968	49	7.678 min
fcnn_sgd_cosine	1.1841	49	7.656 min

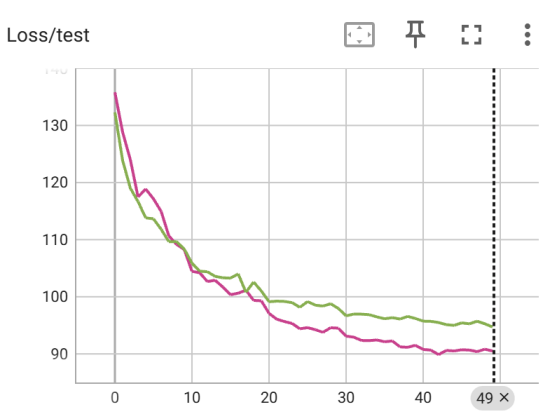
AdamW + step 与 sgd + step在test/train下的正确率、loss对比：



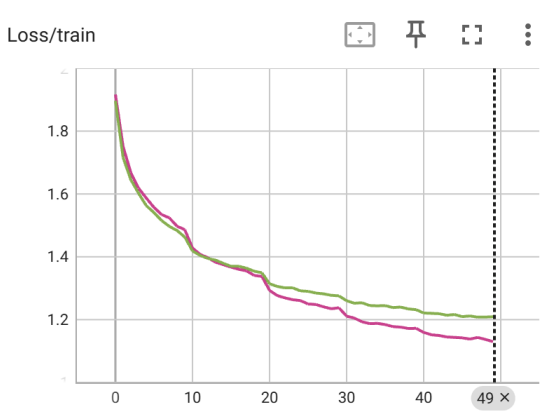
Run ↑	Value	Step	Relative
fcnn_adamw_step	59.66	49	7.636 min
fcnn_sgd_step	58.09	49	7.669 min



Run ↑	Value	Step	Relative
fcnn_adamw_step	59.794	49	7.636 min
fcnn_sgd_step	57.122	49	7.671 min



Run ↑	Value	Step	Relative
fcnn_adamw_step	90.503	49	7.636 min
fcnn_sgd_step	94.7175	49	7.669 min

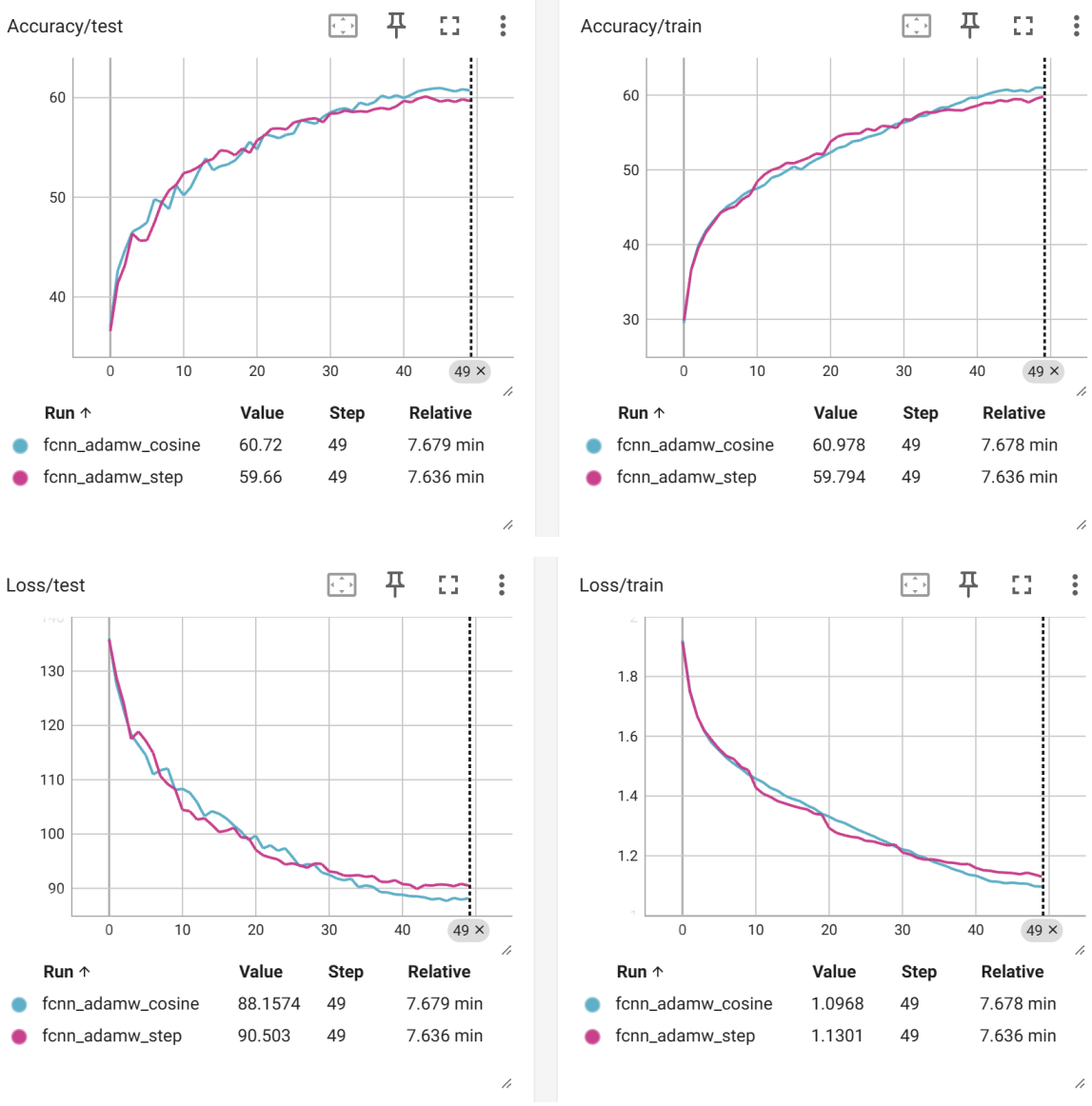


Run ↑	Value	Step	Relative
fcnn_adamw_step	1.1301	49	7.636 min
fcnn_sgd_step	1.2092	49	7.671 min

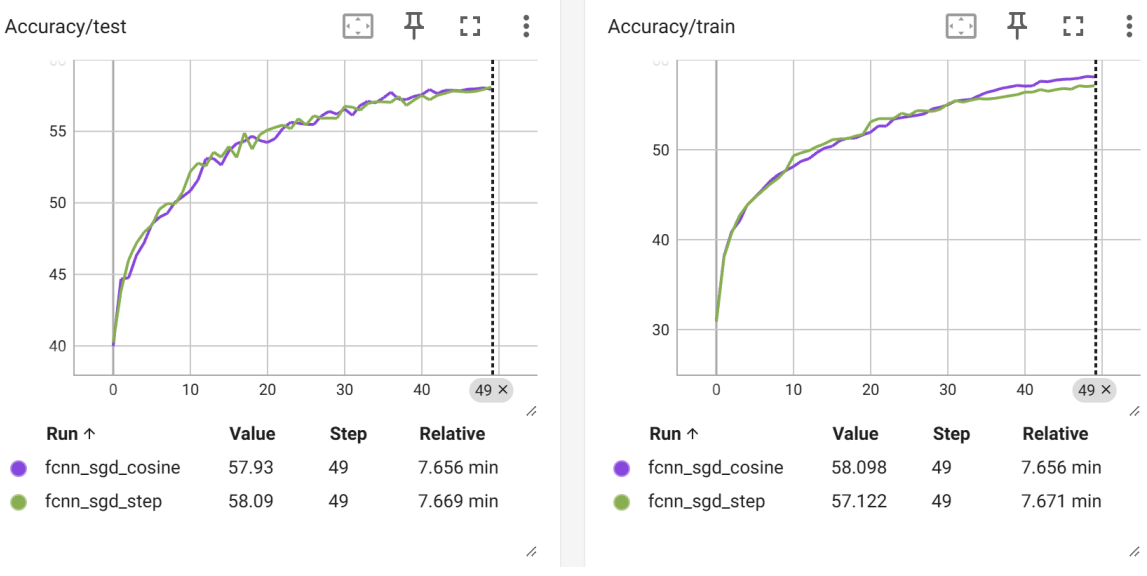
可以看到，AdamW在训练集和每轮测试集中，表现整体均比 sgd 收敛更快、准确率更高且loss更低。

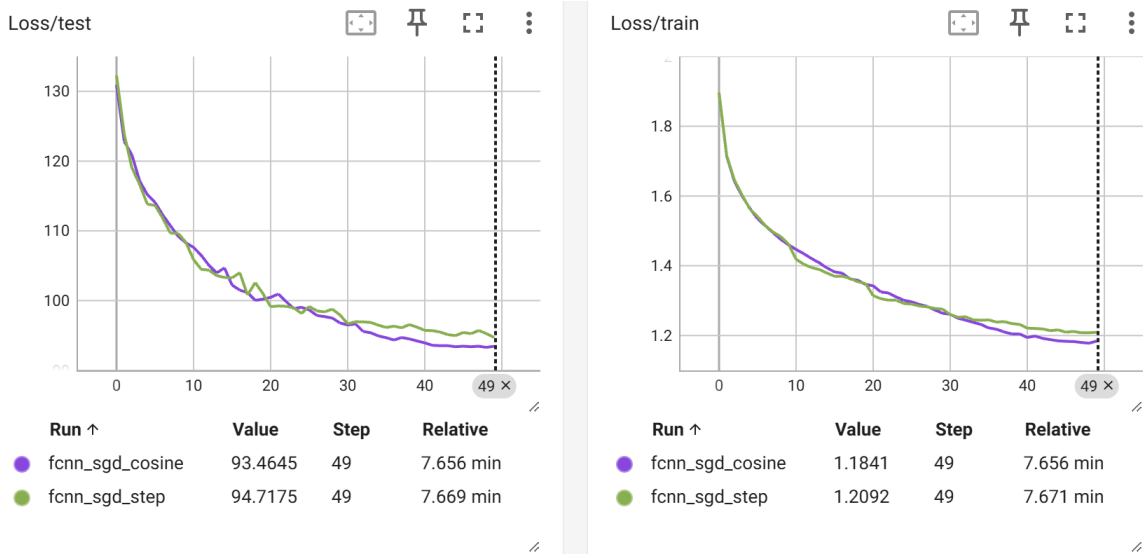
2. Scheduler: StepLR and CosineAnnealingLR

AdamW + step 与 AdamW + cosine在test/train下的正确率、loss对比：



sgd + step 与 sgd + cosine在test/train下的正确率、loss对比：





两种scheduler在结果反应来看区别不大，从loss来看cosine较step效果较好一些。

因此，最终选择 AdamW + CosineAnnealingLR的组合，它们在学习率、epoch、batch_size等参数组合相同的情况下，正确率最高，达到60.7%。

CNN

发现linearclassifier的效果一般，fcnn调参后变化可以优化的水平依然不佳，而引入卷积神经网络可以显著提升效果。比如选择这个结构的cnn：

```
class CNNClassifier(nn.Module):
    def __init__(self, in_channels: int, out_channels: int):
        super().__init__()
        self.conv1=nn.Conv2d(in_channels, 32, kernel_size=3, stride=1, padding=1)
        self.conv2=nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3=nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.fc1=nn.Linear(128*4*4, 256)
        self.fc2=nn.Linear(256, out_channels)

        self.relu=nn.ReLU()
        self.dropout=nn.Dropout(p=0.5)
        self.pool=nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

    def forward(self,x:torch.Tensor):
        x=self.conv1(x)
        x=self.relu(x)
        x=self.pool(x)

        x=self.conv2(x)
        x=self.relu(x)
        x=self.pool(x)

        x=self.conv3(x)
        x=self.relu(x)
        x=self.pool(x)

        x=x.view(x.size(0), -1)
```

```
        x=self.fc1(x)
        x=self.relu(x)
        x=self.dropout(x)
        x=self.fc2(x)
        return x
```

同时，注意到train或者test函数中向前传播的部分，fcnn、linearclassifier的输入需要展成一维向量，而cnn可以直接处理形状为[batch, 3, 32, 32]的输入，分类处理：

```
# forward
if isinstance(model, CNNClassifier):     ##卷积
    outputs = model(inputs)
else:
    outputs = model(inputs.view(inputs.size(0), -1))
```

cnn效果明显优于fcnn，如代码中选择在**sgd + step**的组合下训练，其在测试集的最终正确率可达80%。具体的正确率和loss对比图：