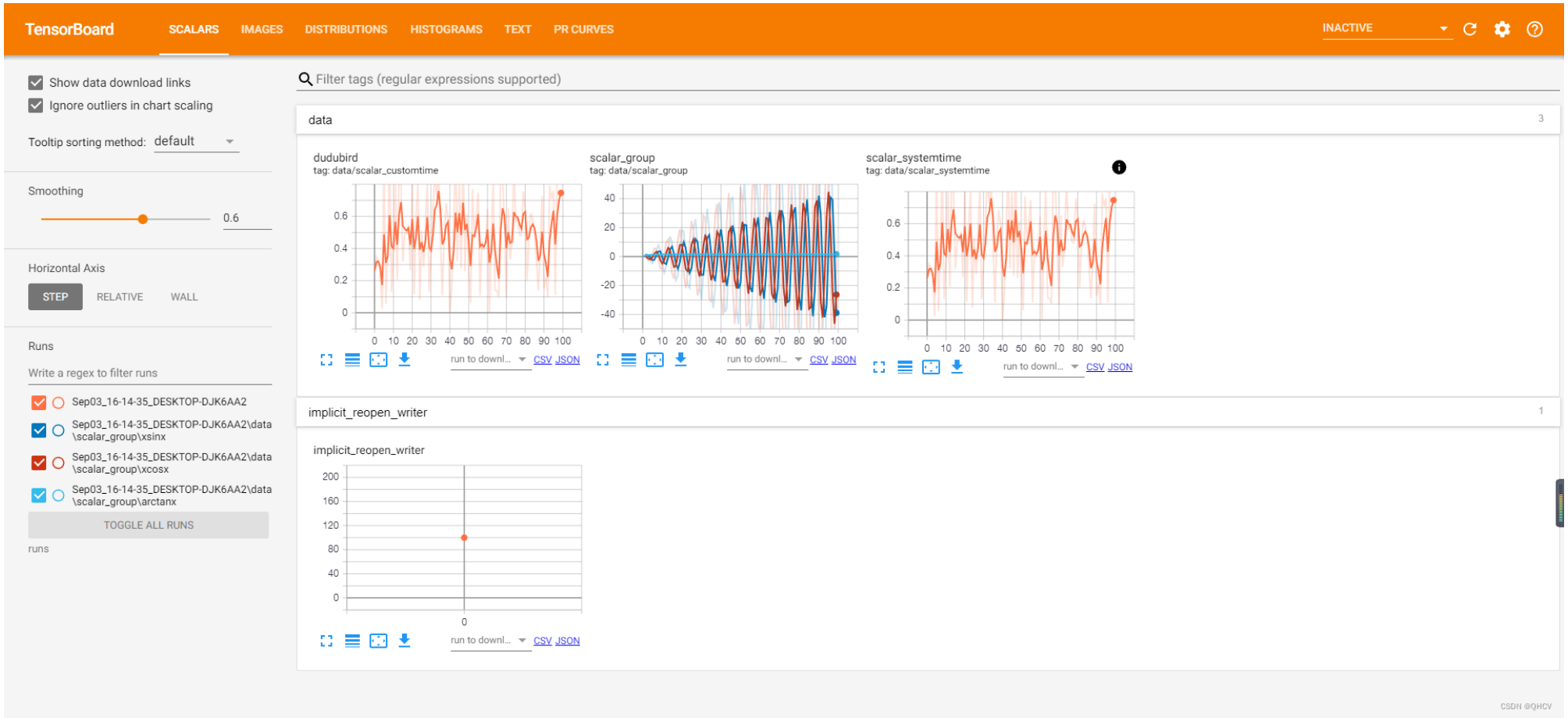
第二章 模型管理与可视化工具

模型可视化工具——Tensorboard

Tensorboard：是一组用于数据可视化的工具，本质上是一个 实验日志可视化工具。主要功能包括：

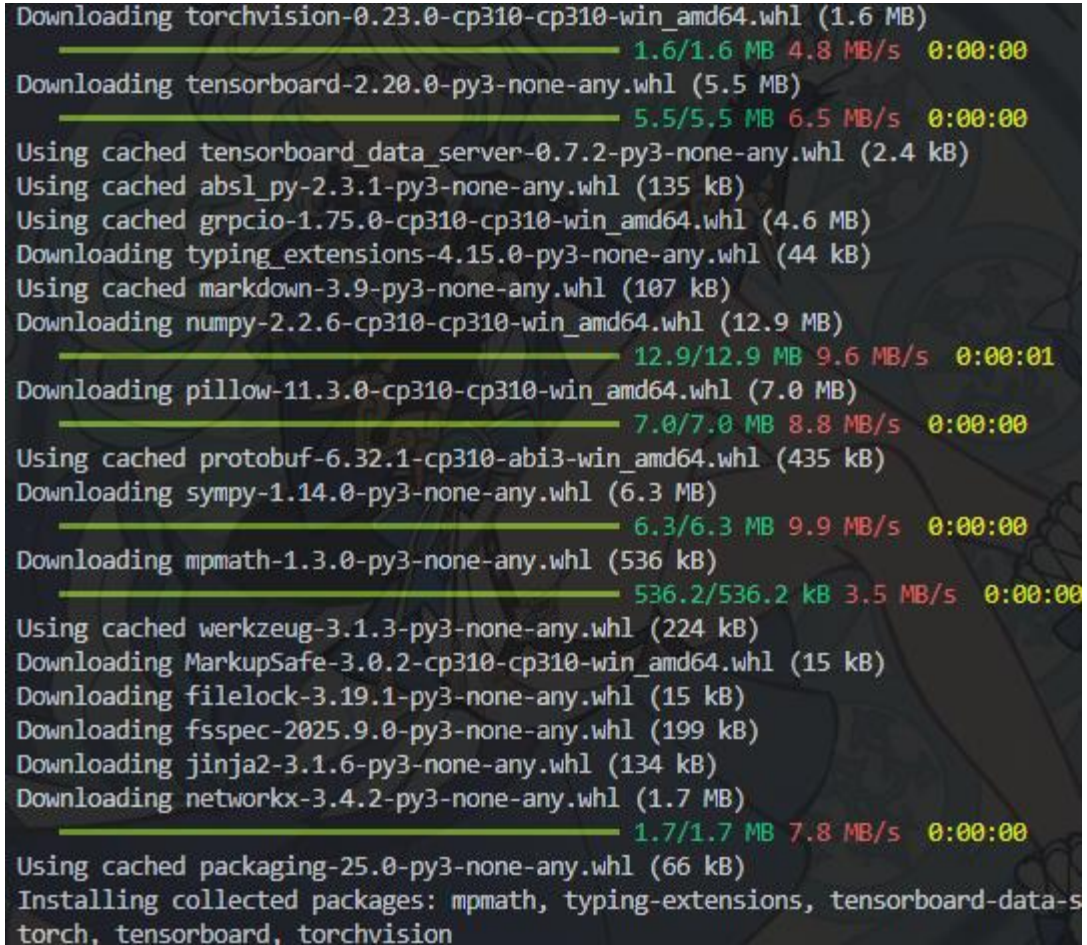
1. 可视化模型的网络架构
2. 跟踪模型指标，如损失和准确性等
3. 检查机器学习工作流程中权重、偏差和其他组件的直方图
4. 显示非表格数据，包括图像、文本和音频
5. 将高维嵌入投影到低维空间



PyTorch+Tensorboard环境安装

环境安装：

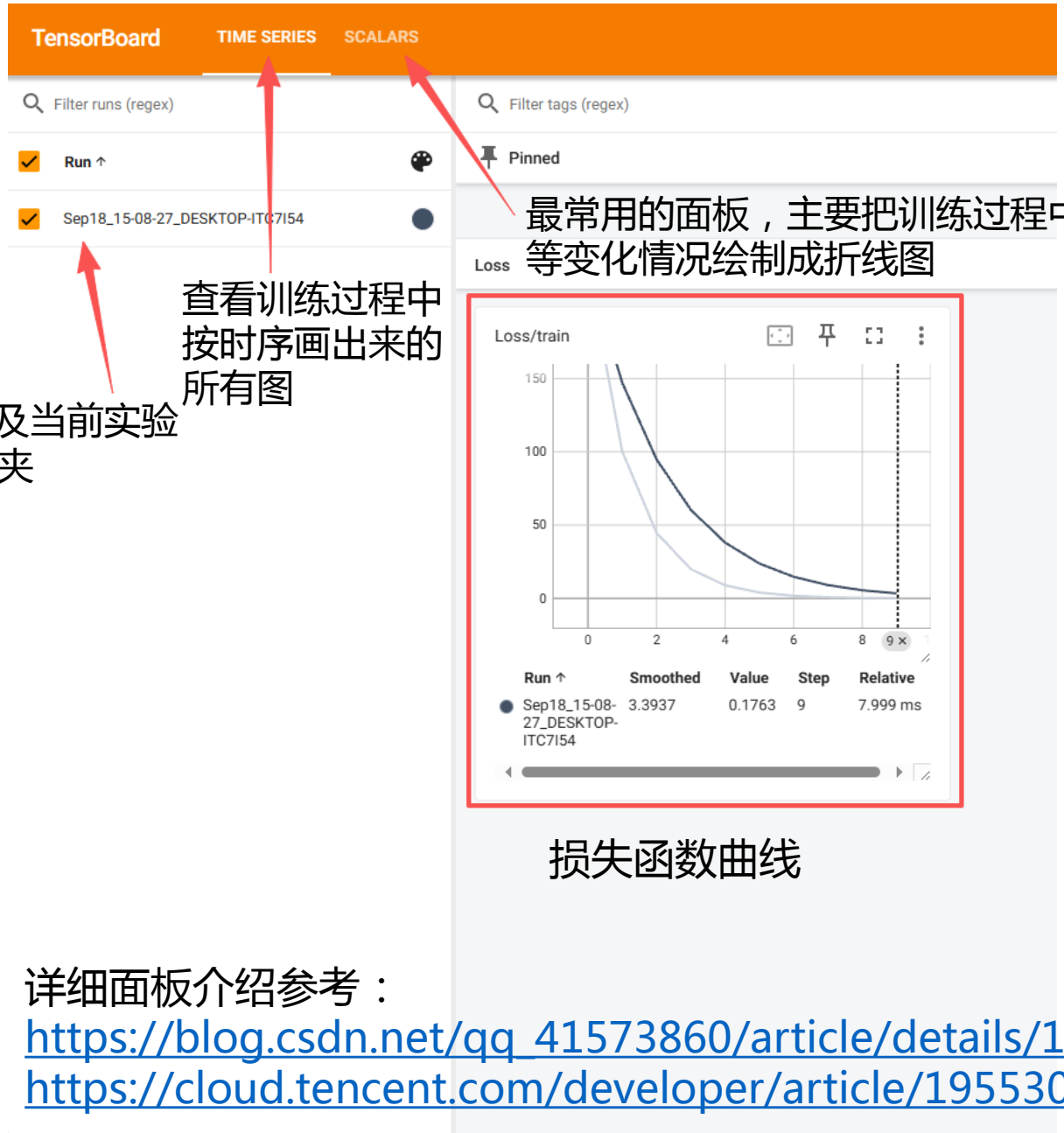
```
conda create -n torch_env python=3.10 -y
conda activate torch_env
pip install torch torchvision tensorboard
pip install scikit-learn
pip install matplotlib seaborn
pip install pytorch-lightning
```



Tensorboard示例——线性回归

```python
# 导入相应模块
import torch
from torch.utils.tensorboard import SummaryWriter # TensorBoard 的写入接口，用来把标量、图像、直方图等写入事件文件，以便在 TensorBoard 中可视化。

writer = SummaryWriter() # 默认会在当前目录下创建 runs/<自动生成的名字>/，也可以指定文件夹名称：SummaryWriter(log_dir="my_log_dir")

# 随机生成一些数据用于训练
x = torch.arange(-5, 5, 0.1).view(-1, 1)
y = -5 * x + 0.1 * torch.randn(x.size())

# 定义模型、损失函数、优化器
model = torch.nn.Linear(1, 1)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

# 训练函数
def train_model(iter):
    for epoch in range(iter):
        y1 = model(x) # 前向传播得到预测值
        loss = criterion(y1, y) # 计算预测值与真实值的损失
        writer.add_scalar("Loss/train", loss, epoch) # 将损失值写入 TensorBoard
        optimizer.zero_grad() # 梯度清零
        loss.backward() # 反向传播计算梯度
        optimizer.step() # 更新模型参数

train_model(10) # 训练 10 个 epoch（这里是全批次训练，每个 epoch 都用同一批 100 个样本）
writer.flush() # 把缓冲区的日志强制写入磁盘（通常在大量写入时有用）
writer.close() # 关闭 writer，释放资源。之后 TensorBoard 可以读取到这些事件文件并显示曲线
```

线性回归结果及面板介绍



最常用的面板，主要把训练过程中的acc、val_acc、loss、weight等变化情况绘制成折线图

查看训练过程中按时序画出来的所有图

Run文件夹及当前实验的日志文件夹

损失函数曲线

详细面板介绍参考：
https://blog.csdn.net/qq_41573860/article/details/106674370
https://cloud.tencent.com/developer/article/1955304

Tensorboard示例——基因表达数据对疾病二分类

训练数据：随机生成的基因表达模拟数据，共1000个样本，50个基因，疾病标签为0或1

```
1  X, X.shape
```

✓ 0.0s

```
(tensor([[ 1.9269,  1.4873,  0.9007,  ..., -0.4879, -0.9138, -0.6581],
         [ 0.0780,  0.5258, -0.4880,  ...,  0.4880,  0.7846,  0.0286],
         [ 0.6408,  0.5832,  1.0669,  ...,  1.4506,  0.2695, -0.2104],
         ...,
         [-1.7840, -0.2211,  1.2709,  ..., -1.4763, -0.5407, -1.4797],
         [ 1.0723, -0.5803, -0.8088,  ...,  1.3814,  2.7166, -0.7876],
         [ 1.0787, -1.2681,  1.4045,  ...,  1.3973, -1.6720,  1.4393]]),
 torch.Size([1000, 50]))
```

```
1  y[:5], y.shape
```

✓ 0.0s

```
(tensor([1., 1., 1., 0., 1.]), torch.Size([1000]))
```

模型：两层的MLP

```python
# 2. 定义模型
class MLP(nn.Module):
    def __init__(self, in_dim, hidden, out_dim):
        super().__init__()
        self.fc1 = nn.Linear(in_dim, hidden)
        self.fc2 = nn.Linear(hidden, out_dim)

    def forward(self, x):
        h = torch.relu(self.fc1(x))
        out = torch.sigmoid(self.fc2(h))
        return out, h

model = MLP(50, 16, 1)
```

利用tensorboard展示训练集和验证集的损失函数、学习率、ROC AUC和PR AUC及其曲线，以及自定义的混淆矩阵

```python
with torch.no_grad():
    y_val_pred, h_val = model(X_val)
    val_loss = criterion(y_val_pred.view(-1), y_val)

    # ROC AUC
    y_val_score = y_val_pred.view(-1).numpy()
    roc_auc = roc_auc_score(y_val.numpy(), y_val_score)

    # PR AUC (average precision)
    pr_auc = average_precision_score(y_val.numpy(), y_val_score)

    # ROC 曲线图
    plot_roc_curve(y_val.numpy(), y_val_score, epoch, writer)

    # Precision-Recall 曲线图
    plot_pr_curve(y_val.numpy(), y_val_score, epoch, writer)

    # 混淆矩阵 (阈值0.5)
    y_val_label = (y_val_score > 0.5).astype(int)
    plot_confusion_matrix(y_val.numpy(), y_val_label, epoch, writer, classes=["Healthy", "Disease"])

# 5. 写入 TensorBoard
writer.add_scalar("Loss/train", loss.item(), epoch)
writer.add_scalar("Loss/val", val_loss.item(), epoch)
writer.add_scalar("Metric/val_ROC_AUC", roc_auc, epoch)
writer.add_scalar("Metric/val_PR_AUC", pr_auc, epoch)
writer.add_scalar("LearningRate", scheduler.get_last_lr()[0], epoch)
```

Tensorboard示例——基因表达数据对疾病二分类

各指标意义：
训练集Loss / 验证集Loss：衡量模型在训练数据 / 新数据上的拟合程度，若训练loss下降而验证loss不降反升则过拟合
学习率：控制每次参数更新的步长，过大则震荡不收敛，过小则收敛慢或陷入局部最优
ROC曲线：横轴为假阳率（FPR），纵轴为真阳率（TPR），数据分布相对平衡时效果好
ROC AUC：ROC曲线下面积，0.5 = 完全随机，1.0 = 完美分类器
PR曲线：横轴为召回率（Recall），纵轴为Precision（精确率），类别不平衡时比ROC更有意义
PR AUC：PR曲线下面积
混淆矩阵：直观展示预测结果与真实标签的对比

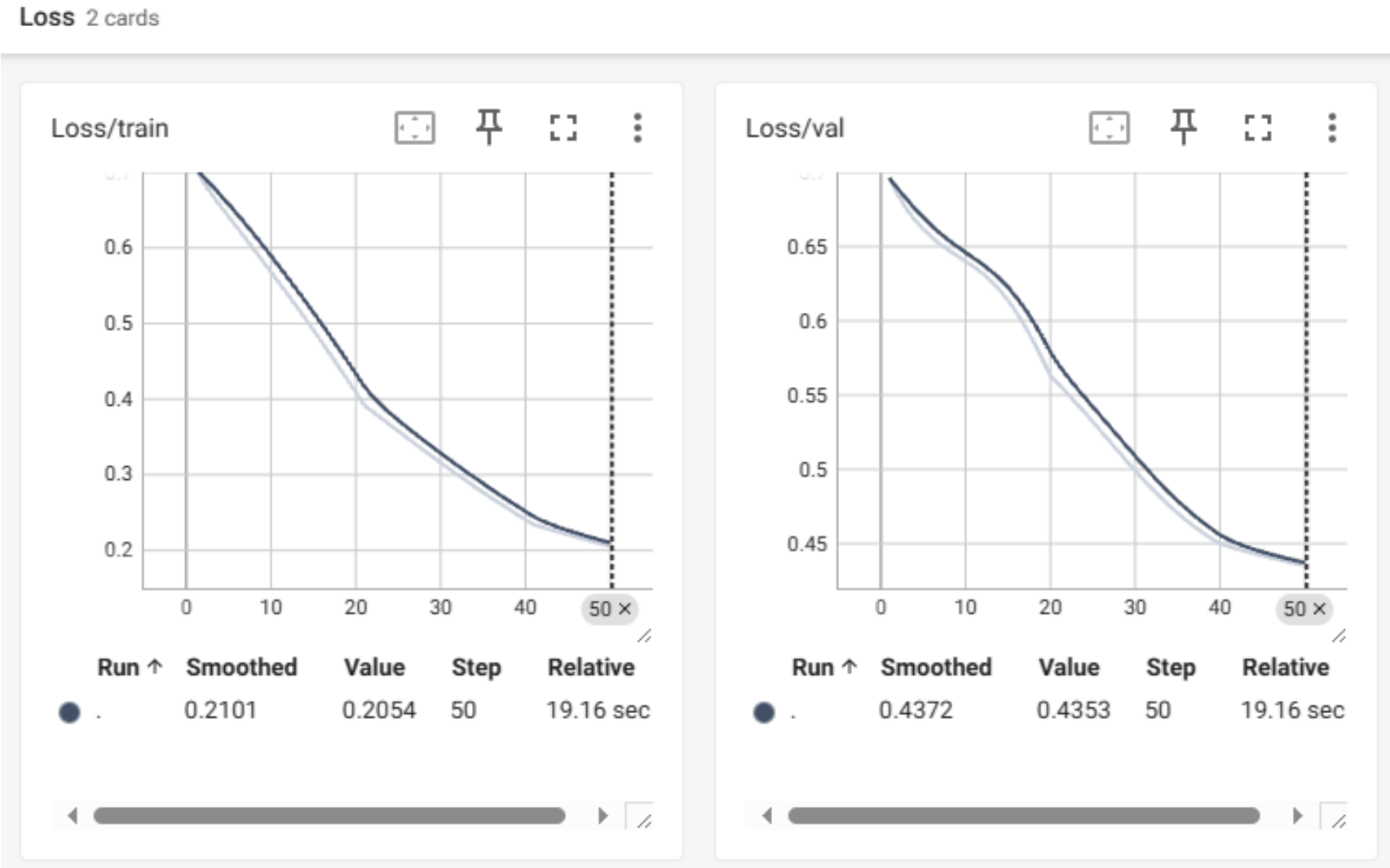
	预测阳性	预测阴性
真实阳性 (TP)	真阳性	假阴性 (FN)
真实阴性 (TN)	假阳性 (FP)	真阴性

从混淆矩阵可以计算：
Precision = TP / (TP+FP)
Recall = TP / (TP+FN)

FN 很多 → 模型漏诊严重（高风险疾病场景很致命）
FP 很多 → 模型误诊严重（可能造成资源浪费）

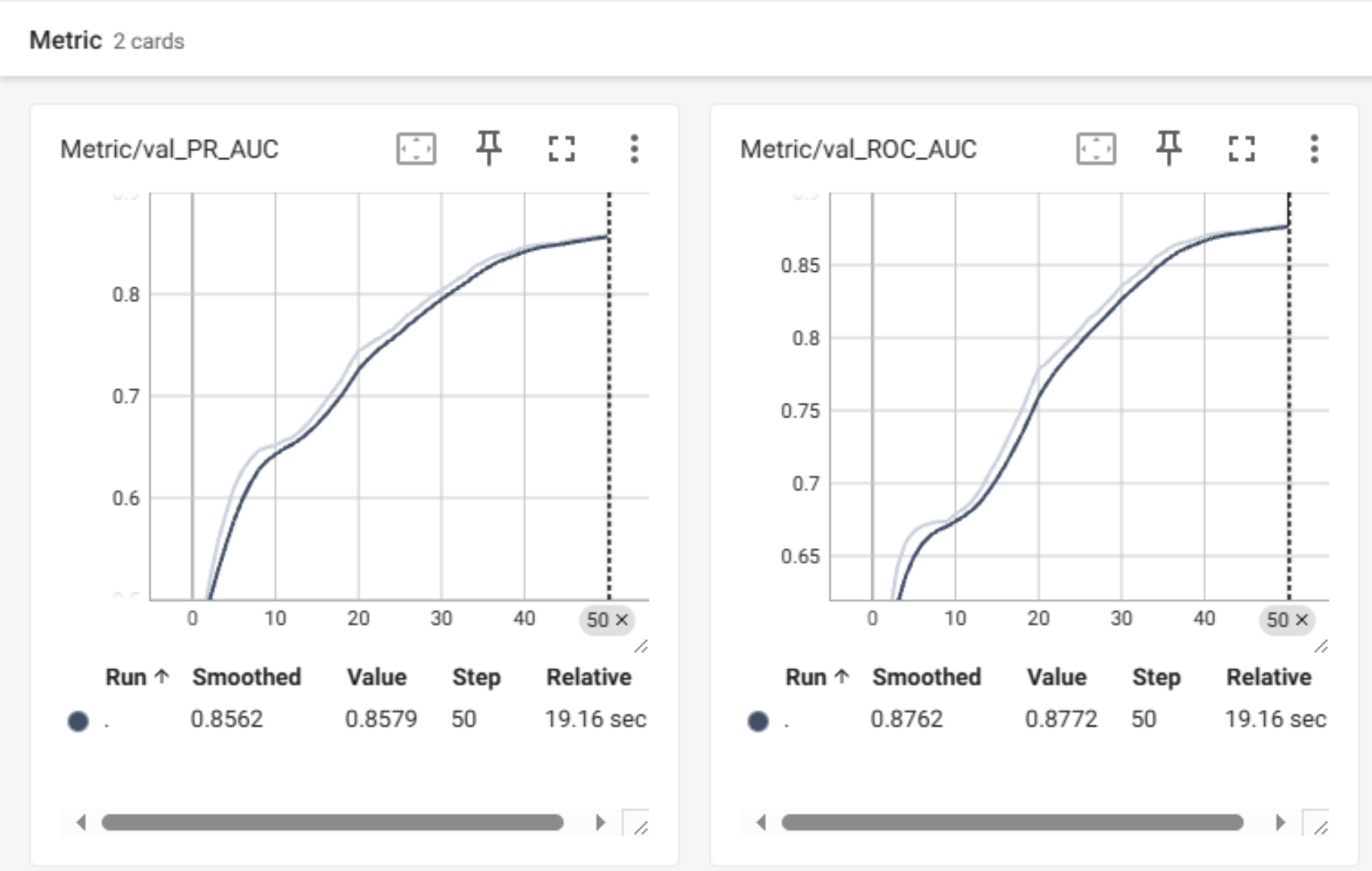
Tensorboard示例——基因表达数据对疾病二分类

训练集和验证集的loss曲线



Tensorboard示例——基因表达数据对疾病二分类

ROC AUC和PR AUC值变化的曲线图



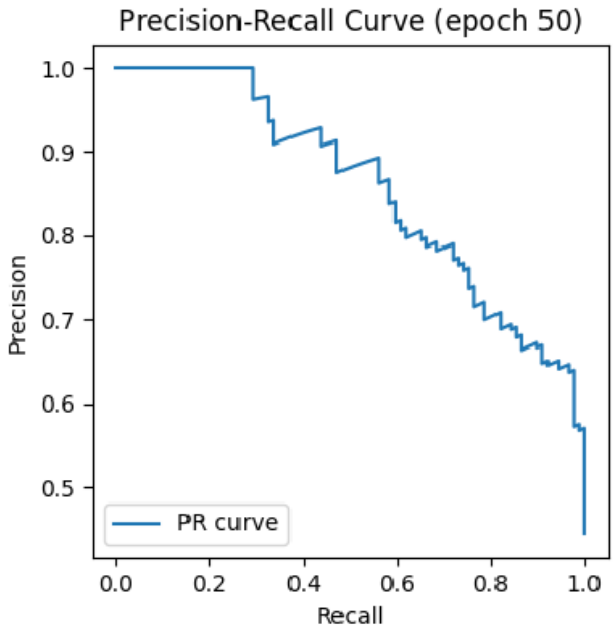
Tensorboard示例——基因表达数据对疾病二分类

每个epoch的PR曲线图和ROC曲线图

Precision_Recall_Curve

Precision_Recall_Curve

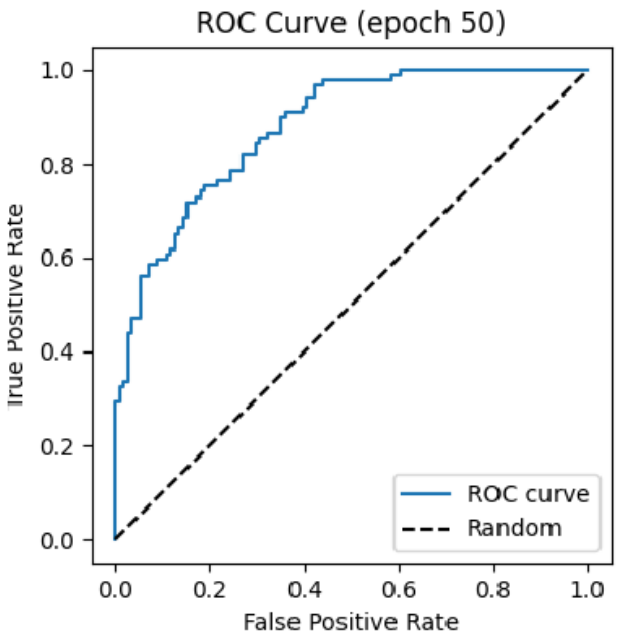
Step 50



ROC_Curve

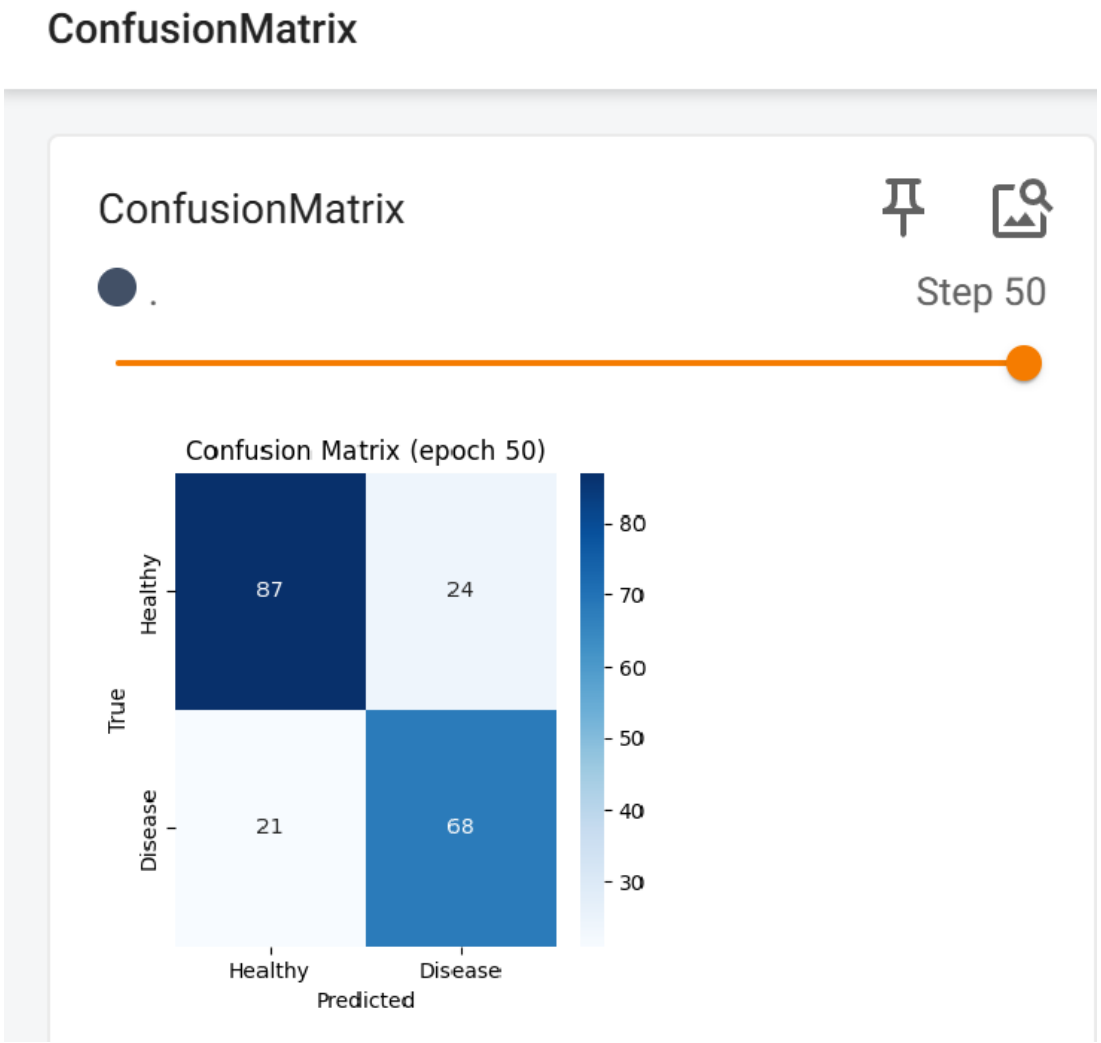
ROC_Curve

Step 50

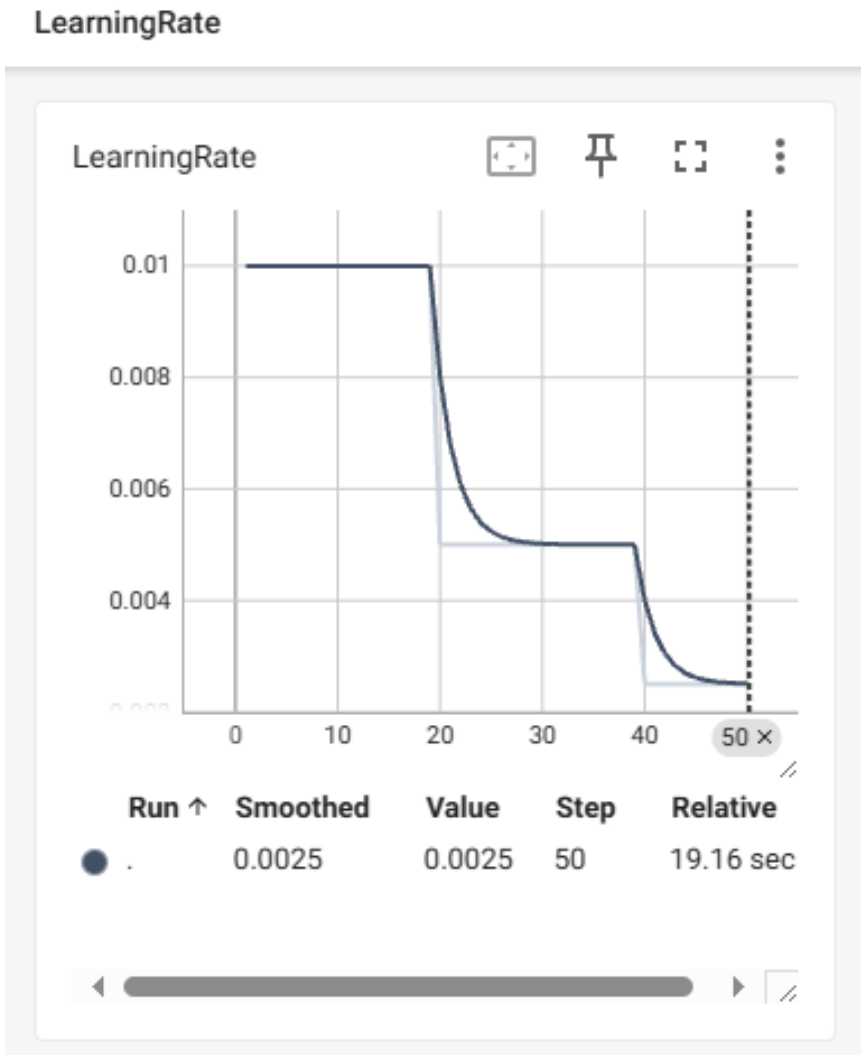


Tensorboard示例——基因表达数据对疾病二分类

每个epoch的混淆矩阵



学习率变化曲线图



模型管理工具——Lightning Module

Lightning Module：一个将深度学习模型训练过程进行高度抽象的python包。
优点：

训练循环标准化：不再手写 optimizer.zero_grad()、loss.backward()、optimizer.step()，框架自动管理
轻松支持 GPU / TPU / 多卡训练：几乎不改模型代码

官方文档：Welcome to ⚡ PyTorch Lightning — PyTorch Lightning 2.5.5 documentation
参考资料：Pytorch Lightning 完全攻略 - 知乎
视频教程：【官方双语】用PyTorch+Lightning优化神经网络代码_哔哩哔哩_bilibili

模型管理工具——Lightning Module

```python
# model
class Net(nn.Module):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x
```

```python
# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
   model.train()
   for batch_idx, (data, target) in enumerate(train_loader):
      data, target = data.to(device), target.to(device)
      optimizer.zero_grad()
      output = model(data)
      loss = F.nll_loss(output, target)

      loss.backward()
      optimizer.step()
      if batch_idx % args.log_interval == 0:
          print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

```python
# model
class Net(LightningModule):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

  def train_dataloader(self):
    mnist_train = MNIST(os.getcwd(), train=True, download=True,
                        transform=transforms.ToTensor())
    return DataLoader(mnist_train, batch_size=64)

  def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
    scheduler = StepLR(optimizer, step_size=1)
    return optimizer, scheduler

  def training_step(self, batch, batch_idx):
    data, target = batch
    output = self.forward(data)
    loss = F.nll_loss(output, target)
    return {'loss': loss}
```

模型定义、数据加载、优化器等差别不大

极大简化训练步骤且易于复用、修改和迁移

Lightning优化疾病分类代码——模型定义

PyTorch

```python
# 2. 定义模型
class MLP(nn.Module):
    def __init__(self, in_dim, hidden, out_dim):
        super().__init__()
        self.fc1 = nn.Linear(in_dim, hidden)
        self.fc2 = nn.Linear(hidden, out_dim)

    def forward(self, x):
        h = torch.relu(self.fc1(x))
        out = torch.sigmoid(self.fc2(h))
        return out, h

model = MLP(50, 16, 1)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

Lightning

```python
# 2. LightningModule
class DiseaseClassifier(pl.LightningModule):
    def __init__(self, input_dim=50, hidden=16, lr=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.fc1 = nn.Linear(input_dim, hidden)
        self.fc2 = nn.Linear(hidden, 1)
        self.criterion = nn.BCELoss()
```

Lightning优化疾病分类代码——模型训练

PyTorch

```python
# 4. 训练循环
for epoch in range(1, 51):
    # ---- Training ----
    model.train()
    optimizer.zero_grad()
    y_pred, h = model(X_train)
    loss = criterion(y_pred.view(-1), y_train)
    loss.backward()
    optimizer.step()
    scheduler.step()

    # ---- Validation ----
    model.eval()
    with torch.no_grad():
        y_val_pred, h_val = model(X_val)
        val_loss = criterion(y_val_pred.view(-1), y_val)
```

Lightning

```python
def training_step(self, batch, batch_idx):
    x, y = batch
    y_hat, _ = self(x)
    loss = self.criterion(y_hat.view(-1), y)
    self.log("Loss/train", loss, prog_bar=True)
    return loss

def validation_step(self, batch, batch_idx):
    x, y = batch
    y_hat, h = self(x)
    loss = self.criterion(y_hat.view(-1), y)
```

Lightning优化疾病分类代码——模型评估

PyTorch

```python
# 辅助函数：绘制混淆矩阵
def plot_confusion_matrix(y_true, y_pred, epoch, writer, classes=None):
# 辅助函数：绘制 ROC 曲线
def plot_roc_curve(y_true, y_scores, epoch, writer):
# 辅助函数：绘制 PR 曲线
def plot_pr_curve(y_true, y_scores, epoch, writer):
# ---- Validation ----
model.eval()
with torch.no_grad():
    y_val_pred, h_val = model(X_val)
    val_loss = criterion(y_val_pred.view(-1), y_val)

    # ROC AUC
    y_val_score = y_val_pred.view(-1).numpy()
    roc_auc = roc_auc_score(y_val.numpy(), y_val_score)

    # PR AUC (average precision)
    pr_auc = average_precision_score(y_val.numpy(), y_val_score)

    # ROC 曲线图
    plot_roc_curve(y_val.numpy(), y_val_score, epoch, writer)

    # Precision-Recall 曲线图
    plot_pr_curve(y_val.numpy(), y_val_score, epoch, writer)

    # 混淆矩阵 (阈值0.5)
    y_val_label = (y_val_score > 0.5).astype(int)
    plot_confusion_matrix(y_val.numpy(), y_val_label, epoch, writer, c
```

夹杂在validation内

Lightning

```python
def validation_epoch_end(self, outputs):
    y_hat = torch.cat([o["y_hat"] for o in outputs]).detach().cpu().numpy()
    y_true = torch.cat([o["y"] for o in outputs]).cpu().numpy()

    # ROC AUC / PR AUC
    roc_auc = roc_auc_score(y_true, y_hat)
    pr_auc = average_precision_score(y_true, y_hat)
    self.log("Metric/val_ROC_AUC", roc_auc, prog_bar=True)
    self.log("Metric/val_PR_AUC", pr_auc, prog_bar=True)

    # 混淆矩阵
    y_label = (y_hat > 0.5).astype(int)
    cm = confusion_matrix(y_true, y_label)
    fig, ax = plt.subplots(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Healthy
    ax.set_xlabel("Predicted")
    ax.set_ylabel("True")
    ax.set_title(f"Confusion Matrix (epoch {self.current_epoch})")
```

单独函数统一管理，无需调用，模块自动管理

Lightning优化疾病分类代码——日志记录

PyTorch

```
# 3. TensorBoard
writer = SummaryWriter(log_dir="runs/bio_demo")
```

```
# ---- Validation ----
model.eval()
with torch.no_grad():
    y_val_pred, h_val = model(X_val)
    val_loss = criterion(y_val_pred.view(-1), y_val)

    # ROC AUC
    y_val_score = y_val_pred.view(-1).numpy()
    roc_auc = roc_auc_score(y_val.numpy(), y_val_score)

    # PR AUC (average precision)
    pr_auc = average_precision_score(y_val.numpy(), y_val_score)

    # ROC 曲线图
    plot_roc_curve(y_val.numpy(), y_val_score, epoch, writer)

    # Precision-Recall 曲线图
    plot_pr_curve(y_val.numpy(), y_val_score, epoch, writer)

    # 混淆矩阵 (阈值0.5)
    y_val_label = (y_val_score > 0.5).astype(int)
    plot_confusion_matrix(y_val.numpy(), y_val_label, epoch, writer, c
```

```
# 5. 写入 TensorBoard
writer.add_scalar("Loss/train", loss.item(), epoch)
writer.add_scalar("Loss/val", val_loss.item(), epoch)
writer.add_scalar("Metric/val_ROC_AUC", roc_auc, epoch)
writer.add_scalar("Metric/val_PR_AUC", pr_auc, epoch)
writer.add_scalar("LearningRate", scheduler.get_last_lr()[0], epoch)
```

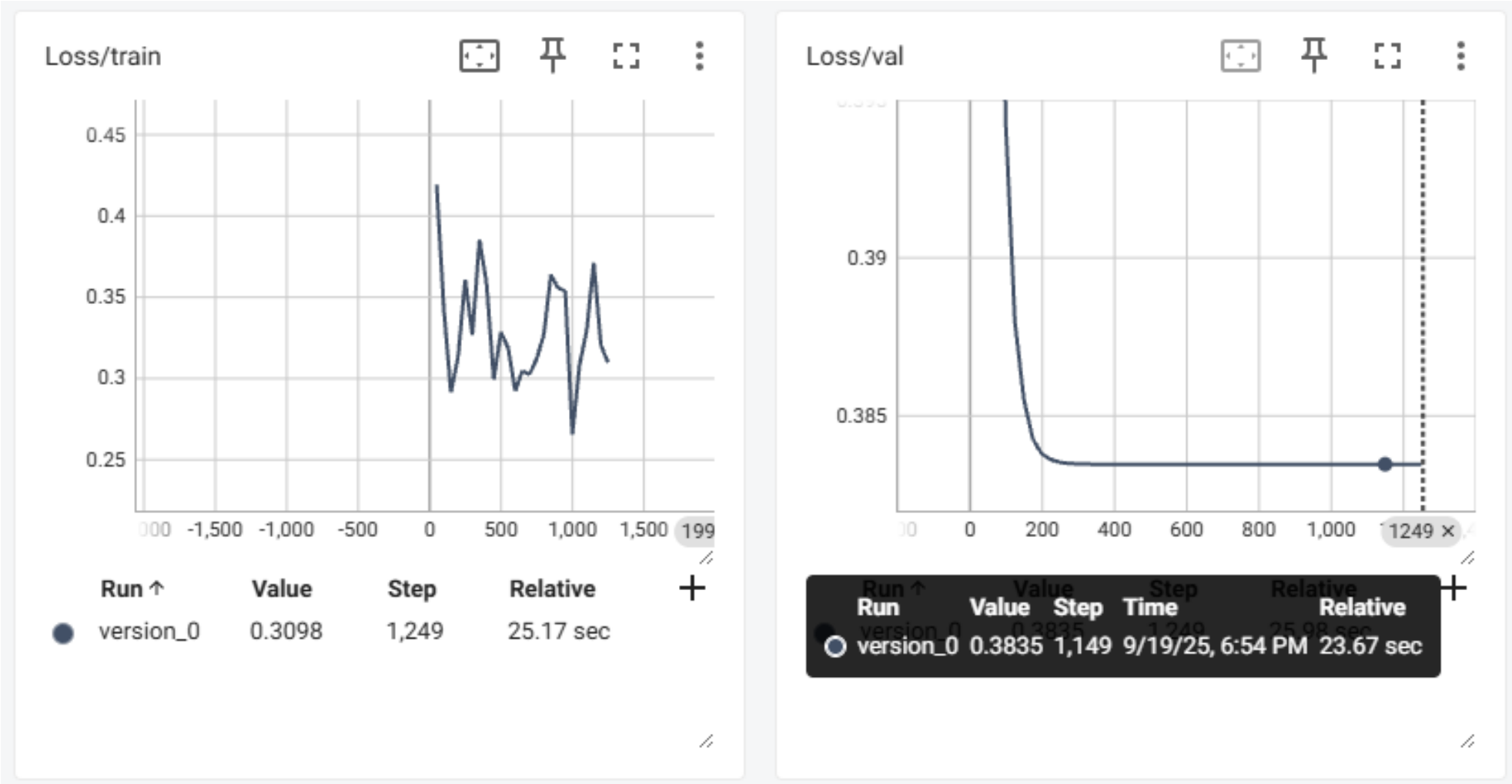
声明后一个一个手动记录

Lightning

```
def validation_epoch_end(self, outputs):
    y_hat = torch.cat([o["y_hat"] for o in outputs]).detach().cpu().numpy()
    y_true = torch.cat([o["y"] for o in outputs]).cpu().numpy()

    # ROC AUC / PR AUC
    roc_auc = roc_auc_score(y_true, y_hat)
    pr_auc = average_precision_score(y_true, y_hat)
    self.log("Metric/val_ROC_AUC", roc_auc, prog_bar=True)
    self.log("Metric/val_PR_AUC", pr_auc, prog_bar=True)

    # 混淆矩阵
    y_label = (y_hat > 0.5).astype(int)
    cm = confusion_matrix(y_true, y_label)
    fig, ax = plt.subplots(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Healthy","D
    ax.set_xlabel("Predicted")
    ax.set_ylabel("True")
    ax.set_title(f"Confusion Matrix (epoch {self.current_epoch})")
    self.logger.experiment.add_figure("ConfusionMatrix", fig, self.current_epoch
    plt.close(fig)
logger = TensorBoardLogger("runs", name="bio_demo_lightning")

model = DiseaseClassifier()
trainer = pl.Trainer(max_epochs=50, accelerator="auto", devices=1, logger=logger)
trainer.fit(model, train_loader, val_loader)
```

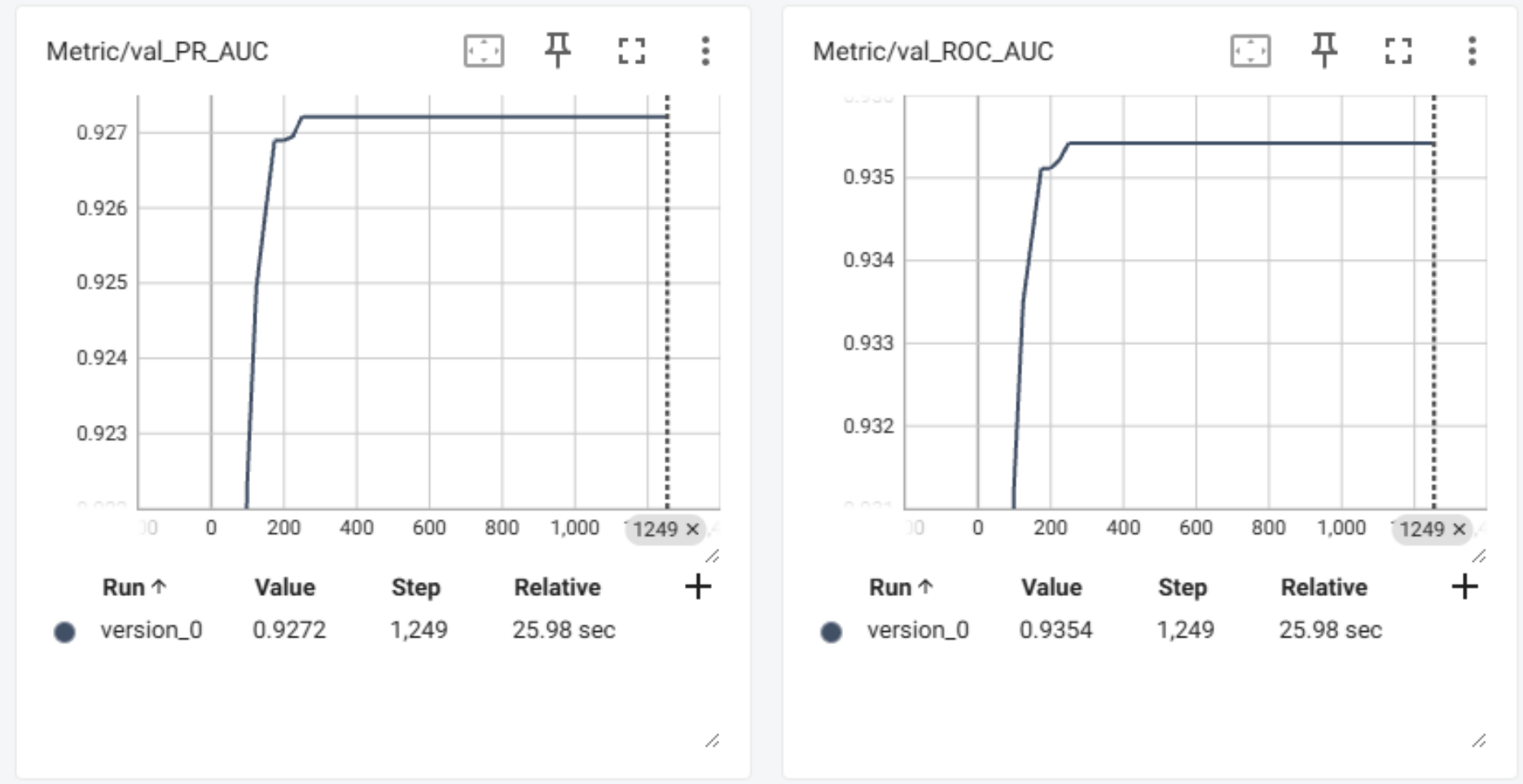
传参即可

TensorBoard+Lightning——疾病分类结果



训练数据分布会导致训练集Loss震荡，总体趋势是下降即可，主要看验证集Loss

TensorBoard+Lightning——疾病分类结果



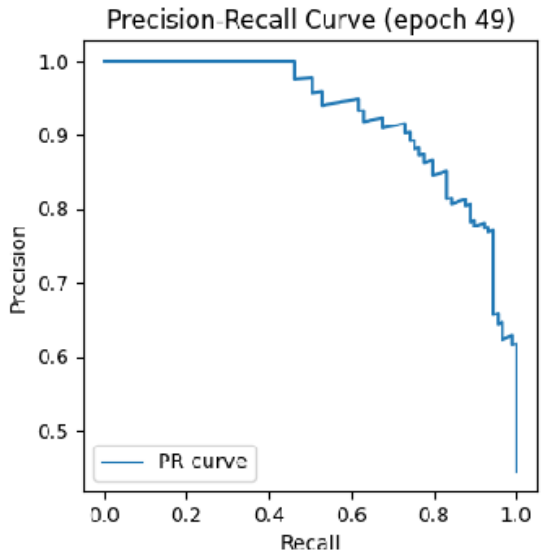
由于训练数据较小，后期auc基本不变
另外1250step = train data(800) / batchsize(32) * epoch(50)

TensorBoard+Lightning——疾病分类结果

Precision_Recall_Curve

version_0

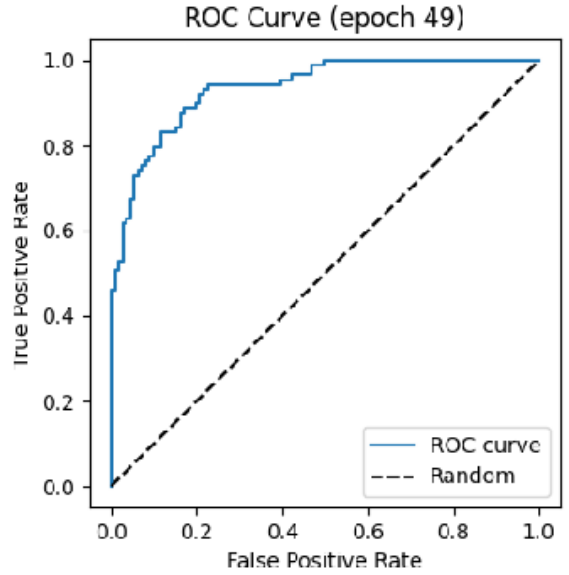
Step 49



ROC_Curve

version_0

Step 49



随堂小测 & 课外学习

- 作业：
  - 1.
  - 2.
- 参考资料：
  - LightningModule官方文档： PyTorch Lightning 2.5.5 documentation
  - 参考资料：Pytorch Lightning 完全攻略 - 知乎
  - 视频资料：【官方双语】用PyTorch+Lightning优化神经网络代码_哔哩哔哩_bilibili