手写数字识别报告

姓名：宋正非

学号：2151094

专业：自动化

2023.12.16

目录

一、 实验目标

本实验目标是通过搭建卷积神经网络并对其训练，以实现从手写数字图像中识别和分类具体数字的功能。

手写数字识别目前被广泛应用于支票处理、自动化银行服务、手写数字输入设备等领域。在深度学习的兴起下，尤其是卷积神经网络的不断应用，手写数字的识别取得了显著的进展。手写数字识别在现代社会的许多领域都发挥着作用，推动了许多行业的数字化转型。随着技术的不断发展，手写数字识别的应用场景将进一步拓展，为更多领域带来创新和便利。

二、 MNIST 数据集

MNIST(Modified National Institute of Standards and Technology)数据集是一个广泛用于计算机视觉领域的手写数字识别任务的数据集，由 60000 个训练样本和 10000 个测试样本组成，包含了 0 至 9 共 10 类手写数字图片，每个样本都是一张 28 * 28 像素的灰度手写数字图片。

MNIST 数据集经常被用作深度学习初学者的入门任务，因为它相对较小，训练相对迅速，同时具有足够的复杂性。

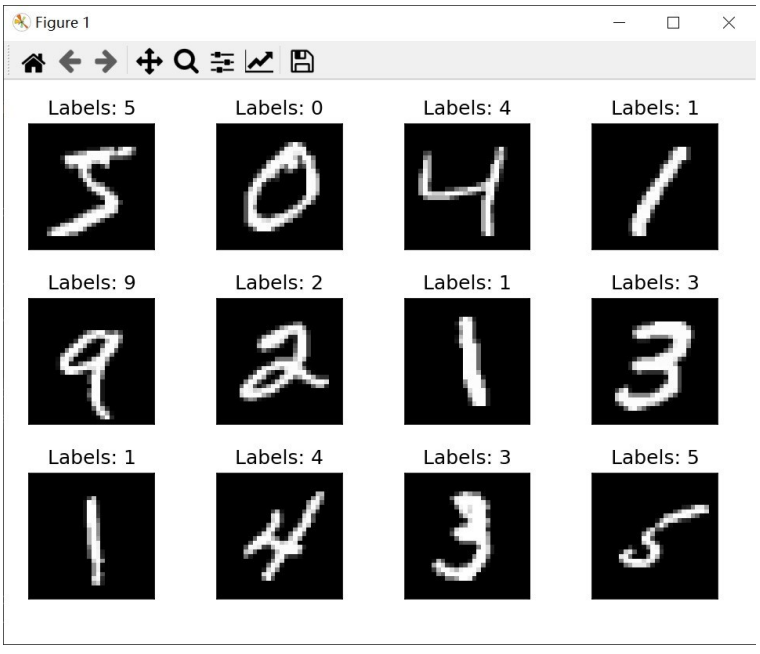

图 1 数据集样例

三、 实验流程

1. 定义超参数

首先，定义超参数，即用于定义模型结构或优化策略的参数；包括每批训练处理的样本数量 batch size、训练数据集的轮次 epoch 以及训练设备的选择。本实验将超参数设定为：batch_size=128, epoch=30, 并选择 device 为 GPU。

2. 处理数据集

下载数据集，将数据集分为训练集与测试集两类。同时对数据集进行处理转换，将数据集中的样本转为张量形式并进行归一化处理，实现数据增强，从而提高模型的泛化能力和效果，减少模型对某些特征的过度依赖从而避免过拟合。

根据每个轮次的样本数量加载数据集并打乱。

3. 构建网络模型

典型的 CNN 架构包括卷积层(Convolutional Layer)、池化层(Pooling Layer)、全连接层(Fully Connected Layer)和激活函数(Activation Function)。 卷积层用于通过卷积操作从输入中提取图像局部特征；激活函数用于引入非线性变换；池化层用于对特征图进行降维，以降低计算量并提高模型的鲁棒性；全连接层用于将特征映射到类别上。

本实验网络结构主要由三部分构成，具体如下：

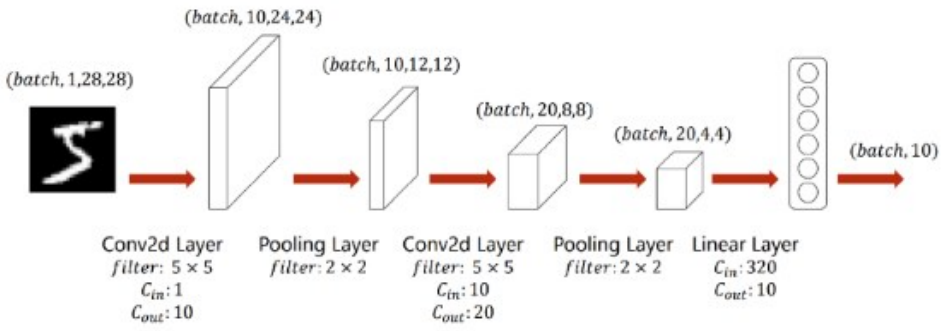

图 2 网络框架

首先使用一个 5×5 的卷积层，图像通道由 1 变为 10、高宽由 28 变为 24。经过 ReLU 激活后，通过核为 2×2 的池化层，使用最大池化法池化后，通道数不变，高宽变为一半。经过上述操作，图像维度由(batch_size, 1，28，28)变为(batch_size, 10，12，12)。

随后再经过一个 5×5 的卷积层，将图像通道由 10 变为 20、高宽由 12 变为 8。经过 ReLU 激活后，通过核为 2×2 的池化层，使用最大池化法，使得通道数不变，高宽变为一半。将图像维度变为(batch_size, 20，4，4)。

最后再将其展平后进入全连接层，使用线性函数输出为 10 类，对应 0 至 9 共 10 个数字。

4. 训练过程

使用训练集对构建好的网络模型进行训练。训练过程主要包括前向传播(Forward Propagation)与后向传播(Backward Propagation)。前向传播为，输入图像通过卷积、池化和全连接层最终得到分类结果；反向传播则根据损失函数(Loss)计算梯度，并使用优化算法更新网络参数，最小化损失函数。

其中，损失函数选择交叉熵损失函数(Cross Entropy Loss)。其定义为：

$$L = -\frac{1}{N}\sum_{i\in N}\sum_{c=1}^{M} y_{ic}\log(p_{ic})$$

优化器(Optimizer)选择 Adam(Adaptive Moment Estimation)优化器，其实现简单，计算高效，能够自动调整学习率(learning rate)，工作性能较优。其中，本实验将学习率设定为 0.0005。

在训练过程中，每一轮训练后通过在测试集上评估得到损失与精确度结果，并使用 TensorBoard 对训练曲线进行绘制，横坐标为训练的轮次，即 30。同时使用 logging 记录训练日志，并以 json 格式输出以记录训练中的模型情况。

四、 模型评估

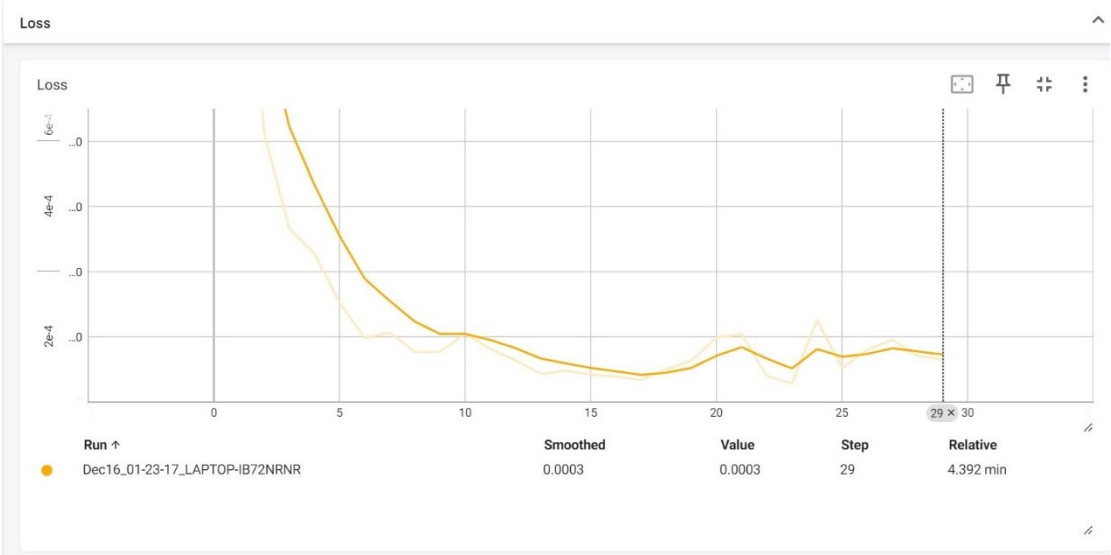

图 3 Loss 曲线

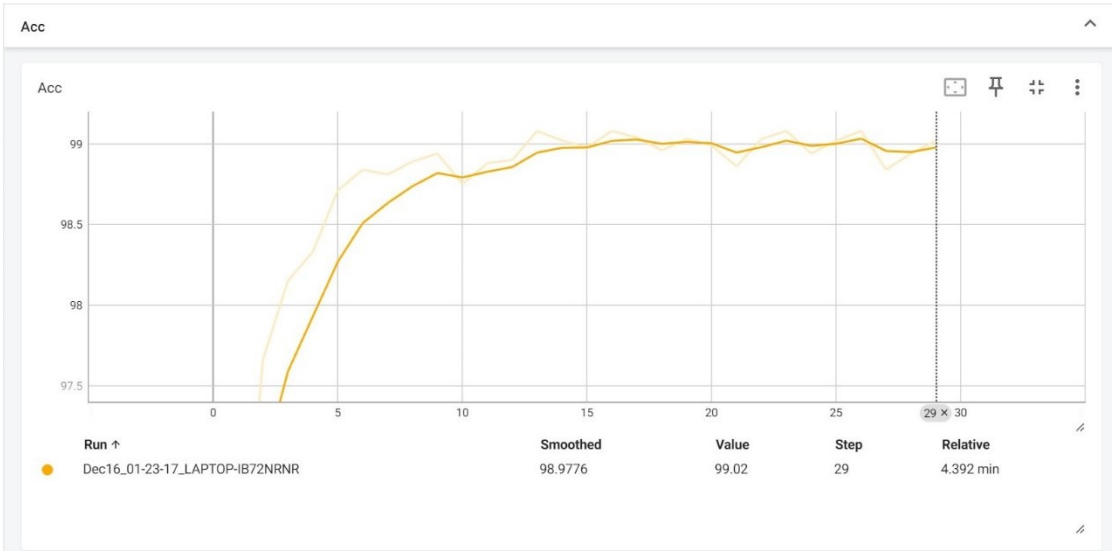

图 4 Accuracy 曲线

从上图可看出，损失值整体较小，随着训练轮次的增加而减小，在 10 轮前降低速度较快，随后降低速度放缓并趋于 $1e^{-4}$ 左右。准确值反应测试集上正确分类的样本数与所有样本数的比值，其整体较高，在训练前 10 轮增长较为迅速，随后在 99%附近震荡。

整体而言，模型的准确度较高、损失值较低，手写数字识别的性能满足预期。

五、 识别效果

在得到模型文件 digit_recognition_model.pt 后，制作识别效果的 demo，效果图如下。

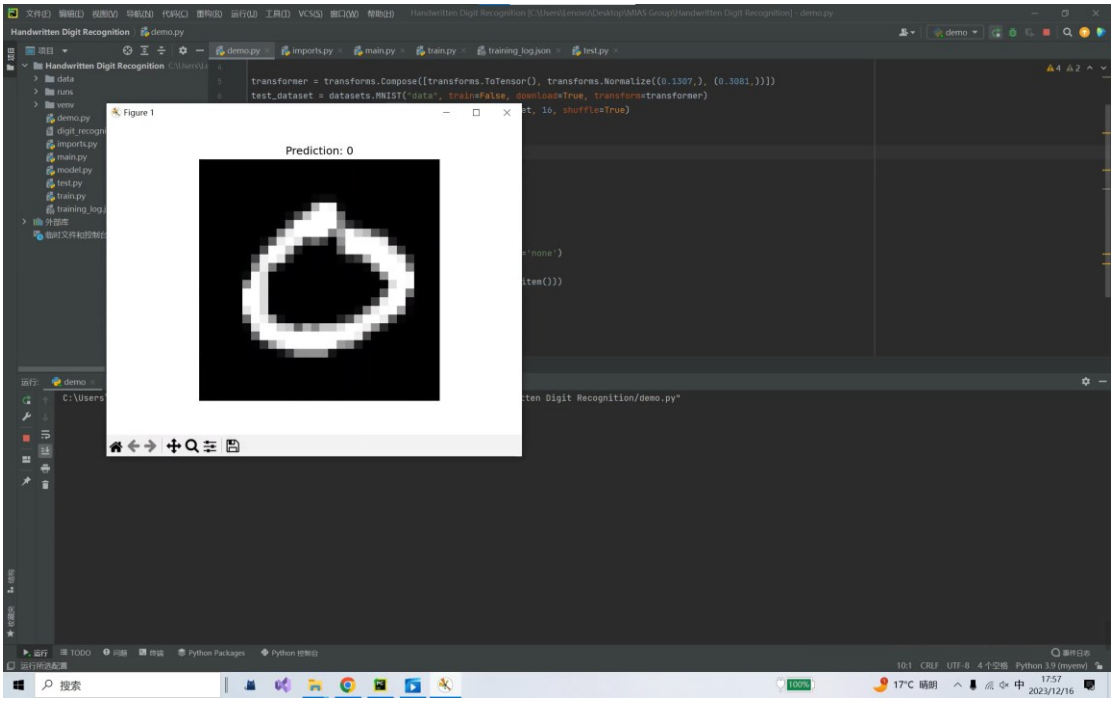

图 5 识别预测样例

六、 心得

1. 参数影响

起初，准确度达到 95%以上，但得到的 Loss 曲线在训练后半期随着轮次增加而上升，如图 6。可能原因包括 batch_size 设置不合适，过小导致训练速度慢且训练不收敛，但分类类别较多时，可能会使得网络有明显的震荡。因此，将 batch_size 由 32 调整为 128。由图 8 可见，整体损失值明显降低，但由图 7 可知，其仍出现在后半阶段随训练轮次增加损失值上升的问题。故对学习率进行调整。

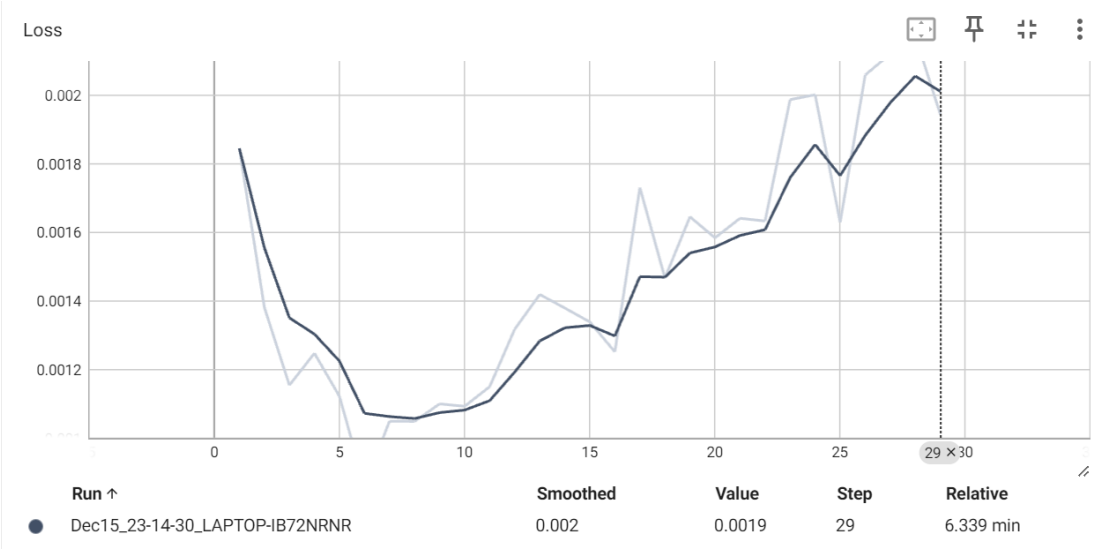

图 6 Loss 曲线（batch_size=32）

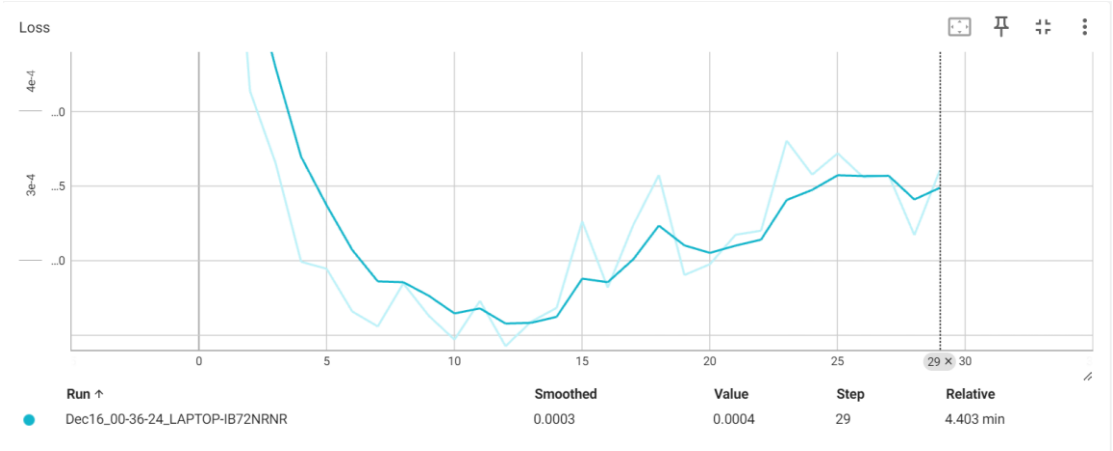

图 7 Loss 曲线（batch_size=128）

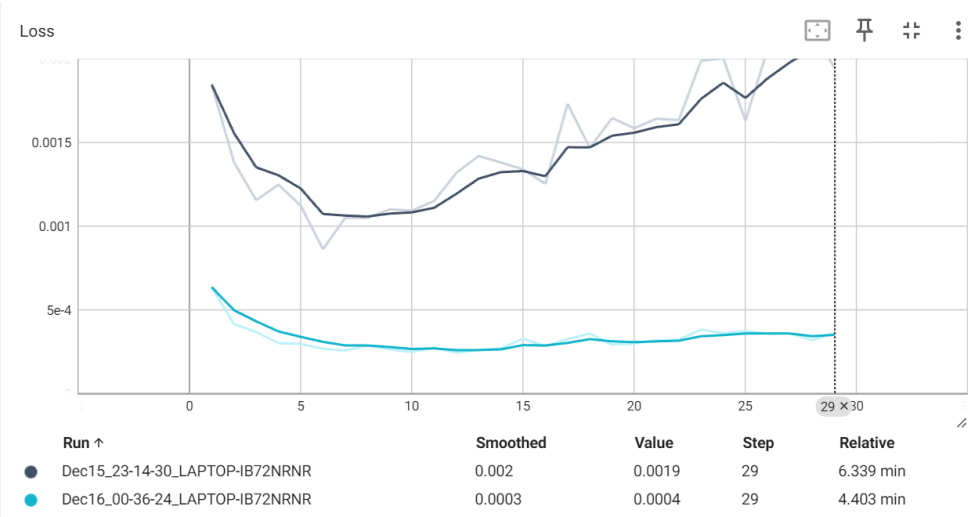

图 8 Loss 曲线对比

原来的学习率为默认的 0.001，可能存在学习率太大的问题。故改为 0.000001 进行尝试，此时损失值变化随训练轮次下降（图 9），但精确度不足（图 10）。

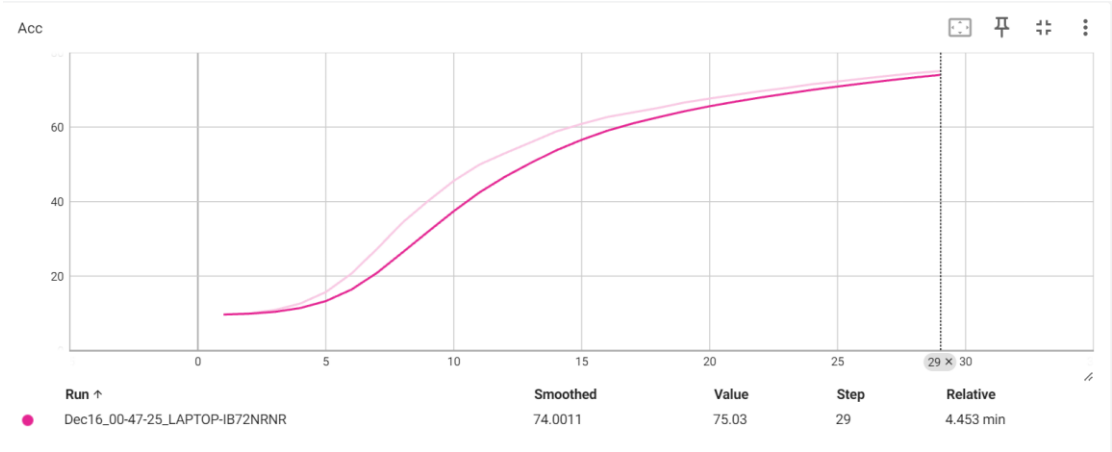

图 9 Accuracy 曲线（learning rate=0.000001）

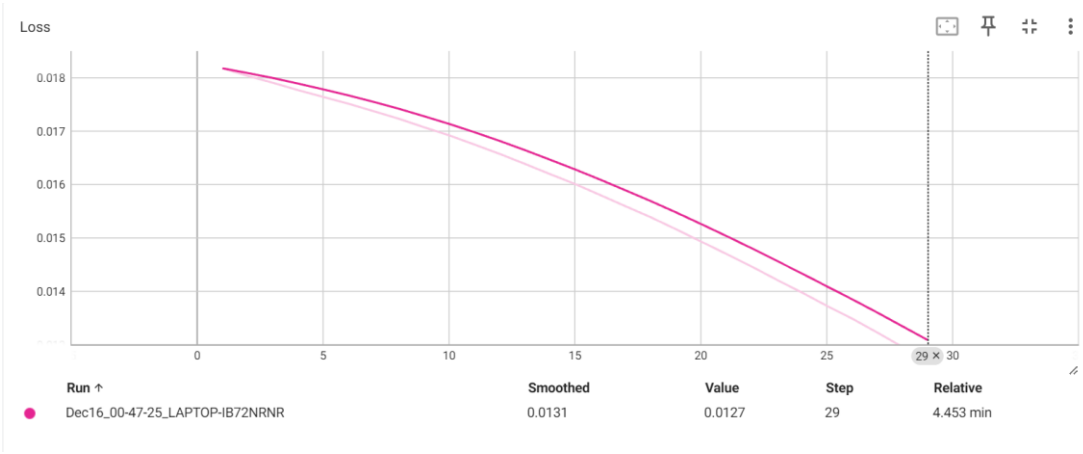

图 10 Loss 曲线（learning rate=0.000001）

将学习率改为 0.00001，此时精确度有所上升，但仍未达 90%（图 11）。

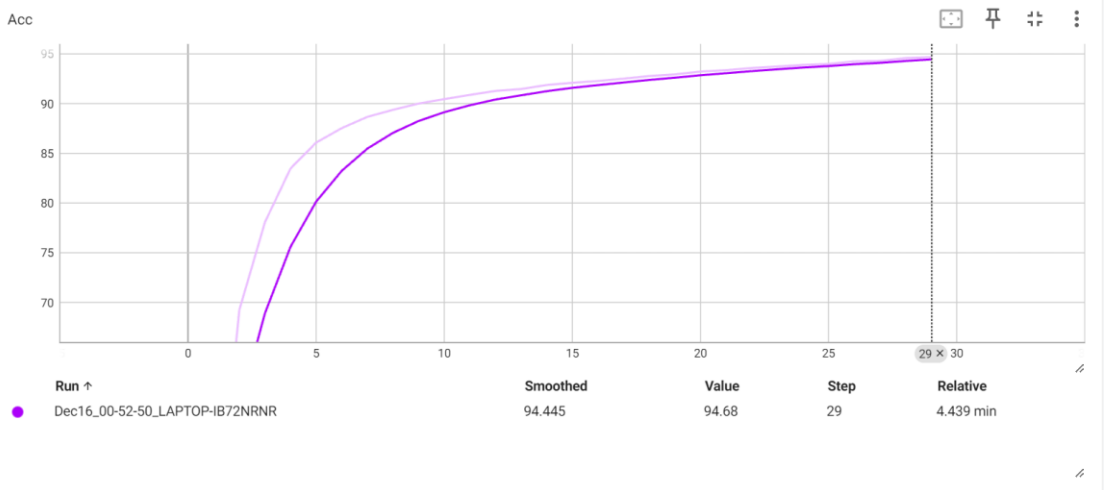

图 11 Accuracy 曲线（learning rate=0.00001）

最终，将学习率继续增大调整为 0.0005 时，得到图 3、图 4 所示的损失与精确度曲线。

2. 问题解决

本实验的进度有几天卡在了使用 TensorBoard 上。起初查到的资料都是需要 tensorflow 框架的，但我之前使用的是 pytorch 因此环境中没安装过 tensorflow。然后在摸索之中由于误操作把之前用的 python 解释器变得 invalid 了。随后也没找到办法改变。因此又重新新建了环境，但是因为 pytorch、tensorflow、python、cuda 之间的版本不匹配一直没解决（cuda 是 11.8 的，要安装 torch 对应的版本就需要 python 的版本比较高，但是这对于匹配的 tensorflow 来说版本又太高了）。我在思考如何解决这个问题的时候也觉得在同一个环境下同时需要安装 pytorch 与 tensorflow 有些奇怪。

之后，在重新搜索有没有别的办法时，才发现在 torch 下就可以使用 TensorBoard，不需要大费周章地安装 tensorflow。这一款基于浏览器的机器学习可视化工具也确实比之前调用 matplotlib 绘制曲线更方便。

导致这个问题的原因是我在信息没有检索全的情况下就开始操作，在以后的实验中需要注意避免。