Lab-01 | 基于CNN的图像分类

姓名：张英祺　　学号：1120222198

一、实验目标

本次实验通过使用 Pytorch 深度学习框架完成 GTSRB 图像分类任务，使同学们进一步熟悉 Pytorch 训练框架，掌握 Pytorch 中的数据加载、数据增强实现、卷积神经网络搭建、分类网络训练、分类模型测试以及模型优化等内容。本实验基于卷积神经网络实现交通标志图像的分类，与传统图像分类方法不同，卷积神经网络无需人工提取特征，可以根据输入图像，自动学习包含丰富语义信息的特征，得到更为全面的图像特征描述，可以很好地表达图像的不同类别信息。

二、实验内容

1. 掌握Pytorch中加载数据的方法，根据提供的图片和标签，加载GTSRB数据。
2. 掌握Pytorch中卷积神经网络的构建方法，搭建合适的网络结构用于GTSRB数据集图像分类任务。
3. 掌握Pytorch中损失函数和优化器的定义方法，并针对图像分类任务进行合适的选择。
4. 掌握Pytorch中模型训练的步骤，学会调整参数来提高模型的训练效果。

三、实验原理

1. CNN 与图像分类任务

卷积神经网络 (CNN) 是一种深度学习模型，在图像处理任务中有着广泛的应用。由于其结合了神经网络的能力和卷积层的局部感知特性，可有效从图像中提取特征并实现准确分类。卷积神经网络的三个基本组件包括卷积层、池化层和输出层，输入图像先通过多步卷积运算提取特征，最后通过全连接层将输出的特征图进行降维，并将学到的特征表示映射到样本标记空间进行分类。

2. 卷积层的设计

在设计 CNN 网络时，我们需要考虑各层之间如何进行搭配，通常采用 "卷积层+激活函数+池化层" 作为一个模块，最后添加线性层进行分类。激活函数通常使用 ReLU、PReLU 或 ELU 等，池化操作常用最大池化或均值池化。为保持卷积前后的尺寸不变，假设输入大小为 $n \times n$ ，卷积核的尺寸 (kernel size)、步长 (stride) 和填充 (padding) 应满足 $n_{output} = \dfrac{n_{input} - kernelsize + 2 \times padding}{stride} + 1$ 。对于彩色图像，输入通道数为 3，输出通道数等于卷积核的数量。

3. 实验数据集 GTSRB

GTSRB (German Traffic Sign Recognition Benchmark) 数据集由德国弗劳恩霍夫研究所于 2011 年发布，包含超过 50,000 张交通标志图像，涵盖 43 种不同的交通标志类别。该数据集分为训练集 (按标签分类) 和测试集 (未给出标签)，每个标签下包含一个 csv 文件，记录图像名称、大小、Roi 坐标和 classid。

四、实验过程

1. 实验环境

本项目使用的编程语言为 Python 3.7.16，用到的外部库主要包括 Pyplot, Numpy, Pandas, Pytorch, scikit-learn, ipykernel 等。建议使用 conda 创建一个新的虚拟环境来运行该项目。环境配置方法如下所示：

1. 创建 conda 环境并将其命名为 dl，选择 Python 版本为 3.7.16。
2. 激活 conda 环境，使用 pip 读取 requirements.txt 安装本实验用到的所有代码库。
3. 使用 conda list 检查虚拟环境下的代码库是否齐全。

实验代码在 ImageClassification.py 和 utils.py 中，使用 JupyterLab 进行展示，程序入口为 main.ipynb。

2. 数据加载与预处理

数据集包含 Training 和 Final_Test 两个文件夹，图片以 .ppm 格式保存，因此可以使用 PIL 提供的 Image 类进行读取和处理。该数据集共包含 43 个分类，读取图片后统一 resize 为 (60,60) 大小，最后返回 data:list 和 label:list (直接从 csv 文件中获取)。需要注意本实验使用了虚拟环境运行，因此目标路径采用的是绝对路径的方式，定位方式如下所示。

由于本数据集提供了图片的 Roi 信息，在预处理阶段，我们可以先根据 Roi 对图片进行裁剪放大，这样可以有效提高训练精度，下述代码展示了如何使用 pandas 库读取 csv 文件中的 Roi 信息并对图片进行预处理。实验代码中包括了 loadTrainData() 和 loadTrainData_Roi() 两种方法，其中普通方法读取耗时约 12 秒，根据 Roi 裁剪的方法耗时约 500 秒。

加载数据集 Python

```python
import os
import pandas as pd
from PIL import Image

# read the image file
def loadTrainData_Roi(image_size:tuple=(28,28), showExample=False):
    data = []
    label = []
    classes = 43
    project_path = os.path.abspath(os.path.join(os.getcwd(), '.')) + "\\GTSRB"
    _average_size = [0,0]

    for i in range(classes):
        path = os.path.join(project_path, "Training\\{:05d}".format(i))
        images = os.listdir(path)

        _csv = pd.read_csv(path + "\\" + images[len(images)-1], sep=';', encoding="utf-8")
        label = label + _csv['ClassId'].values.tolist()
        _Roi_X1 = _csv['Roi.X1'].values.tolist()
        _Roi_Y1 = _csv['Roi.Y1'].values.tolist()
        _Roi_X2 = _csv['Roi.X2'].values.tolist()
        _Roi_Y2 = _csv['Roi.Y2'].values.tolist()
        _sum_size = [0,0]

        # print(path)
        for i in range(len(images)):
            try:
                _image = Image.open(path + "\\" + images[i])
                _image = _image.crop((_Roi_X1[i],_Roi_Y1[i],_Roi_X2[i],_Roi_Y2[i]))
                _sum_size[0] = _sum_size[0] + _Roi_X2[i] - _Roi_X1[i]
                _sum_size[1] = _sum_size[1] + _Roi_Y2[i] - _Roi_Y1[i]

                _image = _image.resize(image_size)
                _image = np.array(_image)
                data.append(_image)
            except:
                _average_size[0] = (_average_size[0] + _sum_size[0]/len(_sum_size))/2
                _average_size[1] = (_average_size[1] + _sum_size[1]/len(_sum_size))/2

    print("data={}, label={}, trainingset={}, average_size={}".format(len(data), len(label), len(_csv), _average_size))

    if showExample:
        _csv.head()
        plt.imshow(data[0]), plt.title("Example:label[{}]:".format(label[0]))

    return data, label
```

获取到训练图片后，我们还要进一步对训练集进行划分。此处采用的是 sklearn.model_selection 下提供的划分方法，首先将 list 数据转为 np.array，按照 20% 的验证集进行划分，将输出结果转为 Tensor 矩阵形式。

注意此处矩阵需要进行转置以符合 nn.Conv2d 的输入，下文会进一步详细说明。

数据预处理 Python

```python
import numpy as np
import random
from sklearn.model_selection import train_test_split

def seperateDataset(data:list, label:list):
    # shuffle the dataset
    randnum = random.randint(0,100)
    random.seed(randnum)
    random.shuffle(data)
    random.seed(randnum)
    random.shuffle(label)

    # Converting lists into numpy arrays
    data = np.array(data)
    label = np.array(label)
    X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.2, random_state=42)

    # Array -> Tesor
    X_train = torch.FloatTensor(X_train)
    y_train = torch.LongTensor(y_train)
    X_test = torch.FloatTensor(X_test)
    y_test = torch.LongTensor(y_test)
    # X_train = X_train.unsqueeze(1)
    # X_test = X_test.unsqueeze(1)

    # Tensor转置 -> nn.Conv2d Tensor(batch_size,channels,height,width)
    X_train = X_train.permute(0, 3, 1, 2)
    X_test = X_test.permute(0, 3, 1, 2)

    print("X_train={}, X_test={}\ny_train={}, y_test={}".format(X_train.shape, X_test.shape, y_train.shape, y_test.shape))

    return X_train, X_test, y_train, y_test
```

3. 网络搭建

分割好数据集后，我们在 ImageClassification.py 中定义网络结构。本实验所用的 CNN 网络采用两个卷积层 + MaxPool 池化层，并选择 nn.ReLU() 作为激活函数。卷积层输出通过 Dropout 和 Flatten 正则化层，最后送入线性层进行分类，具体参数如下所示：

网络搭建 Python

```python
import torch
import torch.nn as nn

# construct CNN model
class Net(nn.Module):
    def __init__(self, image_size:tuple=(28,28), img_channels=3, classes=43):
        super().__init__()

        self.layer1 = nn.Sequential()
        self.layer1.add_module("conv", nn.Conv2d(in_channels = img_channels,
                                                 out_channels = 32,
                                                 kernel_size = 5,
                                                 stride = 1,
                                                 padding = 2))
        self.layer1.add_module('relu', nn.ReLU())
        self.layer1.add_module('pool', nn.MaxPool2d(kernel_size = 2, stride = 2))

        self.layer2 = nn.Sequential()
        self.layer2.add_module("conv", nn.Conv2d(in_channels = 32,
                                                 out_channels = 64,
                                                 kernel_size = 5,
                                                 stride = 1,
                                                 padding = 2))
        self.layer2.add_module('relu', nn.ReLU())
        self.layer2.add_module('pool', nn.MaxPool2d(kernel_size = 2, stride = 2))

        self.dropout = nn.Dropout2d(p = 0.5)
        self.flatten = nn.Flatten(start_dim = 1, end_dim = 3)

        self.fc1 = nn.Linear(in_features = 4*image_size[0]*image_size[1], out_features = 1000)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(in_features = 1000, out_features = classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        # print("conv:{}".format(x.shape))

        x = self.flatten(x)
        x = self.dropout(x)
        # print("flatten:{}".format(x.shape))

        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        # print("output:{}".format(x.shape))

        return x
```

4. 定义损失函数和优化器

本实验是多分类任务，因此选用交叉熵 nn.CrossEntropyLoss() 作为损失函数。在优化器方面，经过横向对比，选择了效果较好的 Adam 优化器。

损失函数和优化器 Python

```python
def cnn(x_train, y_train, x_test, y_test):
    BATCH_SIZE = 100
    model = Net()
    loss_function = nn.CrossEntropyLoss() # 多分类任务
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
    train(x_train, y_train, x_test, y_test,
          BATCH_SIZE, model, loss_function, optimizer)
```

5. 模型训练与验证

将 batch_size、epoch_num 和 val_num 作为参数传入，将训练集按批次划分后，打乱各批次之间的顺序。对每个 epoch 中的每个 batch，计算前向传播、loss 和反向传播，并记录 loss_rate:list。经过每 val_num 个回合进行一次验证，并记录运行时间和准确率 acc_rate:list。

模型训练与验证 Python

```python
import torch
import time
from random import shuffle

# define training function
def train(x_train, y_train, x_test, y_test, model,
          loss_function, optimizer,
          BATCH_SIZE:int = 64, EPOCH_NUM:int = 40, VAL_NUM:int = 2,
          output_log = False):

    train_N = x_train.shape[0]
    loss_rate = []
    acc_rate = []

    _begin = time.time()
    for epoch in range(1,EPOCH_NUM+1):

        _batchindex = list(range(int(train_N / BATCH_SIZE)))
        shuffle(_batchindex)

        for i in _batchindex:
            batch_x = x_train[i*BATCH_SIZE: (i+1)*BATCH_SIZE]
            batch_y = y_train[i*BATCH_SIZE: (i+1)*BATCH_SIZE]

            y_hat = model(batch_x)
            loss = loss_function(y_hat, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        loss_rate.append(loss.item())

        # test
        if epoch % VAL_NUM == 0:
            _end = time.time()
            y_hat = model(x_test)
            y_hat = torch.max(y_hat, 1)[1].data.squeeze()
            acc = torch.sum(y_hat == y_test).float() / y_test.shape[0]
            acc_rate.append(acc.item())

            print(f"[{time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())}] epoch {epoch} | loss:{loss:.4f} | acc:{acc:.4f} | time:{(_end-_begin):.2f}s")
            _begin = time.time()

    print("Finished Training! BATCH_SIZE={}, EPOCH={}, VAL_NUM={}".format(BATCH_SIZE, EPOCH_NUM, VAL_NUM))
    return loss_rate, acc_rate
```

6. 模型预测

训练完成后对未标记的测试集进行预测，加载测试集的方法与训练集类似 (详见 loadTestData_Roi() 方法)。实验要求将预测结果写入 txt 文件中，使用 torch.max() 方法返回 argmax 的值，将预测 label 按行写入文件：

模型预测 Python

```python
import torch
import os

def predict(model, test, showExample=False):
    # 将预测结果写入txt文件中
    with open("predict_labels_1120222198_张英祺.txt", 'w') as f:
        with torch.no_grad():
            out = model(test)
            _, pred = torch.max(out.data, 1)
        for res in pred.tolist():
            f.write(str(res)+'\n')
    print("Finished Predicting! processed images={}".format(len(pred)))

    if showExample:
        path = os.path.abspath(os.path.join(os.getcwd(), '.')) + "\\GTSRB\\Final_Test\\Images"
        images = os.listdir(path)
        for i in range(10):
            print("{} is labeled as {}".format(images[i], pred[i]))

    return None
```

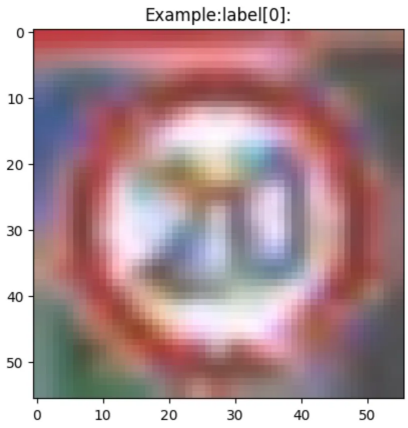
五、实验结果与分析

1. 运行展示

本实验的代码运行结果保存在 main.ipynb 中，使用 image_size=(56,56), batch_size=64, epoch_num=50, val_num=5 对模型进行训练，在验证集上的准确率达到了 0.993，效果较优。

```
# data, label = utils.loadTrainData(image_size=(28,28), showExample=True)
data, label = utils.loadTrainData_Roi(image_size=(56,56), showExample=True)
```

```
data=26640, label=26640, trainingset=180, average_size=[4023.9757339338676, 4100.411432515495]
```



数据加载与预处理

```
c:\Users\lilywhite\.conda\envs\pytorch\lib\site-packages\torch\nn\functional.py:1331:
  warnings.warn(warn_msg)
[2025-04-06 01:47:20] epoch 5 | loss:0.0131 | acc:0.9664 | time:367.25s
[2025-04-06 01:53:44] epoch 10 | loss:0.1573 | acc:0.9788 | time:379.75s
[2025-04-06 02:00:11] epoch 15 | loss:0.0052 | acc:0.9917 | time:381.91s
[2025-04-06 02:06:41] epoch 20 | loss:0.0017 | acc:0.9889 | time:385.52s
[2025-04-06 02:13:13] epoch 25 | loss:0.0023 | acc:0.9895 | time:386.66s
[2025-04-06 02:19:30] epoch 30 | loss:0.0003 | acc:0.9916 | time:372.98s
[2025-04-06 02:25:33] epoch 35 | loss:0.0009 | acc:0.9917 | time:358.89s
[2025-04-06 02:31:37] epoch 40 | loss:0.0032 | acc:0.9927 | time:359.93s
[2025-04-06 02:37:43] epoch 45 | loss:0.0001 | acc:0.9904 | time:361.76s
[2025-04-06 02:43:49] epoch 50 | loss:0.0000 | acc:0.9921 | time:361.55s
Finished Training! BATCH_SIZE=64, EPOCH=50, VAL_NUM=5
```

模型训练

```
utils.predict(model, test, showExample=True)
```

```
Finished Predicting! processed images=12630
00000.ppm is labeled as 12
00001.ppm is labeled as 12
00002.ppm is labeled as 13
00003.ppm is labeled as 12
00004.ppm is labeled as 2
00005.ppm is labeled as 38
00006.ppm is labeled as 25
00007.ppm is labeled as 12
00008.ppm is labeled as 13
00009.ppm is labeled as 2
```

预测分类

2. 模型效果

模型的几次运行测试结果可见输出的 log 文件。在 (batch_size=64, epoch_num=100) 下，未使用 Roi 增强数据集时的验证准确率可达 95%，尝试 Roi 增强数据集后准确率提升较快；此外尝试了一些数据增广方法，如翻转、噪声等，发现模型的准确率没有较为明显的提升，猜测对于此任务来说，已有的数据集效果已经足够好了。想要进一步优化预测效果，可以使用更加先进的模型结构，如 R-CNN 和 YOLO 等。

从性能的角度分析，在数据加载与预处理阶段，如果使用 PIL 库对图片进行 Roi 裁剪，虽然一定程度上可以提高识别精度，但是读取耗时提高了四十余倍。对于 28×28 的图片输入，训练 5 个 epoch 用时约 100 秒；如果将图片超分成 56×56，耗时还会增加四倍左右 (可以预料到的)。由于本实验在核显环境上进行测试，考虑到运行一次所花费的时间，在设置对照试验时，所有参数均选择性能较好的取值。

3. 超参数调整

本实验主要探讨了学习率 (learning_rate) 对训练效果的影响。关于 batch_size 和 epoch 的选择，可以参考实验输出的运行统计图 (包括 loss 曲线和 acc 曲线)。在上面提到的参数情况下，每隔半个数量级设置一个 lr 实验组，记录第 10 个 epoch 下的 loss 与 acc，用来评判该学习率下模型的收敛速度：

learning_rate 设置	loss (epoch=10)	acc (epoch=10)
lr=0.01	3.5604	0.0533
lr=0.005	3.5578	0.0552
lr=0.001	0.8174	0.8453
lr=0.0005	0.1124	0.9221
lr=0.0001	0.1223	0.9465
lr=0.00005	0.2397	0.9437
lr=0.00001	0.6103	0.8452

效果非常明显，在 lr=0.0001 的情况下，模型的训练效果最佳，可以此为基准设置学习率参数。

4. 数据增强与优化方法

本实验尝试了对数据集进行增广，采用的方法包括超分、翻转、噪声、随机亮度等。经过测试，这些方法对模型的提升作用很有限，并不能使准确率进一步提升，推测该数据集所包含的数据已经足够完善，不需要进行进一步增广。不过，实验尝试了在 Roi 裁剪后将图片 resize 到 28*28 和 56*56 两种大小，虽然准确率相差不大，但是在测试集上的预测结果出现了很多差异。

对此我的猜测是，由于该问题是一个 43 分类的任务，如果图片尺寸太小，可能会丢失一定的特征，而验证集和测试集的图片尺寸与训练集相同，因此无法从准确率上得知模型是否出现了过拟合 (也许我的猜测是错误的)。考虑到这一问题，在最后一次训练时我采用了 56*56 的输入。

5. 模型中存在的缺陷

因为时间有限，没有对模型中的所有超参数进行对照试验，因此模型所用的参数不一定最优。训练过程中的 loss 曲线依然存在震荡的问题 (不够光滑)，让人怀疑是不是模型存在着什么缺陷，准确率还能进一步提升吗？下一步可以尝试使用一些改进的 CNN 模型来处理，看看训练效果是否会有改变。

七、心得体会

1. 实现细节

1.1. Tensor 矩阵的转置和降维

在 seperateDataset() 中，我们将输入图片转存为 Tensor 矩阵，其维度为 [batch，in_height, in_width, in_channels] (BHWC)，而 torch.nn 接受的输入为 [batch, in_channels, in_height, in_width] (BCHW)，因此我们需要对 Tensor 矩阵进行一次转置，可以使用 permute(0, 3, 1, 2) 或 tf.transpose() 来实现。

CNN 网络中的各层之间也要匹配矩阵的大小。以图片尺寸 60×60 为例，卷积层的输入大小为 [64, 3, 60, 60]，经过两次卷积和池化后变为 [64, 64, 15, 15]。在送入线性层之前需要合并后三个维度，也就是添加一个 flatten 层，经此处理后矩阵变为 [64, 14400]，经过两个线性层被分类为 [64, 43]，在本实验中创建 CNN 实例时可以传入 image_size, img_channels, classes 作为参数来进行控制。

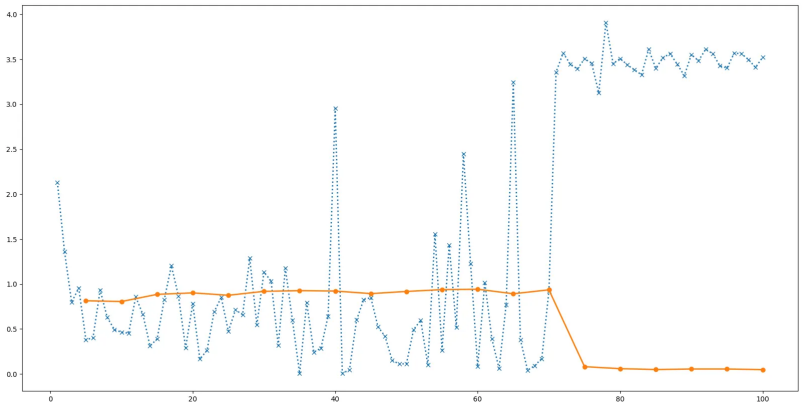
1.2. TensorFlow 与 Pytorch 的线性层区别

在设计 CNN 网络时，我注意到在线性层部分，有的网络会选择在两个 linear 层中间添加 relu 激活函数，有些网络则没有。经过查找资料和询问老师得知，TensorFlow 的线性层自带 relu 激活函数，而 pytorch 要在设置线性层后单独添加，不然的话，两个线性层之间缺少激活函数，实际上和单个线性层没有区别，在实际搭建模型的时候要注意这一点。本实验是一个简单的多分类任务，因此设置一或两个线性层没有太大差别，在网络中可以省略 relu 以减少运算量。

2. debug 经验

2.1. 解决梯度爆炸问题：打乱数据集的重要性

这份代码在初次运行时遇到了一个较为严重的问题：虽然随着训练进行，准确率能够保持在 0.9 左右，但是 loss 曲线存在着非常严重的震荡问题，甚至在超过一定的 epoch 后 (大于 50 次) 模型会变得发散。这很有可能是训练过程中出了什么问题，导致模型的训练过程变得"病态"：



100 个 epoch 下的训练情况，有较严重的问题

经检查后发现，该数据集是按照 label 进行分类的，在读入数据的时候没有设置 shuffle，导致导入的数据也是按照 label 进行排序的，这就导致 sklearn 划分验证集和后面选取 batch 进行训练时，有很大的可能划到同一 label 的数据。在 seperateDataset() 的开头按照同一 random seed 对训练集进行打乱，再次测试发现 loss 明显更加稳定了。

除了数据预处理之外，loss 震荡严重还要考虑以下几个问题：batch_size 过小、学习率 lr 过大、loss 函数/激活函数/优化器设计不合理、或者样本需要数据增强。除了调整模型超参数外，还可以通过一些处理来优化模型，如正则化约束，在本实验中 CNN 的卷积层后添加了正则化层来减少模型的过拟合。

2.2. 解决 loss 震荡问题：使用退火方法改进学习率设置

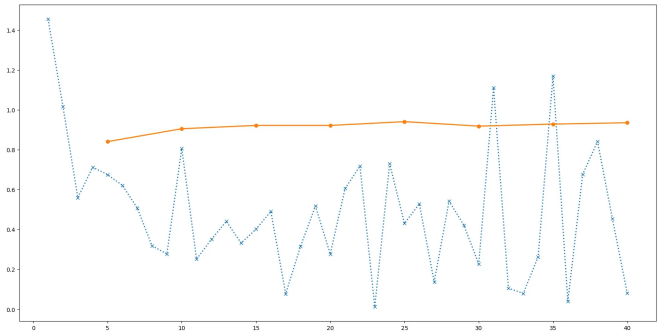
在学习率方面，我设置了根据 epoch 进行调整的学习率，以模拟退火操作。使用此方法确实可以从原理上压缩 loss 曲线，从而解决震荡问题。

根据 epoch 调整学习率 Python

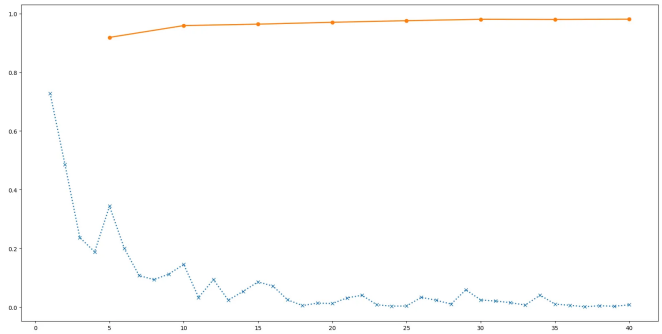
```python
def adjust_learning_rate(learning_rate, learning_rate_decay, optimizer, epoch):
    """Sets the learning rate to the initial LR multiplied by learning_rate_decay(set 0.98, usually) every epoch"""
    learning_rate = learning_rate * (learning_rate_decay ** epoch)

    for param_group in optimizer.param_groups:
        param_group['lr'] = learning_rate

    return learning_rate
```



使用固定学习率 (lr=0.001) 时的 loss 和 acc 曲线



使用改进后的学习率时的 loss 和 acc 曲线

效果十分显著！使用退火的解决思路可以有效限制 loss 的震荡，虽然这种操作有些治标不治本，但是解决了 loss 问题，模型的准确率也变得稳定了，对于训练效果不够理想的小数据集来说，这种方法能很好地实现模型收敛。

3. 思考与改进

相较于 MNIST 数据集的十几个分类，本实验所采用的 GTSRB 有 43 个分类，区分这些特征的难度会更大，相较于前者，是否需要更深层次的网络来提取更细节的特征？通过将原来的单个线性层改为两个逐级收缩的线性层，模型训练时的准确率突破了 99%，是否意味着更多的分类需要更多的线性层用来过渡？与自然语言相比，图像的特征较为浅层，使用 CNN 能很好地提取到，那么限制模型效果的重要因素在于分类这一步上吗？下一步可以考虑使用一些改进的激活函数作为对比，如 PReLU 或 ELU 等，看看模型效果是否能进一步提升。