Assignment5

题目要求通过 PyTorch 重现 LeNet-5，并将 MNIST 数据集按照合理的比例划分为训练集、验证集和测试集。训练 LeNet-5，训练结束后在测试集上进行测试，然后计算测试结果的准确性。另外，自己手写 20 个数字进行测试，输出测试的准确性。

首先我们需要得到 MNIST 数据集，MNIST 数据集，总共有七万个已经打好标签的样本，其中有 6 万个默认为训练集，1 万个默认为测试集合。这里先读取数据，因为我已经下载好了 MNIST 数据集，这里 download=False 就不每一次重复下载了，读取样本的代码如下：

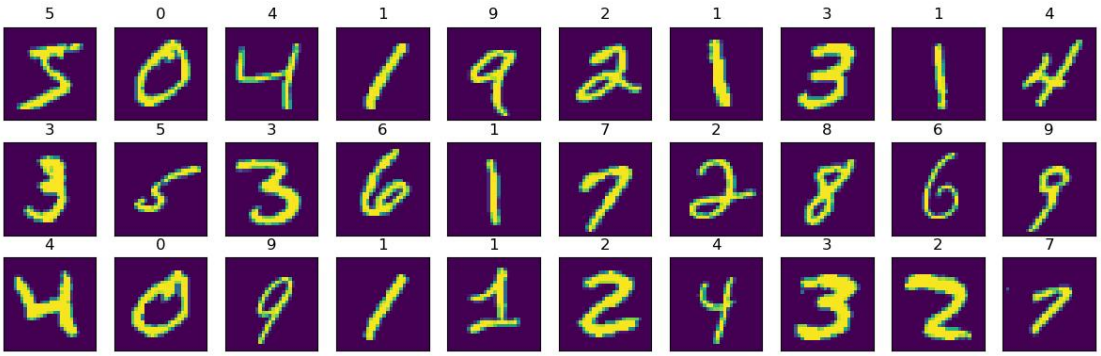
```
# 读取 MNIST 样本
def load_data_mnist(batch_size, resize=None):
    # 将 PIL 图像或 NumPy 数组转换为 torch 张量
    trans = [transforms.ToTensor()]
    # transforms.Compose()函数将这个操作序列打包成一个可用于数据转换的对象
    trans = transforms.Compose(trans)
    # 下载数据集
    mnist_dataset1 = torchvision.datasets.MNIST(
        root=r"D:\PycharmProjects\pythonProject_pytorch\image",
train=True, transform=trans, download=False)
    mnist_dataset2 = torchvision.datasets.MNIST(
        root=r"D:\PycharmProjects\pythonProject_pytorch\image",
train=False, transform=trans, download=False)
    # 这里已经下载到文件夹里了，选择 download=False 避免重复下载
    # 讲两个训练集合并，再打乱
    concat_dataset = data.ConcatDataset([mnist_dataset1,
mnist_dataset2])  # 合并
    mnist_train, mnist_val, mnist_test =
data.random_split(concat_dataset, [50000, 10000, 10000])  # 打乱拆分
    # 划分训练集和验证集，这里从训练集里分出 10000 个样本做验证集合，训练集、验证集、
测试集数量为 50000、10000、10000
    print(len(mnist_train), len(mnist_val), len(mnist_test))
    return (data.DataLoader(mnist_train, batch_size, shuffle=True,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_val, batch_size, shuffle=False,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_test, batch_size, shuffle=False,
                            num_workers=get_dataloader_workers()))
```

这里对 MNIST 数据做了预处理，进行了归一化且把图像转为张量形式，由于 MNIST 数据集分为了两部分，所以我对数据集进行了合并，并且打乱顺序（增强

系统的鲁棒性），之后将合并的总样本分为，5 万个训练集样本，1 万个验证集样本，1 万个测试集样本。并且使用 DataLoader 将这三个数据集变成可迭代对象进行输出。此外我们可以将数据集使用 Matplotlib 给画出来看一下是什么样子的，代码如下：

```
def get_fashion_mnist_labels(labels):
    text_labels = ['0', '1', '2', '3', '4',
                   '5', '6', '7', '8', '9']
    return [text_labels[int(i)] for i in labels]


# 绘制图像列表
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()
    for i, (ax, img) in enumerate(zip(axes, imgs)):
        if torch.is_tensor(img):
            # 图片张量
            ax.imshow(img.numpy())
        else:
            # PIL 图片
            ax.imshow(img)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        if titles:
            ax.set_title(titles[i])
    return axes


X, y = next(iter(DataLoader(mnist_train, batch_size=30)))
show_images(X.reshape(30, 28, 28), 3, 10, titles=mnist_labels(y));
d2l.plt.show()
```

可以观测到图片如下图：



之后我们需要定义 LeNet-5 网络，并且得到三个数据集的迭代样本，这里使用交叉熵损失函数，里面自带 softmax，所以在定义网络时没有在输出加上 softmax。

```
# 定义网络
net = nn.Sequential(nn.Conv2d(1, 6, kernel_size=5, padding=2),
nn.Sigmoid(),
                    nn.AvgPool2d(kernel_size=2, stride=2),
                    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
                    nn.AvgPool2d(kernel_size=2, stride=2),
                    nn.Flatten(),
                    nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
                    nn.Linear(120, 84), nn.Sigmoid(),
                    nn.Linear(84, 10))

batch_size = 256
train_iter, val_iter, test_iter =
load_data_mnist(batch_size=batch_size)
```

由于是处理图像，为了加快迭代速度，这里使用 GPU 来跑模型，下面先在 GPU 上计算数据的精度。

```
# 使用 GPU 计算模型在数据集上的精度
def evaluate_accuracy_gpu(net, data_iter, device=None):
    if isinstance(net, nn.Module):
        net.eval()  # 设置为评估模式
        if not device:
            device = next(iter(net.parameters())).device
    # 正确预测的数量，总预测的数量
    metric = d2l.Accumulator(2)
    with torch.no_grad():
        for X, y in data_iter:
            if isinstance(X, list):
                X = [x.to(device) for x in X]
            else:
                X = X.to(device)
            y = y.to(device)
            metric.add(d2l.accuracy(net(X), y), y.numel())
    return metric[0] / metric[1]
```

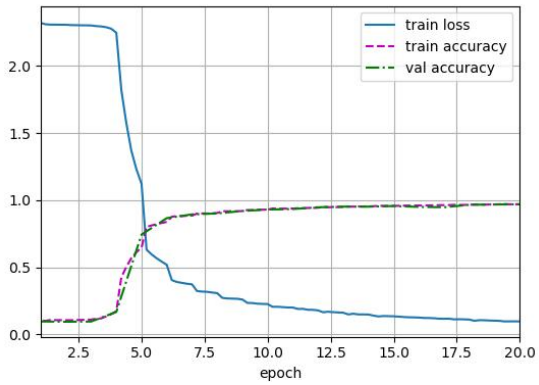
之后在 GPU 部署模型，使用随机梯度下降法，每一个 minibatch 为 256，损失函数使用交叉熵损失函数，在每一个 minibatch 时先清除梯度，进行前向传播和反向传播，再进行 w 和 b 的更新，再去重复下一个 minibatch 直到例遍所有样本，期间也会对验证集进行计算，来判断模型是否按照我们的期望进行收敛，方

便调节超参数。在最后把测试集放到 GPU 上进行前向传播，看测试集上的精度。

```
# 使用 GPU 训练模型
def train(net, train_iter, val_iter, num_epochs, lr, device):
    # 初始化权重
    def init_weights(m):
        if type(m) == nn.Linear or type(m) == nn.Conv2d:
            nn.init.xavier_uniform_(m.weight)
    net.apply(init_weights)
    print('training on', device)
    net.to(device)
    optimizer = torch.optim.SGD(net.parameters(), lr=lr)
    loss = nn.CrossEntropyLoss()
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs],
                            legend=['train loss', 'train accuracy', 'val
accuracy'])
    timer, num_batches = d2l.Timer(), len(train_iter)
    for epoch in range(num_epochs):
        # 训练损失之和，训练准确率之和，样本数
        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate(train_iter):
            timer.start()
            optimizer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            l.backward()
            optimizer.step()
            with torch.no_grad():
                metric.add(l * X.shape[0], d2l.accuracy(y_hat, y),
X.shape[0])
            timer.stop()
            train_l = metric[0] / metric[2]
            train_acc = metric[1] / metric[2]
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (train_l, train_acc, None))
        val_acc = evaluate_accuracy_gpu(net, val_iter)
        animator.add(epoch + 1, (None, None, val_acc))
    test_acc = evaluate_accuracy_gpu(net, test_iter)

    print(f'loss {train_l:.3f}, train acc {train_acc:.3f}, '
          f'val acc {val_acc:.3f}')
    print(f'test acc {test_acc:.3f}')
```

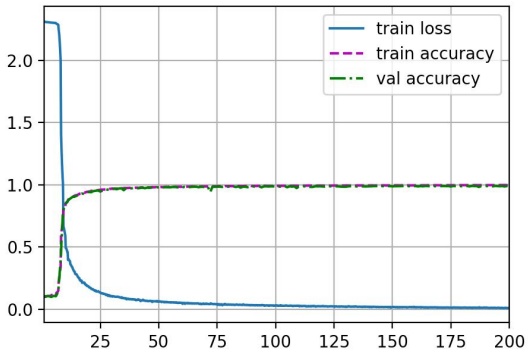
```
    print(f'{metric[2] * num_epochs / timer.sum():.1f} examples/sec '
          f'on {str(device)}')
```

通过调节不同的超参数，我们可以得到它的训练曲线，以及训练精度：



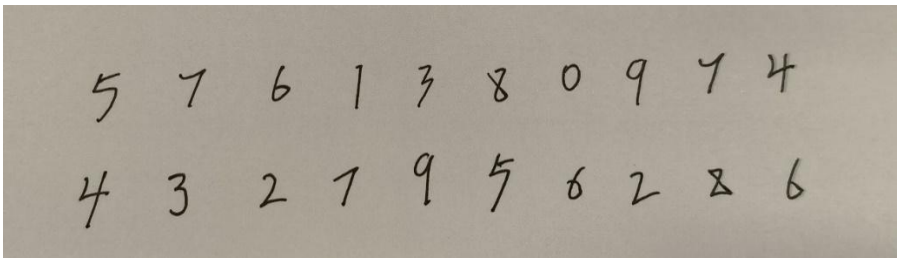

```
loss 0.074, train acc 0.977, val acc 0.973
65397.6 examples/sec on cuda:0
test acc 0.974
```

通过不断地训练我们可以达到一个较好的训练效果，以下是我迭代 20 次，学习率为 0.5 的一个效果，可以看到训练集的精度已经到了 99%的准确率了：

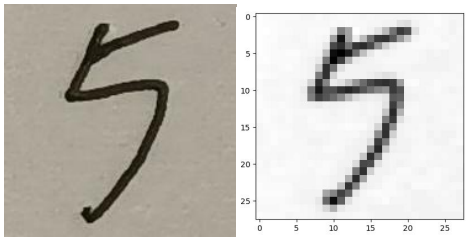



```
<Figure size 1920x894 with 1 Axes>
<Figure size 1920x894 with 1 Axes>
<Figure size 1920x894 with 1 Axes>
loss 0.010, train acc 0.998, val acc 0.989
test acc 0.987
69222.9 examples/sec on cuda:0
```

以上是使用 MNIST 数据集进行训练测试的结果，在调试完参数后，可以导入自己手写的图片来进行模型的测试，以下是我在 A4 纸上写的数字。

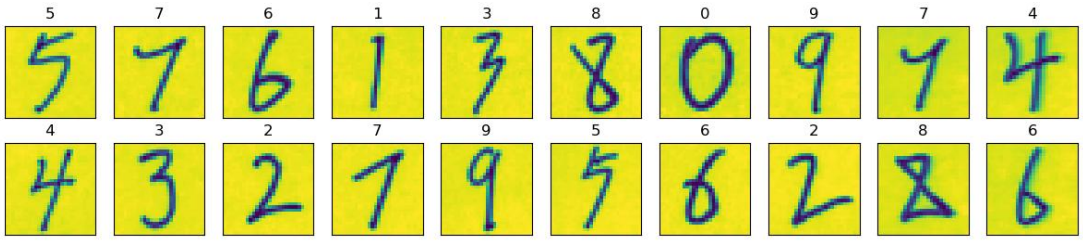


得到了数字后我们需要对图像进行一个切割，和灰度化。



之后将处理好的 20 个数字图片画出来，发现灰度是与 MNIST 的灰度是相反

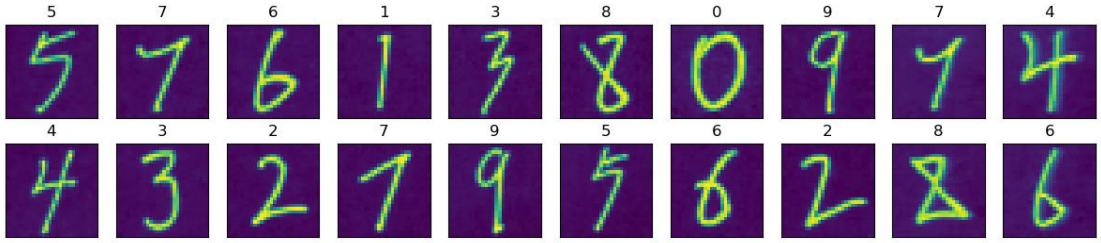
的，MNIST 的数字是白色的背景是黑色的，而手写的数字是黑色的 A4 纸是白色的，图像如下：



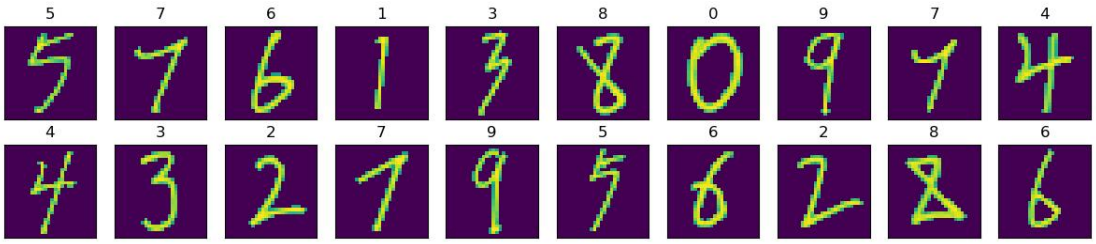
所以我们需要对图像的灰度进行一个反转，来达到与数据集统一的效果。

```
# 灰度反转
gray_img = 1 - gray_img
```

得到处理的图像如下：



并且仿照 MNIST 的做法对黑色部分的灰度降低为 0（变得更黑），处理图形状如下：



再将自己的数据集，做成一个 dataset，这里我直接重新定义了 dataset

```
# 自己手写字体识别
# 图片预处理
trans = transforms.Compose([
    # 转灰度图
    transforms.Grayscale(),
    # 转变图像大小
    transforms.Resize((28, 28)),
    # 把图像归一化，且变为张量
    transforms.ToTensor()
])


class My_image_Dataset(data.Dataset):
    def __init__(self):
```

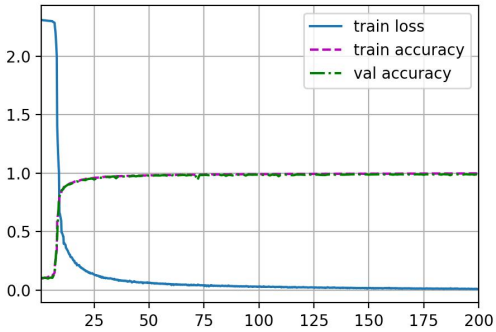
```
        super().__init__()
        # 设定一个空的 list 来存储每次读取到的图片的 tensor，为 cat 做准备
        Imagecat = []
        for i in range(20):
            # 读取图片
            img = Image.open(r"C:\Users\53564\Desktop\手写数字
\IMG_{}.jpg".format(str(i)))
            # 进行图像的预处理（归一化和灰度和变 tensor）
            gray_img = trans(img)
            # 灰度反转
            gray_img = 1 - gray_img
            # 将图像黑色更黑
            threshold = 0.5
            gray_img[gray_img < threshold] = 0.0
            # 增加维度，将 1*28*28 扩展为 1*1*28*28
            gray_img = gray_img.unsqueeze(0)
            # 把每一次处理好的图片 tensor 放进列表中
            Imagecat.append(gray_img)
        # 按照 batch_size 维度来合成一个 4D 的 tensor
        self.data = torch.cat(Imagecat, dim=0)
        self.label = torch.tensor([5, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2,
7, 9, 5, 6, 2, 8, 6])
        # 可以检验 x 维度是否正确
        # print(self.data.shape)

    def __getitem__(self, index):
        x = self.data[index]
        y = self.label[index]
        return x, y

    def __len__(self):
        return len(self.data)
```

之后就可以把模型设定为评估模式进行前向传播了，这里我们还使用之前定义好的在 GPU 上计算精度的方法，计算精度。

```
net.eval() # 设置为评估模式
MyDataset = My_image_Dataset()
My_image_Dataset_iter = data.DataLoader(MyDataset, batch_size=20,
shuffle=False)
My_image_Dataset_iter_acc = evaluate_accuracy_gpu(net,
My_image_Dataset_iter)
print(f'My_image_Dataset_iter_acc {My_image_Dataset_iter_acc:.3f}')
```

```
for i, (X, y) in enumerate(My_image_Dataset_iter):
    X, y = X.to(d2l.try_gpu()), y.to(d2l.try_gpu())
    y_hat = net(X)
    probs = nn.functional.softmax(y_hat, dim=1)
    y_hat_output = torch.max(probs, dim=1)
    print("预测输出为：  ", y_hat_output)
```

由于直接使用交叉熵损失函数，最后一层不用加 softmax，我们要想看自己的数据集得到的概率要在最后一层加上 softmax，再使用 torch.max()来得到最终网络预测出的结果是第几类。只训练 20 个周期得到的训练效果并不太好，通过增加训练次数，调节超参数，可以得到较好的训练结果，当训练次数到 200 次，batch_size 为 20，学习率为 0.5 时得到的 loss 为 0.01，训练集准确性为 0.998，验证集准确性为 0.989，测试集准确性为 0.987。

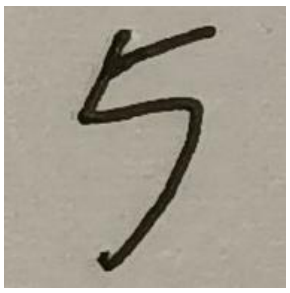



```
<Figure size 1920x894 with 1 Axes>
<Figure size 1920x894 with 1 Axes>
<Figure size 1920x894 with 1 Axes>
loss 0.010, train acc 0.998, val acc 0.989
test acc 0.987
69222.9 examples/sec on cuda:0
```

而通过自己写的训练集的准确性为 0.85，有三个认错了。

```
loss 0.016, train acc 0.996, val acc 0.988
test acc 0.987
67049.8 examples/sec on cuda:0
My_image_Dataset_iter_acc 0.850
预测输出为：  torch.return_types.max(
values=tensor([0.4318, 0.9999, 0.9949, 0.9978, 0.6222, 0.9997, 0.9996, 0.9994, 0.9937,
        0.5928, 0.9934, 0.9996, 0.9998, 0.9898, 0.9991, 0.6296, 0.8752, 0.9998,
        0.9411, 0.9970], device='cuda:0', grad_fn=<MaxBackward0>),
indices=tensor([7, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2, 7, 9, 7, 6, 2, 2, 6],
       device='cuda:0'))
```

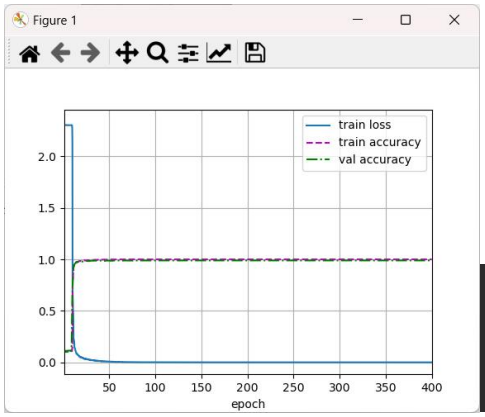
第一个错的是下面手写的 5，神经网络把它认作为了 7，应该是写的字下面部分像个 7，所以认错了。



第二个错也是 5，同样是认成了 7，也是因为我写的时候下面的部分像 7

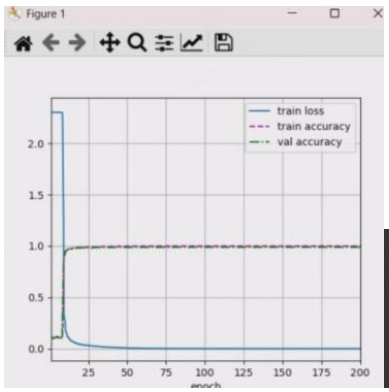
第三个错是我写的数字 8，如下图所示，神经网络认成了 2，因为我刻意把 8 里面的部分写成类似于 2 的样子，所以认错了。

强行增加训练次数还会出现过拟合的现象，准确度反而会下降，比如我训练了 400 次后反而精确度下降了：



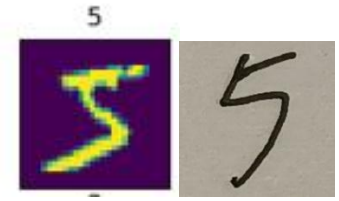

```
My_image_Dataset_iter_acc 0.750
预测输出为，  torch.return_types.max(
values=tensor([0.4986, 1.0000, 1.0000, 1.0000, 0.5041, 1.0000, 1.0000, 1.0000, 1.0000,
        0.9859, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 0.5404, 0.7249, 1.0000,
        0.9270, 1.0000], device='cuda:0', grad_fn=<MaxBackward0>),
indices=tensor([9, 7, 6, 1, 9, 8, 0, 9, 7, 4, 4, 3, 2, 7, 9, 9, 5, 2, 2, 6],
       device='cuda:0'))
```

在调整超参数继续训练，能发现准确度会提升，此时类似于 2 的 8 也能够预测出来了，但是 5 还是学习不出来，第一个 5 由于比较像 9 还被预测为 9。




```
My_image_Dataset_iter_acc 0.900
预测输出为，  torch.return_types.max(
values=tensor([0.7530, 1.0000, 0.9992, 1.0000, 0.7624, 1.0000, 1.0000, 0.9999, 0.9965,
        0.9629, 0.9461, 1.0000, 1.0000, 1.0000, 0.9985, 0.4480, 0.7068, 1.0000,
        0.9258, 0.9957], device='cuda:0', grad_fn=<MaxBackward0>),
indices=tensor([9, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2, 7, 9, 1, 6, 2, 8, 6],
       device='cuda:0'))
```

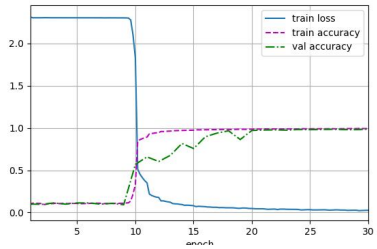
对比数据集的 5 和自己写的 5 可以发现问题，是因为数据集的 5 的写法和中国数字 5 的写法不太一样，需要额外去投喂我这种写法的数据集，专门去训练 5 的辨识能力，这样才能辨识出我这种写法的 5。



由于训练集中的 5 没有我这种写法，我又另外写了一堆 5 和 8 去投进神经网络的训练中，去进行特定数字的训练,代码如下，加入进训练神经网络的过程中。

```
for i, (X, y) in enumerate(My_image_Dataset_iter):
    X, y = X.to(d2l.try_gpu()), y.to(d2l.try_gpu())
    y_hat = net(X)
    l = loss(y_hat, y)
    l.backward()
    optimizer.step()
```

这种做法的效果非常明显，改进后的数据集后我只用了 30 个 epoch 就已经达到了非常高的准确率，而此时我手写的 20 个数据的准确率达到了 100%，每个手写的字都能识别出来。




```
loss 0.025, train acc 0.992, val acc 0.987
test acc 0.988
26893.8 examples/sec on cuda:0
My_image_Dataset_iter_acc 1.000
预测输出为：  torch.return_types.max(
values=tensor([0.9987, 0.9985, 0.9981, 0.9999, 0.9743, 1.0000, 0.9998, 0.9873, 0.9887,
        0.9794, 0.9974, 0.9972, 0.9990, 0.8943, 0.9579, 0.9420, 0.9227, 0.9987,
        0.9548, 0.9965], device='cuda:0', grad_fn=<MaxBackward0>),
indices=tensor([5, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2, 7, 9, 5, 6, 2, 8, 6],
       device='cuda:0'))
```

最后全部代码如下：

```
"""
作者: Xueqi Li
日期: 2023 年 04 月 06 日
assignment 5 实现 LeNet 神经网络
"""
import torch
import torchvision
from torch import nn
from torch.utils import data
```

```
from torchvision import transforms
from d2l import torch as d2l
from PIL import Image


# 使用 4 个进程来读取数据
def get_dataloader_workers():
    return 4


# 读取 MNIST 样本
def load_data_mnist(batch_size, resize=None):
    # 将 PIL 图像或 NumPy 数组转换为 torch 张量
    trans = [transforms.ToTensor()]
    # transforms.Compose()函数将这个操作序列打包成一个可用于数据转换的对象
    trans = transforms.Compose(trans)
    # 下载数据集
    mnist_dataset1 = torchvision.datasets.MNIST(
        root=r"D:\PycharmProjects\pythonProject_pytorch\image",
train=True, transform=trans, download=False)
    mnist_dataset2 = torchvision.datasets.MNIST(
        root=r"D:\PycharmProjects\pythonProject_pytorch\image",
train=False, transform=trans, download=False)
    # 这里已经下载到文件夹里了，选择 download=False 避免重复下载
    # 讲两个训练集合并，再打乱
    concat_dataset = data.ConcatDataset([mnist_dataset1,
mnist_dataset2])  # 合并
    mnist_train, mnist_val, mnist_test =
data.random_split(concat_dataset, [50000, 10000, 10000])  # 打乱拆分
    # 划分训练集和验证集，这里从训练集里分出 10000 个样本做验证集合，训练集、验证集、
测试集数量为 50000、10000、10000
    print(len(mnist_train), len(mnist_val), len(mnist_test))
    return (data.DataLoader(mnist_train, batch_size, shuffle=True,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_val, batch_size, shuffle=False,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_test, batch_size, shuffle=False,
                            num_workers=get_dataloader_workers()))


# 定义网络
net = nn.Sequential(nn.Conv2d(1, 6, kernel_size=5, padding=2),
nn.Sigmoid(),
                    nn.AvgPool2d(kernel_size=2, stride=2),
```

```
                nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
                nn.AvgPool2d(kernel_size=2, stride=2),
                nn.Flatten(),
                nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
                nn.Linear(120, 84), nn.Sigmoid(),
                nn.Linear(84, 10))

# batch_size = 256
batch_size = 50
train_iter, val_iter, test_iter = load_data_mnist(batch_size=batch_size)


# 使用 GPU 计算模型在数据集上的精度
def evaluate_accuracy_gpu(net, data_iter, device=None):
    if isinstance(net, nn.Module):
        net.eval()  # 设置为评估模式
        if not device:
            device = next(iter(net.parameters())).device
    # 正确预测的数量，总预测的数量
    metric = d2l.Accumulator(2)
    with torch.no_grad():
        for X, y in data_iter:
            if isinstance(X, list):
                X = [x.to(device) for x in X]
            else:
                X = X.to(device)
            y = y.to(device)
            metric.add(d2l.accuracy(net(X), y), y.numel())
    return metric[0] / metric[1]


# 使用 GPU 训练模型
def train(net, train_iter, val_iter, num_epochs, lr, device):
    # 初始化权重
    def init_weights(m):
        if type(m) == nn.Linear or type(m) == nn.Conv2d:
            nn.init.xavier_uniform_(m.weight)
    net.apply(init_weights)
    print('training on', device)
    net.to(device)
    optimizer = torch.optim.SGD(net.parameters(), lr=lr)
    loss = nn.CrossEntropyLoss()
    animator = d2l.Animator(xlabel='epoch', xlim=[1, num_epochs],
                            legend=['train loss', 'train accuracy', 'val
```

```
accuracy'])
    timer, num_batches = d2l.Timer(), len(train_iter)
    for epoch in range(num_epochs):
        # 训练损失之和，训练准确率之和，样本数
        metric = d2l.Accumulator(3)
        net.train()
        for i, (X, y) in enumerate(train_iter):
            timer.start()
            optimizer.zero_grad()
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            l.backward()
            optimizer.step()
            with torch.no_grad():
                metric.add(l * X.shape[0], d2l.accuracy(y_hat, y),
X.shape[0])
            timer.stop()
            train_l = metric[0] / metric[2]
            train_acc = metric[1] / metric[2]
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches - 1:
                animator.add(epoch + (i + 1) / num_batches,
                             (train_l, train_acc, None))
        val_acc = evaluate_accuracy_gpu(net, val_iter)
        animator.add(epoch + 1, (None, None, val_acc))
    test_acc = evaluate_accuracy_gpu(net, test_iter)

    print(f'loss {train_l:.3f}, train acc {train_acc:.3f}, '
          f'val acc {val_acc:.3f}')
    print(f'test acc {test_acc:.3f}')
    print(f'{metric[2] * num_epochs / timer.sum():.1f} examples/sec '
          f'on {str(device)}')


lr, num_epochs = 0.3, 200
train(net, train_iter, val_iter, num_epochs, lr, d2l.try_gpu())
d2l.plt.show()

# 自己手写字体识别
# 图片预处理
trans = transforms.Compose([
    # 转灰度图
    transforms.Grayscale(),
    # 转变图像大小
```

```
    transforms.Resize((28, 28)),
    # 把图像归一化，且变为张量
    transforms.ToTensor()
])


class My_image_Dataset(data.Dataset):
    def __init__(self):
        super().__init__()
        # 设定一个空的 list 来存储每次读取到的图片的 tensor，为 cat 做准备
        Imagecat = []
        for i in range(20):
            # 读取图片
            img = Image.open(r"C:\Users\53564\Desktop\手写数字
\IMG_{}.jpg".format(str(i)))
            # 进行图像的预处理（归一化和灰度和变 tensor）
            gray_img = trans(img)
            # 灰度反转
            gray_img = 1 - gray_img
            # 将图像黑色更黑
            threshold = 0.5
            gray_img[gray_img < threshold] = 0.0
            # 增加维度，将 1*28*28 扩展为 1*1*28*28
            gray_img = gray_img.unsqueeze(0)
            # 把每一次处理好的图片 tensor 放进列表中
            Imagecat.append(gray_img)
        # 按照 batch_size 维度来合成一个 4D 的 tensor
        self.data = torch.cat(Imagecat, dim=0)
        self.label = torch.tensor([5, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2,
7, 9, 5, 6, 2, 8, 6])
        # 可以检验 x 维度是否正确
        # print(self.data.shape)

    def __getitem__(self, index):
        x = self.data[index]
        y = self.label[index]
        return x, y

    def __len__(self):
        return len(self.data)


net.eval() # 设置为评估模式
MyDataset = My_image_Dataset()
```

```
My_image_Dataset_iter = data.DataLoader(MyDataset, batch_size=20,
shuffle=False)
My_image_Dataset_iter_acc = evaluate_accuracy_gpu(net,
My_image_Dataset_iter)
print(f'My_image_Dataset_iter_acc {My_image_Dataset_iter_acc:.3f}')


for i, (X, y) in enumerate(My_image_Dataset_iter):
    X, y = X.to(d2l.try_gpu()), y.to(d2l.try_gpu())
    y_hat = net(X)
    probs = nn.functional.softmax(y_hat, dim=1)
    y_hat_output = torch.max(probs, dim=1)
    print("预测输出为： ", y_hat_output)
```

画图代码如下：

```
# 将自己的图片转变的代码
import torch
import torchvision
from torch import nn
from torch.utils import data
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from torchvision import transforms
from d2l import torch as d2l
from PIL import Image

# 图片预处理
trans = transforms.Compose([
    # 转灰度图
    transforms.Grayscale(),
    # 转变图像大小
    transforms.Resize((28, 28)),
    # 把图像归一化，且变为张量
    transforms.ToTensor()
])


# 设定一个空的 list 来存储每次读取到的图片的 tensor，为 cat 做准备
Imagecat = []
for i in range(20):
    # 读取图片
    img = Image.open(r"C:\Users\53564\Desktop\手写数字
\IMG_{}.jpg".format(str(i)))
    # 进行图像的预处理（归一化和灰度和变 tensor）
    gray_img = trans(img)
```

```
    gray_img = 1 - gray_img
    # 将图像黑色部分变成全黑，白色部分（字体部分不变）
    threshold = 0.5
    gray_img[gray_img < threshold] = 0.0
    # 增加维度，将 1*28*28 扩展为 1*1*28*28
    gray_img = gray_img.unsqueeze(0)
    # 把每一次处理好的图片 tensor 放进列表中
    Imagecat.append(gray_img)
    # 按照 batch_size 维度来合成一个 4D 的 tensor
data = torch.cat(Imagecat, dim=0)
label = torch.tensor([5, 7, 6, 1, 3, 8, 0, 9, 7, 4, 4, 3, 2, 7, 9, 5, 6,
2, 8, 6])
dataset = TensorDataset(data, label)
print(len(dataset))
print(dataset[19][1])


def get_fashion_mnist_labels(labels):
    text_labels = ['0', '1', '2', '3', '4',
                   '5', '6', '7', '8', '9']
    return [text_labels[int(i)] for i in labels]


# 绘制图像列表
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = d2l.plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()
    for i, (ax, img) in enumerate(zip(axes, imgs)):
        if torch.is_tensor(img):
            # 图片张量
            ax.imshow(img.numpy())
        else:
            # PIL 图片
            ax.imshow(img)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        if titles:
            ax.set_title(titles[i])
    return axes


X, y = next(iter(DataLoader(dataset, batch_size=20)))
show_images(X.reshape(20, 28, 28), 2, 10,
```

```
titles=get_fashion_mnist_labels(y));
d2l.plt.show()
```