

1. 实验实现的功能简述及运行说明

1.1 实验实现功能

利用CNN进行手写数字识别

1.2 运行说明

- 程序运行：

在linux环境下在命令行中输入: `python3 path to/train.py` 运行程序

(例如/mnt/f/study/CV/hw5/train.py)

- 运行的结果：

输入的数据存放在 `train` 文件夹中

测试的数据存放在 `test` 文件夹中

测试的前一部分数字图片存放在 `test_images.png` 中

测试结果存放在 `predictions.txt` 中

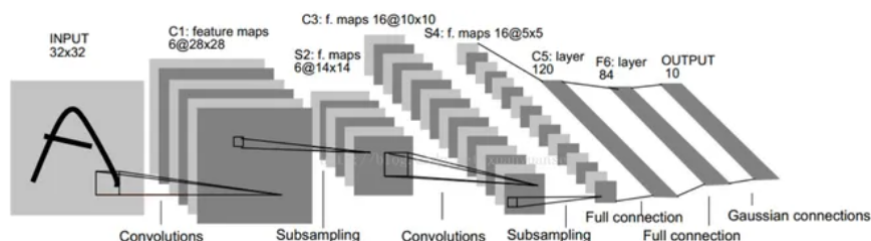
2. 作业的开发与运行环境

- ubuntu 22.04.3 LTS linux子系统
- 使用python + torch库

3. 主要用到的函数与算法

3.1 LeNet-5 模型

LeNet-5是一个深度卷积神经网络，可以解决手写数字识别问题，主要由卷积层、池化层和全连接层三个部分构成，如下图所示：



主要卷积操作如下：

- 卷积层C1包括6个卷积核，每个卷积核的大小为 5×5 ，步长为1，填充为0。每个卷积核会产生一个大小为 28×28 的特征图
- 采样层每个窗口的大小为 2×2 ，步长为2。每个池化操作从4个相邻的特征图中选择最大值，产生 14×14 的特征图，减少特征图的大小
- 卷积层C3包括16个卷积核，每个卷积核的大小为 5×5 ，步长为1，填充为0。每个卷积核会产生一个大小为 10×10 的特征图

3.2 BP算法

通过最小化误差函数来优化网络的权重和偏置。网络的权重和偏置是通过随机初始化得到的，然后，网络通过反向传播算法不断地调整权重和偏置，使得误差函数最小化

4. 实验步骤及代码具体实现

4.1 编写训练函数代码

- 将用到的 torch 等库导入进来：

```
1 from model import Model
2 import numpy as np
3 import os
4 import torch
5 import torchvision
6 from torchvision.datasets import mnist
7 from torch.nn import CrossEntropyLoss
8 from torch.optim import SGD
9 from torch.utils.data import DataLoader
10 from torchvision.transforms import ToTensor
11 import matplotlib
12 matplotlib.use('Agg') # 使用 'Agg' 后端
13 import matplotlib.pyplot as plt
```

- 定义模型：
 - 定义卷积层、最大池化层和全连接层。定义每个层的输入和输出通道数等参数。逐渐减小输入图像的空间尺寸，增加通道的数量
 - 定义 forward 方法进行前向传播。输入经过卷积层、最大池化操作、再次进行卷积池化操作。最后将特征图转换为一维形状，通过多个全连接层进行分类输出。

```
1 class Model(Module):
2     def __init__(self):
3         super(Model, self).__init__()
4         self.conv1 = nn.Conv2d(1, 6, 5)
5         self.relu1 = nn.ReLU()
6         self.pool1 = nn.MaxPool2d(2)
7         self.conv2 = nn.Conv2d(6, 16, 5)
8         self.relu2 = nn.ReLU()
9         self.pool2 = nn.MaxPool2d(2)
10        self.fc1 = nn.Linear(256, 120)
11        self.relu3 = nn.ReLU()
12        self.fc2 = nn.Linear(120, 84)
13        self.relu4 = nn.ReLU()
```

```

14         self.fc3 = nn.Linear(84, 10)
15         self.relu5 = nn.ReLU()
16
17     def forward(self, x):
18         y = self.conv1(x)
19         y = self.relu1(y)
20         y = self.pool1(y)
21         y = self.conv2(y)
22         y = self.relu2(y)
23         y = self.pool2(y)
24         y = y.view(y.shape[0], -1)
25         y = self.fc1(y)
26         y = self.relu3(y)
27         y = self.fc2(y)
28         y = self.relu4(y)
29         y = self.fc3(y)
30         y = self.relu5(y)
31         return y

```

- 初始化参数并加载数据:

- 设置训练和测试的Batch size。
- 使用 `torchvision.datasets.mnist.MNIST` 加载MNIST数据集; 使用 `torch.utils.data.DataLoader` 创建训练和测试数据加载器
- 定义优化器和损失函数。使用随机梯度下降作为优化器, 交叉熵损失作为损失函数。
- 设置训练总周期数

```

1     # Batch size
2     batch_size = 256
3     # Load MNIST dataset
4     train_dataset = mnist.MNIST(root='./train', train=True,
transform=ToTensor())
5     test_dataset = mnist.MNIST(root='./test', train=False,
transform=ToTensor())
6     train_loader = DataLoader(train_dataset, batch_size=batch_size)
7     test_loader = DataLoader(test_dataset, batch_size=batch_size)
8     model = Model().to(device)
9     # Define the optimizer and loss function
10    sgd = SGD(model.parameters(), lr=1e-1)
11    loss_fn = CrossEntropyLoss()
12    all_epoch = 100
13    prev_acc = 0
14    all_predictions = []

```

- 训练模型:

对每个训练批次进行前向传播、计算损失和反向传播更新参数

```

1  for current_epoch in range(all_epoch):
2      # Training
3      model.train()
4      for idx, (train_x, train_label) in enumerate(train_loader):
5          train_x = train_x.to(device)
6          train_label = train_label.to(device)
7          sgd.zero_grad()
8          predict_y = model(train_x.float())
9          loss = loss_fn(predict_y, train_label.long())
10         loss.backward()
11         sgd.step()

```

- 测试:

对测试集进行预测，计算正确预测的数量和计算准确率并进行输出

```

1  # Evaluation
2      all_correct_num = 0
3      all_sample_num = 0
4      model.eval()
5
6      for idx, (test_x, test_label) in enumerate(test_loader):
7          test_x = test_x.to(device)
8          test_label = test_label.to(device)
9          predict_y = model(test_x.float()).detach()
10         predict_y = torch.argmax(predict_y, dim=-1)
11         current_correct_num = predict_y == test_label
12         all_correct_num +=
np.sum(current_correct_num.to('cpu').numpy(), axis=-1)
13         all_sample_num += current_correct_num.shape[0]
14         all_predictions.extend(predict_y.to('cpu').numpy())
15
16         acc = all_correct_num / all_sample_num
17         acc_all.append(acc)
18         print('accuracy: {:.3f}'.format(acc), flush=True)
19         # Save the model
20         if not os.path.isdir("models"):
21             os.mkdir("models")
22         torch.save(model, 'models/mnist_{:.3f}.pkl'.format(acc))
23         # Early stopping condition
24         if np.abs(acc - prev_acc) < 1e-4:
25             break
26         prev_acc = acc

```

- 结果保存:

保存测试集中的第一批图像。将所有预测结果保存到 predictions.txt 中

```

1  # Save the model
2      if not os.path.isdir("models"):
3          os.mkdir("models")
4      torch.save(model, 'models/mnist_{:.3f}.pkl'.format(acc))
5      # Early stopping condition
6      if np.abs(acc - prev_acc) < 1e-4:
7          break
8      prev_acc = acc
9
10     print("Model finished training")
11
12     # Display first few images and their labels
13     def imshow(img, file_name):
14         npimg = img.numpy()
15         plt.imshow(np.transpose(npimg, (1, 2, 0)))
16         plt.savefig(file_name)
17
18     # Get a batch of test images
19     dataiter = iter(test_loader)
20     images, labels = next(dataiter)
21
22     # Save images
23     imshow(torchvision.utils.make_grid(images), 'test_images.png')
24     print('Images saved as test_images.png')
25     ct = 0
26     # Save all predictions to a .txt file
27     with open('predictions.txt', 'w') as f:
28         for prediction in all_predictions:
29             f.write('%5s ' % prediction)
30             ct = ct + 1
31             if(ct % 8 == 0):
32                 f.write('\n')
33
34     print('All predictions saved as predictions.txt')
35

```

4.2 结果测试

- 输入 `python3 train.py` 运行训练程序得到结果:

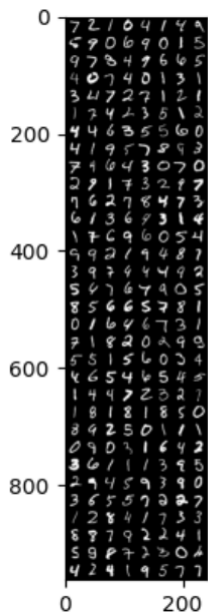
```

fengnian@MICROSO-VU1H310:/mnt/f/study/CV/hw5$ python3 train.py
accuracy: 0.873
accuracy: 0.884
accuracy: 0.941
accuracy: 0.959
accuracy: 0.968
accuracy: 0.974
accuracy: 0.974
accuracy: 0.978
accuracy: 0.980
accuracy: 0.981
accuracy: 0.982
accuracy: 0.983
accuracy: 0.983
Model finished training
Images saved as test_images.png
All predictions saved as predictions.txt

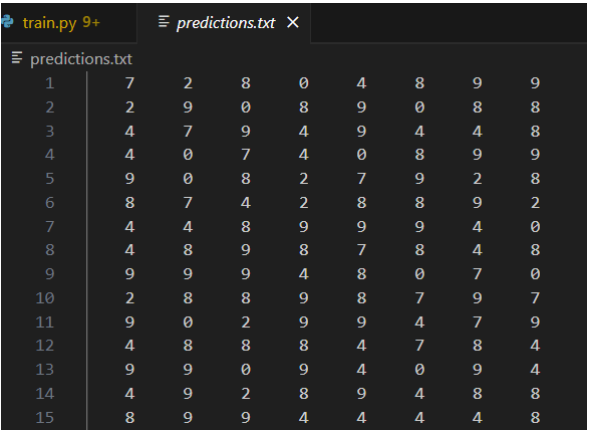
```

5. 实验结果与分析

- 输入的第一批图片展示：



- 对应的结果



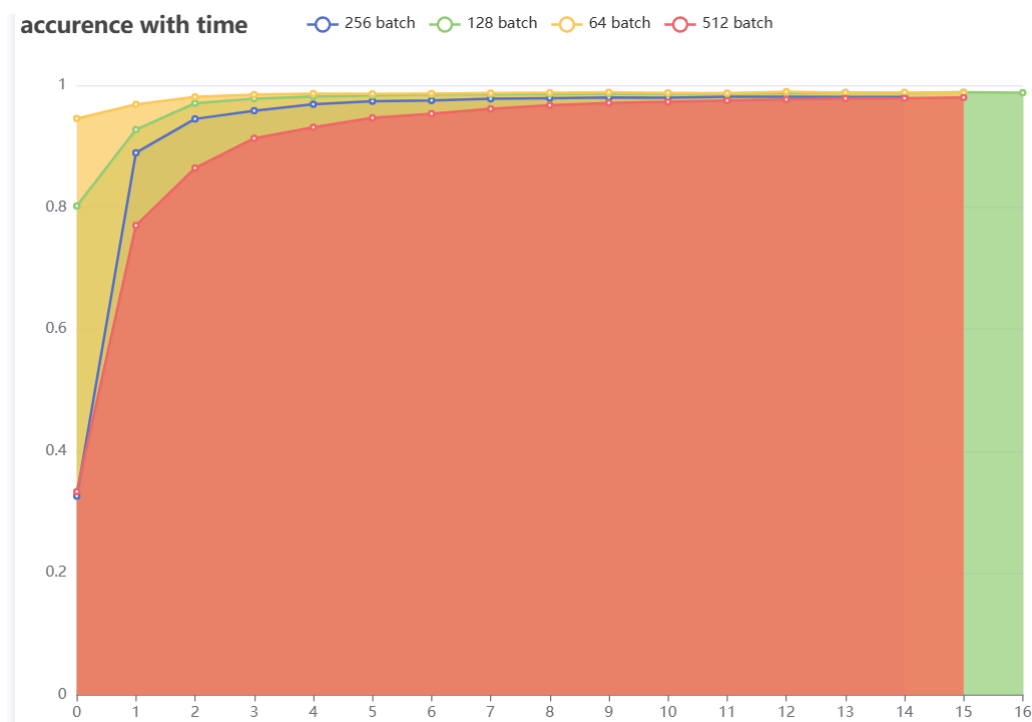
train.py 9+ predictions.txt X

predictions.txt

1	7	2	8	0	4	8	9	9
2	2	9	0	8	9	0	8	8
3	4	7	9	4	9	4	4	8
4	4	0	7	4	0	8	9	9
5	9	0	8	2	7	9	2	8
6	8	7	4	2	8	8	9	2
7	4	4	8	9	9	9	4	0
8	4	8	9	8	7	8	4	8
9	9	9	9	4	8	0	7	0
10	2	8	8	9	8	7	9	7
11	9	0	2	9	9	4	7	9
12	4	8	8	8	4	7	8	4
13	9	9	0	9	4	0	9	4
14	4	9	2	8	9	4	8	8
15	8	9	9	4	4	4	4	8

- 预测准确性结果：准确率基本维持在很高的水平，但每次训练都有偶然性，可能就出现准确率下降的情况

我调整了batch size，并对不同batch size下准确率的情况进行了记录，如下图：



其中，横坐标表示batch size，纵坐标表示准确率变化，通过坐标图可以看出，batch size越小，准确率会有一点提升，但是会感觉到时间明显上升

6. 心得体会

在本次实验中，我主要熟悉了LeNet-5等神经网络模型，了解了各层的作用，在测试的过程中也逐渐体会到各个参数的影响和作用，对于深度学习有了更深的认识

7. 参考文献

- [1] 卷积神经网络经典回顾之LeNet-5: <https://zhuanlan.zhihu.com/p/616996325>
- [2] 神经网络，BP算法的理解与推导: <https://zhuanlan.zhihu.com/p/45190898>