计算机视觉与应用实践实验报告（二）

目录

一、实验目的

- 试用 tensorflow playgroud，并写一篇使用报告。
- 实现 LeNet-5 在 MNIST 数据集上的训练和测试，并进行分析，完成实验报告，提交代码。

二、实验原理

2.1 MNIST 数据集

如下图所示，MNIST 是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所（National Institute of Standards and Technology (NIST)）发起整理，一共统计了来自 250 个不同的人手写数字图片，其中 50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。

1998 年，Yan LeCun 等人发表了论文《Gradient-Based Learning Applied to Document Recognition》，首次提出了 LeNet-5 网络，利用上述数据集实现了手写字体的识别。

官网上提供了数据集的下载，主要包括四个文件：

文件下载	文件用途
train-images-idx3-ubyte.gz	训练集图像
train-labels-idx1-ubyte.gz	训练集标签
t10k-images-idx3-ubyte.gz	测试集图像
t10k-labels-idx1-ubyte.gz	测试集标签

在上述文件中，训练集一共包含了 60,000 张图像和标签，而测试集一共包含了 10,000 张图像和标签。测试集中前 5000 个来自最初 NIST 项目的训练集.，后 5000 个来自最初 NIST 项目的测试集。前 5000 个比后 5000 个要规整，这是因为前 5000 个数据来自于美国人口普查局的员工，而后 5000 个来自于大学生。

该数据集自 1998 年起，被广泛地应用于机器学习和深度学习领域，用来测试算法的效果，例如线性分类器（Linear Classifiers）、K-近邻算法（K-Nearest Neighbors）、支持向量机（SVMs）、神经网络（Neural Nets）、卷积神经网络（Convolutional nets）等等。

2.2 LeNet-5 模型

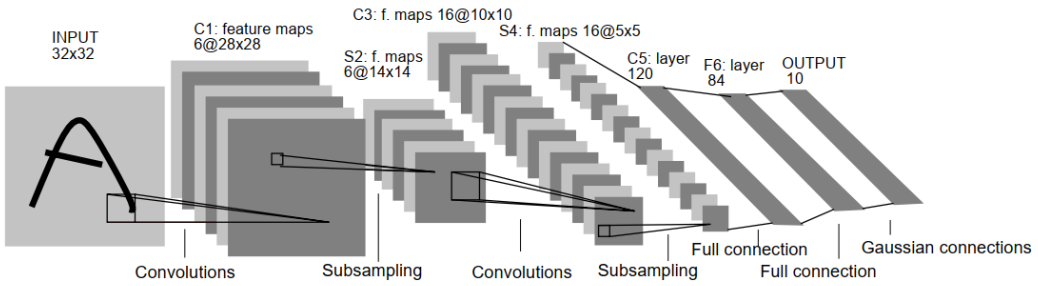


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5 网络模型作为卷积神经网络中的开创性工作，提取了三大思想：

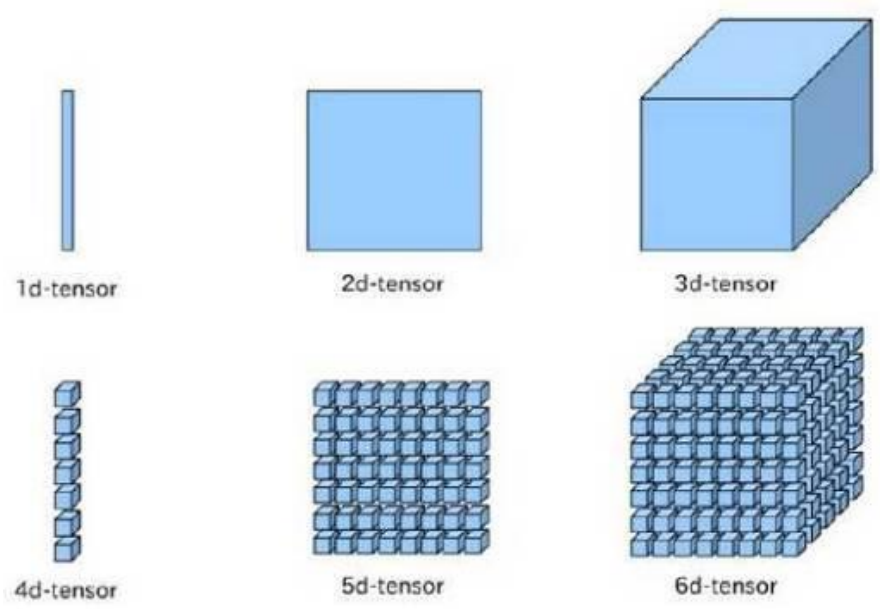
- 局部感知
- 权值共享
- 下采样

因为图像特征分布在图像的像素上，利用卷积操作可以在多个位置提取相类似的特征，于是有了局部感知。另外由于当年并没有计算能力强悍的 GPU 来辅助训练神经网络，因此通过下采样层有效地加快训练和提取更高维特征，能够节省参数和计算，这与当年的技术相比是一个关键的优势。另外原论文中提到，全卷积不应该被放在第一层，图像特征有着高度的空间相关性，因此权值共享可以充分利用图像上的空间相关性。

LeNet-5 共有 7 层，不包含输入层，每层都包含可训练参数；网络的结构为：输入 → 卷积 → 池化 → 卷积 → 池化 → 全连接 → 全连接 → 全连接 → 输出。每个层有多个 Feature Map，每个 Feature Map 通过一种卷积滤波器提取输入的一种特征，然后每个 Feature Map 有多个神经元。

三、tensorflow 使用报告

TensorFlow 是目前深度学习的一门框架，也是现在很热门、很多人使用的一门框架，他是编写深度学习算法时会用到的一种框架，它封装了很多类和函数，省去了我们要从最基层开始编写的时间，无论遇到什么问题，TensorFlow 都可以在一定程度上提供 API 的支持。那为什么叫 TensorFlow 呢，可以拆开来理解，Tensor 就是张量的意思，Flow 就是流动，那么合起来就是张量在流动？这样可能还是很抽象，我们可以把张量理解为多维的数组，如下图所示：



在 TensorFlow 中，所有的数据都可以借助张量的形式来表示。而流动就是张量数据沿着边在不同的节点间流动并发生转化

TensorFlow 能够帮助我们直接解决各种机器学习任务。目标就是在一般情况下，总的来说 TensorFlow 就是为了快而设计的，所以它针对我们实际使用的硬件和平台做了优化。其中在机器学习框架方面，TensorFlow 的真正独特之处在于，能够在 5 行或者 10 行代码中构建模型。然后应用这个模型，进行扩展做出产品。

一开始，python 是 tensorflow 的唯一选择，但是现在有了来自很多其他语言的支持。因为 TensorFlow 是开源的，长期以来在社区的支持下，越来越多的语言开始支持 TensorFlow。

Tensoflow 中我最喜欢的部分其实是 TensorBoard 这个可视化工具，它可以把模型运行过程进行可视化，并且尝试对模型预测的结果进行调试。

四、关键程序代码

```
class MNISTDataset(torch.utils.data.Dataset):

    def __init__(self, root, train=True, transform=None):

        self.file_pre = 'train' if train == True else 't10k'
```

```
        self.transform = transform
        self.label_path = os.path.join(root, '%s-labels-idx1-ubyte.gz' % self.file_pre)
        self.image_path = os.path.join(root, '%s-images-idx3-ubyte.gz' % self.file_pre)
        self.images, self.labels = self.__read_data__(self.image_path, self.label_path)

    def __read_data__(self, image_path, label_path):
        # Read dataset.
        with gzip.open(label_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8)
        with gzip.open(image_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8, offset=16).reshape(len(labels), 28, 28)
        return images, labels

    def __getitem__(self, index):
        image, label = self.images[index], int(self.labels[index])
        if self.transform is not None:
            image = self.transform(np.array(image))
        return image, label

    def __len__(self):
        return len(self.labels)
#加载数据
train_dataset = MNISTDataset('C:/Users/80712/Desktop/shiyan2/data/MNIST/',
transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize((0.1037,), (0.3081,))]))
test_dataset = MNISTDataset('C:/Users/80712/Desktop/shiyan2/data/MNIST/', train=False,
transform=torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
torchvision.transforms.Normalize((0.1037,), (0.3081,))]))
# Data Loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size,
shuffle=False)
```

```
def fc_in(image, Conv, Pool):
    for i, j in zip(Conv, Pool):
        hk = (image[0] - i[0] + 2 * i[2]) / i[1] + 1
        wk = (image[1] - i[0] + 2 * i[2]) / i[1] + 1
        hp = (hk - j[0] + 2 * j[2]) / j[1] + 1
        wp = (wk - j[0] + 2 * j[2]) / j[1] + 1
        image = (hp, wp)
    return (int(image[0]), int(image[1]))
fc_in((28, 28), ((5, 1, 0), (5, 1, 0)), ((2, 2, 0), (2, 2, 0)))


# LeNet-5 模型
class LeNet5(torch.nn.Module):
    def __init__(self, num_classes):
        super(LeNet5, self).__init__()
        self.layer1 = torch.nn.Sequential(torch.nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
                                          torch.nn.BatchNorm2d(6),
                                          torch.nn.ReLU(),
                                          torch.nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = torch.nn.Sequential(torch.nn.Conv2d(6, 16, kernel_size=5, stride=1,
padding=0),
                                          torch.nn.BatchNorm2d(16),
                                          torch.nn.ReLU(),
                                          torch.nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc1 = torch.nn.Sequential(torch.nn.Linear(4 * 4 * 16, 120),
                                       torch.nn.ReLU())
        self.fc2 = torch.nn.Sequential(torch.nn.Linear(120, 84),
                                       torch.nn.ReLU())
        self.fc3 = torch.nn.Linear(84, num_classes)


    def forward(self, x):
```

```
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        out = self.fc3(out)
        return out

# Make model
model = LeNet5(num_classes).to(device)
# Loss and optimizer
criterion = torch.nn.CrossEntropyLoss()
# optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# 训练模型
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optim
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss {:.4f}'.format(epoch+1, num_epochs, i+1, total_step,
loss.item()))



# 模型测试.
model.eval()
with torch.no_grad():
    total = 0
    correct = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    print ('Test Accuracy of the model on the 10000 test images: {} %'.format(100 * correct /
total))


# Save the model.
torch.save(model.state_dict(), 'LeNet5.ckpt')
```

五、实验结果

```
Epoch [9/10], Step [300/600], Loss 0.0049
Epoch [9/10], Step [400/600], Loss 0.0031
Epoch [9/10], Step [500/600], Loss 0.0890
Epoch [9/10], Step [600/600], Loss 0.0025
Epoch [10/10], Step [100/600], Loss 0.0020
Epoch [10/10], Step [200/600], Loss 0.0026
Epoch [10/10], Step [300/600], Loss 0.0025
Epoch [10/10], Step [400/600], Loss 0.0223
Epoch [10/10], Step [500/600], Loss 0.0234
Epoch [10/10], Step [600/600], Loss 0.0756
Test Accuracy of the model on the 10000 test images: 99.07 %
```

六、实验分析与总结

本次所使用的 LeNet-5 模型的是 CNN 中最基础的模型，其构建过程是所有其他网络的基本范式。通过这次搭建，我进一步熟悉了 pytorch 的各种用法；了解网络的搭建方法，弄懂了其参数和输入输出关系，明白了其中卷积池化后与全连接的之间维度上的匹配问题。

需要改进的地方：

1. 模型评估改进，还可以用 tensorboard 生成具体的测试集和训练集损失函数迭代曲线，以及准确率的迭代曲线。