卷积神经网络实战

PYTORCH 版

Dezeming Family

Dezeming

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录

前言及简介

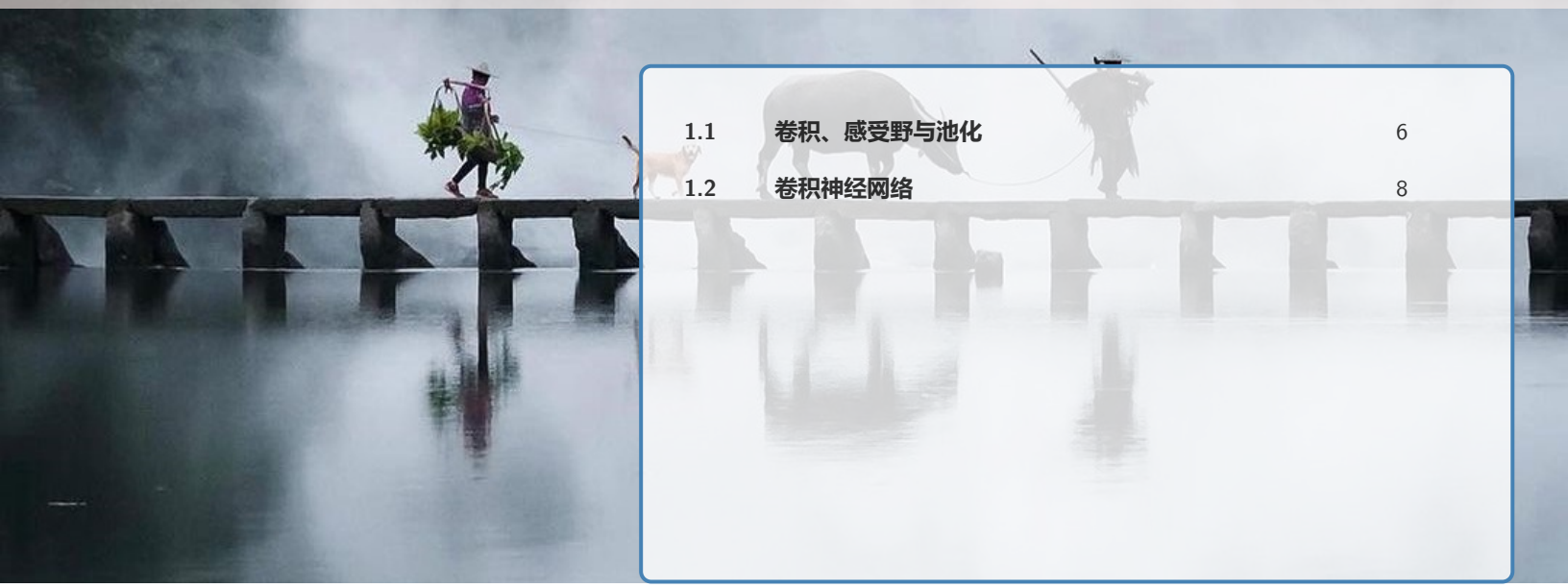
DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列书，可以从我们的网站 *[https://dezeming.top/]* 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

0.1 本书前言

接着上一本全连接神经网络的书，我们开始写本书。本书的意义在于开始着手一种特定的网络模型——卷积神经网络。卷积网络有很多种较为复杂的变形，在图像处理领域有很广泛的应用。

我们本书会从基本原理入手，讲解卷积网络的构成和意义。然后使用 pytorch 搭建并训练一个卷积神经网络模型。

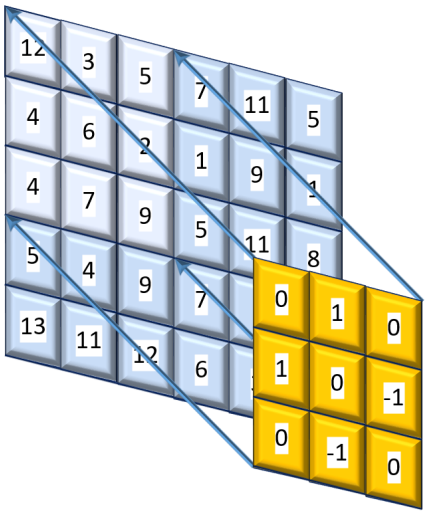
1. 卷积神经网络模型



本章从基本原理的角度来描述卷积神经网络，给出一些一般概念的解释。

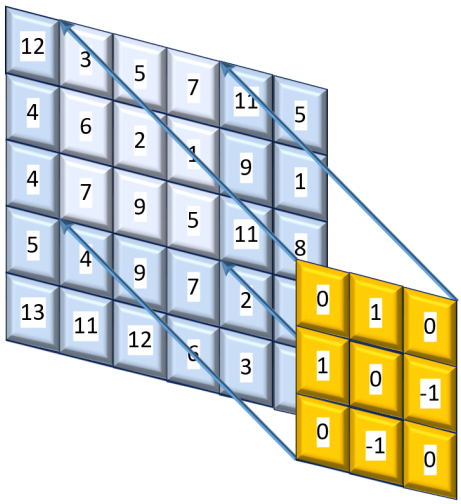
1.1 卷积、感受野与池化

图像卷积，就是使用卷积核中每个元素乘以图像中对应区域的每个值，再相加，比如下图中对左上角 3×3 区域的卷积：

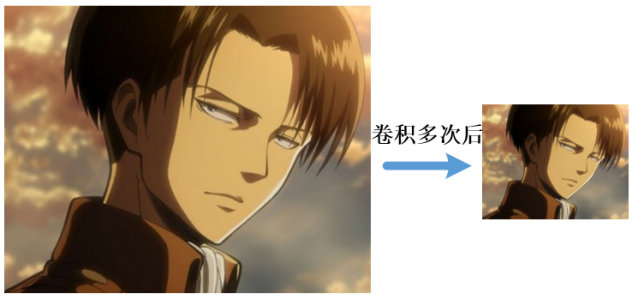


$$12\times0+3\times1+5\times0+4\times1+6\times0+2\times(-1)+4\times0+7\times(-1)+9\times0=-2 \tag{1.1.1}$$

然后平移卷积核并再次执行卷积：



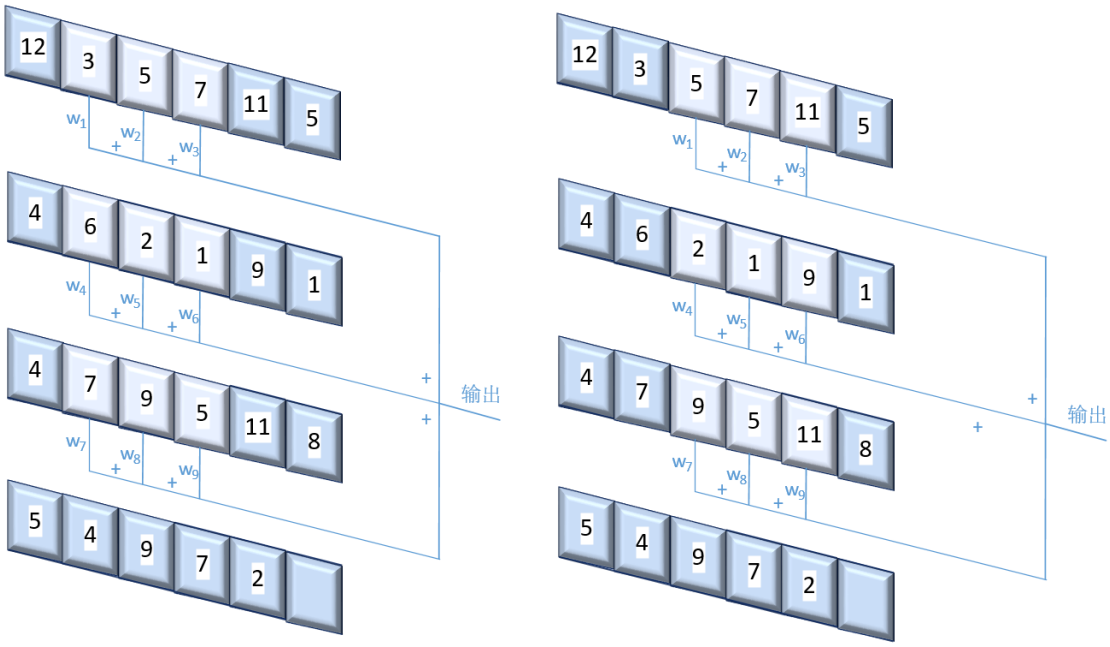
将整个图像区域遍历一遍以后，原来的 5×6 的图像就能得到 $(5-2)\times(6-2)$ 的卷积后的图像。一幅图像在卷积多次以后，我们还是能分辨出来这个图像描述的内容，例如下图中的“兵长”。所以在进行图像识别过程中，或许我们并没有把图像整体丢进网络的必要。



卷积网络就是给图像输入加几个卷积层，然后把卷积后的结果“Flatten”成向量，再把这个向量输入到全连接网络中。

卷积核是左右上下移动来实现对整个图像进行卷积的，移动的步长称为“stride”，一般设置为 1 就表示一次只移动一个像素的步长。

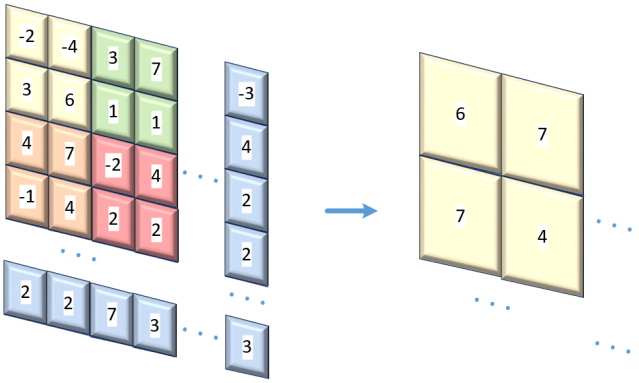
一个卷积核可以总结一个图像中的特征类型，对一幅图像，我们可以使用多个卷积核来得到多幅图像特征（feature map）。卷积层中，每一层其实就是全连接网络中去掉某些部分得到的：对于 3×3 的卷积核，5×6 的图像得到 $(5-2)\times(6-2)$ 的输出，而每个输出并不是前一层图像所有像素加权的输入，而只是 3×3 个像素的加权输入（换言之，参数相比于全连接网络来说变得非常少）。而且我们可以看到卷积层的网络参数权重都是一样的：



卷积层的训练其实就可以对每个卷积区域都训练一个卷积核，然后把所有卷积核取平均，得到更新以后的卷积核。

卷积神经网络的感受野表示当前层的 feature map 中每个标量值表示输入图像映射区域的大小。例如对于 3×3 的卷积核，卷积一次以后，第二层的感受野就是 3×3，再卷积一次以后，第三层的感受野就是 5×5。如果卷积核是 5×5 的，那么卷积一次以后的感受野就是 5×5 的大小。

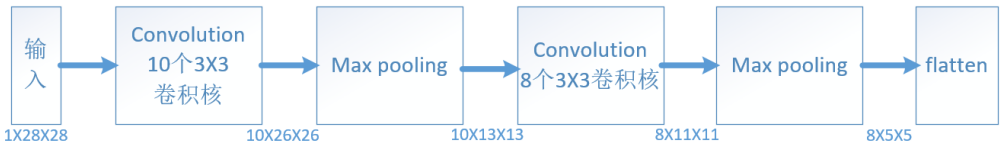
3×3 的卷积核卷积一次以后，100×100 的图像得到 98×98 的输出，我们是不是需要将这 98×98 的输出全部用于下一层的计算呢？其实可以使用池化来进行下采样操作。比如对生成的 98×98 的图像，我们把每 2×2 的局部区域提取一个最大值，得到最大池化（max pooling）的结果：



1.2 卷积神经网络

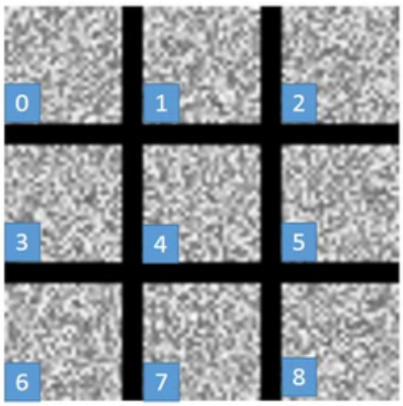
假设我们的网络第一层有 10 个卷积核，我们将一张图像卷积以后，就得到了 10 个 feature map。这个时候，输入的数据其实就是一个立方体了（多层图像叠加）。在下一层的卷积层中（假如该层有 8 个卷积核），卷积核不再是一个平面 3×3 的核，而是一个 $3\times3\times8$ 的立方体核，这样，

10 个 feature map 卷积以后得到 8 个 feature map，而不是得到 10×8 个 feature map。



对于训练好的神经网络，我们可以尝试将输入定为参数，然后将某一层的每个 feature map 中所有元素相加的和设为输出，然后通过 Gradient ascent（梯度上升）方法来找到可以使输出最大化的输入，以此来猜测神经网络到底学到了些什么。

我们也可以找到使得后面的全连接层甚至输出层最大化的输入，但我们可以根据训练结果发现，使得网络最大化激活的输入不一定我们能够理解。比如在手写数字辨识中，我们可以看到可以最大化输出 0-9 的输入图像表示如下：



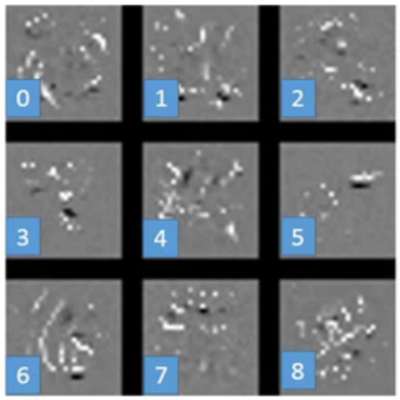
最大化输出（y_i 表示输出为 i 的概率，这个概率可以由 softmax 和交叉熵来得到）：

$$arg \max_{input} y_i \tag{1.2.1}$$

我们对输入设定一个限制，让输入的像素值之和尽可能小（因为手写数字辨识中大部分区域没有笔画）：

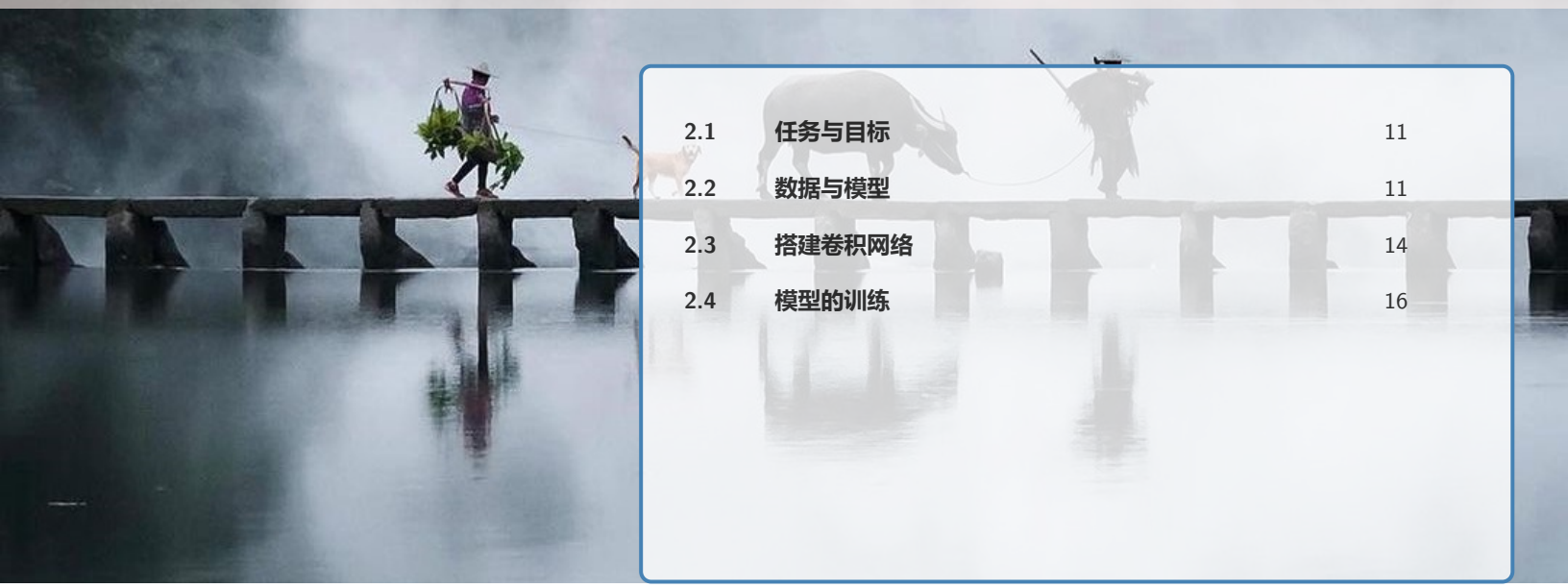
$$arg \max_{input} \left(y_i - \sum_{i,j} |x_{i,j}| \right) \tag{1.2.2}$$

这样得到的可以最大化输出的输入图像如下，可以看到具备了一些实际意义。



我们会在《神经网络的可解释性实战——pytorch 版》中对这些内容进行更详细的描述。至于 visualization 的结果是不是就是“可解释的”，或许我们只能将显示得看着像“2”的东西解释为“2”。而机器到底学到了些什么，这似乎更难去理解，这也不作为本书的讨论范畴。

2. 卷积神经网络实战



本章开始描述如何搭建并训练一个简单的卷积神经网络。

2.1 任务与目标

我们的任务是对图像进行分类。图像分类数据集大家可以从网上自行搜索，这里我给出几个链接 [3][4][5][6]，我们以课程 [1] 的作业 3[7] 给出的数据集为例，进行食物的分类。其他图像分类任务也是大同小异。

food-11 数据集有三个文件夹分别是 training、validation 和 testing。训练集和验证集的数据名称都是“类别标签 + 编号”，例如第 1 类第 1 个数据是“0-0”。因为我们并不需要将 test 结果上传到 kaggle 上测试，我们只需要用训练集来训练数据，然后用验证集来验证我们的模型即可。

2.2 数据与模型

本节代码见 Chapter2-1.py。

我们先用程序把数据导出：

```
1  import numpy as np
2  import torch
3  import torch.nn as nn
4  import torchvision.transforms as transforms
5  import cv2
6  import os
7  #总文件目录
8  data_dir = './food-11'
9  #根据路径读取图像数据文件
10 def readfile(path):
```

```
    image_dir = sorted(os.listdir(path))
    x = np.zeros((len(image_dir), 128, 128, 3), dtype=np.uint8)
    y = np.zeros((len(image_dir)), dtype=np.uint8)
    for i, file in enumerate(image_dir):
        img = cv2.imread(os.path.join(path, file))
        x[i, :, :] = cv2.resize(img, (128, 128))
        y[i] = int(file.split("_")[0])
    return x, y
#生成训练和验证数据集
train_x, train_y = readfile(os.path.join(data_dir, "training"))
val_x, val_y = readfile(os.path.join(data_dir, "validation"))
```

打印一下训练集的数据格式，可以得到：

```
print(train_x.shape)
# 输出：
# (9866, 128, 128, 3)
```

说明有 9866 张图像，每张图像大小为 128×128，每个图像有三通道。

我们定义生成 Dataset 数据集的类：

```
from torch.utils.data import DataLoader, Dataset
class ImgDataset(Dataset):
  def __init__(self, x, y=None, transform=None):
    self.x = x
    self.y = y
    if y is not None:
      # 需要把label设置为64位有符号整形
      self.y = torch.LongTensor(y)
    self.transform = transform
  def __len__(self):
    return len(self.x)
  def __getitem__(self, index):
    X = self.x[index]
    if self.transform is not None:
      X = self.transform(X)
    if self.y is not None:
      Y = self.y[index]
      return X, Y
    else:
      return X
```

以及定义数据转换方式：

```
import torchvision.transforms as transforms
# 训练集数据每次提取的时候都会进行一些随机旋转等方法来获得不一样的数据，起到
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(),  # 隨機將圖片水平翻轉
    transforms.RandomRotation(15),  # 隨機旋轉圖片
    transforms.ToTensor(),  # 將圖片轉成 Tensor
])
# 提取测试集不需做 data augmentation(数据扩增)
test_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
])
```

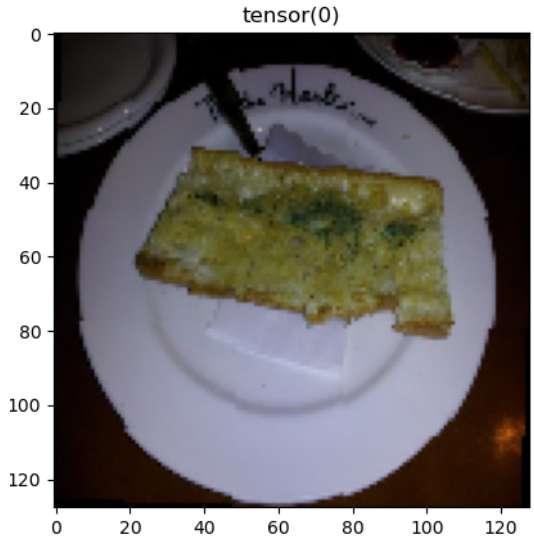
然后定义数据加载：

```
batch_size = 128
train_set = ImgDataset(train_x, train_y, train_transform)
val_set = ImgDataset(val_x, val_y, test_transform)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=False)
```

我们将图像进行一下显示：

```
import matplotlib.pyplot as plt
figure = plt.figure()
img, label = train_set[100]
#squeeze函数把为1的维度去掉，然后把(3,128,128)的数据格式转换为(128,128,3)
img = img.squeeze().permute(1,2,0)
# OpenCV读取的图像是BGR格式的，要这样转换为RGB格式
img = img[...,[2,1,0]]
plt.title(label)
plt.imshow(img, cmap="gray")
plt.show()
```

得到显示结果：



2.3 搭建卷积网络

本节代码见 Chapter2-2.py。

在搭建卷积网络之前，先简单介绍一下 Batch Normalization。在卷积神经网络的卷积层之后，程序里总会添加 BatchNorm2d 进行数据的归一化处理，举个例子：

```
nn.Conv2d(3, 64, 3, 1, 1),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.MaxPool2d(2, 2, 0),
```

在进行模型训练之前，需对数据做归一化处理，使其分布一致。而在深度神经网络训练过程中，通常一次训练是一个 batch，而非全体数据，每个 batch 可能具有不同的分布。Batch Normalization 将数据拉回到均值为 0，方差为 1 的正态分布上 (归一化)，一方面使得数据分布一致，另一方面避免梯度消失、梯度爆炸 [8]。

```
# torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride,
    padding)
# torch.nn.MaxPool2d(kernel_size, stride, padding)
# input 维度 [3, 128, 128]
class Classifier(nn.Module):
  def __init__(self):
    super(Classifier, self).__init__()
    self.convolution = nn.Sequential(
      nn.Conv2d(3, 64, 3, 1, 1), # [64, 128, 128]
      # 对小批量（batch)3d（加上通道数）数据组成的4d输入进行标准化操作
      nn.BatchNorm2d(64),
```

```
      nn.ReLU(),
      nn.MaxPool2d(2, 2, 0),  # [64, 64, 64]

      nn.Conv2d(64, 128, 3, 1, 1),  # [128, 64, 64]
      nn.BatchNorm2d(128),
      nn.ReLU(),
      nn.MaxPool2d(2, 2, 0),  # [128, 32, 32]

      nn.Conv2d(128, 256, 3, 1, 1),  # [256, 32, 32]
      nn.BatchNorm2d(256),
      nn.ReLU(),
      nn.MaxPool2d(2, 2, 0),  # [256, 16, 16]

      nn.Conv2d(256, 512, 3, 1, 1),  # [512, 16, 16]
      nn.BatchNorm2d(512),
      nn.ReLU(),
      nn.MaxPool2d(2, 2, 0),  # [512, 8, 8]

      nn.Conv2d(512, 512, 3, 1, 1),  # [512, 8, 8]
      nn.BatchNorm2d(512),
      nn.ReLU(),
      nn.MaxPool2d(2, 2, 0),  # [512, 4, 4]
    )
    self.fc = nn.Sequential(
      nn.Linear(512 * 4 * 4, 1024),
      nn.ReLU(),
      nn.Linear(1024, 512),
      nn.ReLU(),
      nn.Linear(512, 11)
    )
  def forward(self, x):
    out = self.convolution(x)
    #print(out.size())
    #print(out.size()[0])
    out = out.view(out.size()[0], -1)
    #print(out.size())
    return self.fc(out)
```

我们在 forward 函数中注释掉的部分是查看输出结果的尺寸，这里显示一下：

```
torch.Size([128, 512, 4, 4])
```

```
128
torch.Size([128, 8192])
```

out.size() 的第 0 维表示 batch size 的大小，我们设置的是 128。

我们生成模型，并定义损失函数与优化器：

```
model = Classifier().cuda()
loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

2.4 模型的训练

本节代码仍见 Chapter2-2.py。

训练前先定义训练轮数。这里我们导入 time 包来计算时间。

```
num_epoch = 100
import time
for epoch in range(num_epoch):
  # for循环里面的内容
  ......
```

现在我们开始实现 for 循环里面的内容：

```
  epoch_start_time = time.time()
  train_acc = 0.0
  train_loss = 0.0
  val_acc = 0.0
  val_loss = 0.0
  # model.train()的作用是启用batch normalization和drop out
  model.train()
  for i, data in enumerate(train_loader):
    # 进行训练
    ......
  # 在model(test_datasets)之前，需要加上model.eval()。否则的话，有输入
    数据，即使不训练，它也会改变权值。
  model.eval()
  with torch.no_grad():
    # 进行测试
    ......
```

首先是进行训练的程序：

```
  optimizer.zero_grad()
```

```
    # 计算
    train_pred = model(data[0].cuda()) #data[0]表示训练用的数据
    batch_loss = loss(train_pred, data[1].cuda())
    batch_loss.backward()
    optimizer.step()
    #argmax返回最大的索引值。因为最大值的索引就是分的类别。
    train_acc += (train_pred.cpu().argmax(1) == data[1]).type(torch.
        float).sum().item()
    train_loss += batch_loss.item()
```

然后是来测试的程序：

```
for i, data in enumerate(val_loader):
    val_pred = model(data[0].cuda())
    batch_loss = loss(val_pred, data[1].cuda())
    val_acc += (val_pred.cpu().argmax(1) == data[1]).type(torch.float).
        sum().item()
    val_loss += batch_loss.item()
# 将结果 print 出來
print('[%03d/%03d] %2.2f sec(s) Train Acc: %3.6f Loss: %3.6f | Val Acc
    : %3.6f loss: %3.6f' % \
     (epoch + 1, num_epoch, time.time() - epoch_start_time, \
      train_acc / train_set.__len__(), train_loss / train_set.__len__()
          , val_acc / val_set.__len__(),
      val_loss / val_set.__len__()))
```

现在我们已经可以开始训练卷积网络模型了。由于训练结果并没有什么特别的，所以这里也就不再进行展示了，大家可以看到打印输出的 acc 在逐步上升。

Bibliography

[1] https://www.bilibili.com/video/BV1JE411g7XF?p=17

[2] 周志华. 《机器学习》[J]. 中国民商, 2016, 03(No.21):93-93.

[3] http://www.vision.caltech.edu/Image_Datasets/Caltech101/

[4] http://www.vision.caltech.edu/Image_Datasets/Caltech256/

[5] http://host.robots.ox.ac.uk/pascal/VOC/

[6] https://www.robots.ox.ac.uk/ vgg/data/flowers/17/

[7] http://speech.ee.ntu.edu.tw/ tlkagk/courses_ML20.html

[8] https://arxiv.org/abs/1502.03167