# 卷积神经网络CNN

# 目录

Contents

# 学习目标

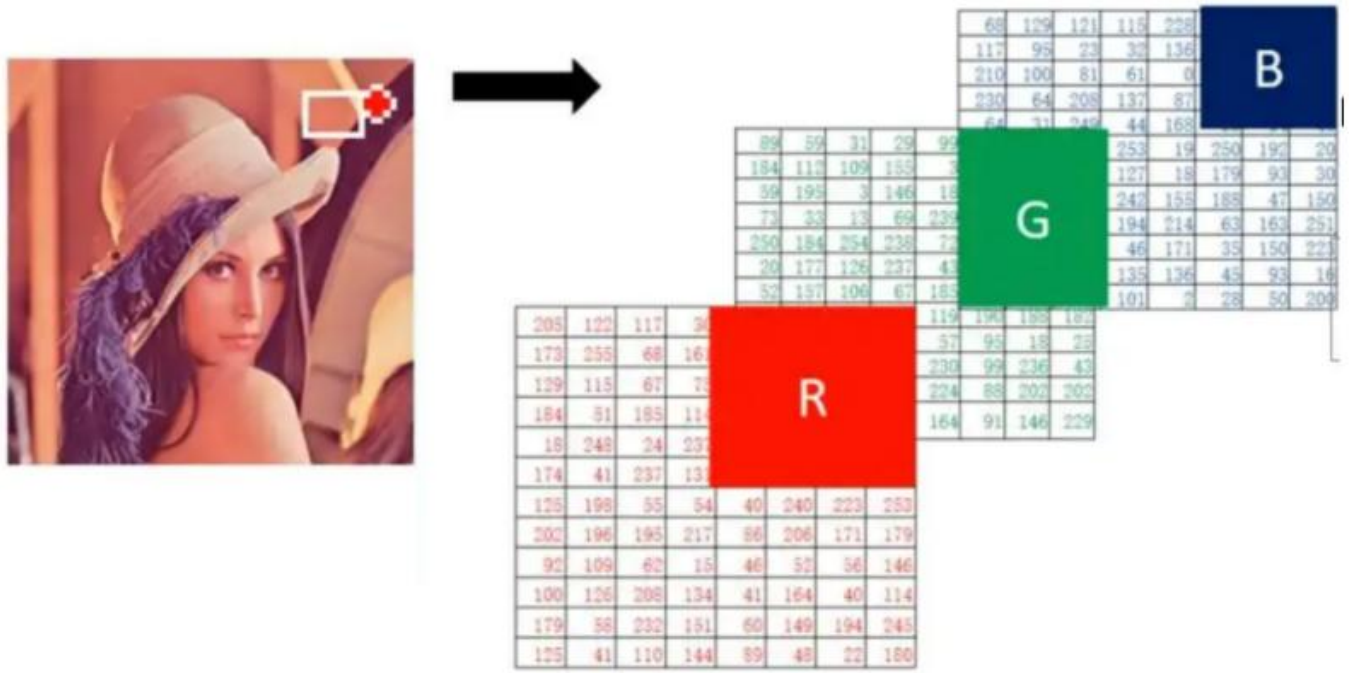
Learning Objectives

1. 知道图像的基本概念
2. 掌握使用matplotlib加载图片方法

## 图像基本概念

图像是由像素点组成的，每个像素点的取值范围为: [0, 255] 。像素值越接近于0，颜色越暗，接近于黑色；像素值越接近于255，颜色越亮，接近于白色。

在深度学习中，我们使用的图像大多是彩色图，彩色图由RGB3个通道组成，如下图所示



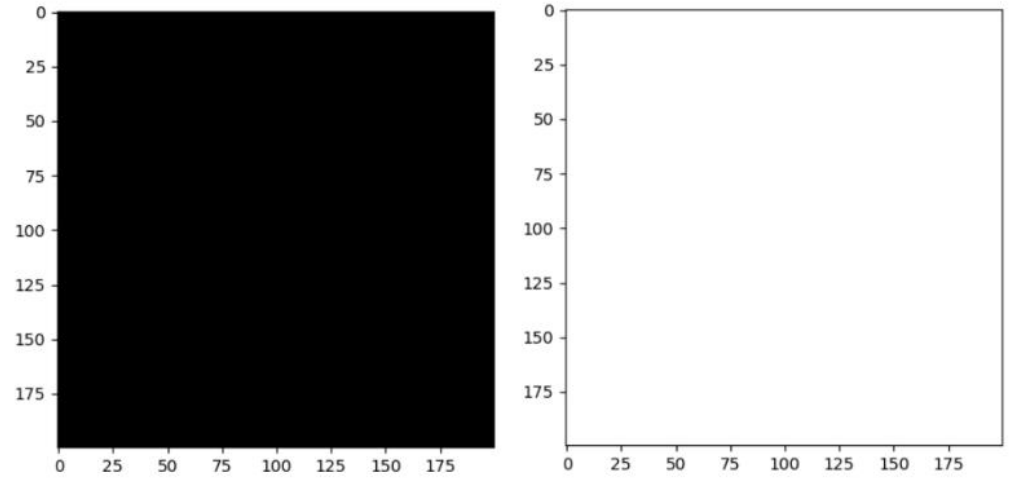
## 图像的加载

使用 *matplotlib* 库来实际理解下上面讲解的图像知识。

```
import numpy as np
import matplotlib.pyplot as plt
# 像素值的理解
def test01():
    # 全0数组是黑色的图像
    img = np.zeros([200, 200, 3])
    # 展示图像
    plt.imshow(img)
    plt.show()
    # 全255数组是白色的图像
    img = np.full([200, 200, 3], 255)
    # 展示图像
    plt.imshow(img)
    plt.show()
```

程序输出结果:

全黑和全白图像:

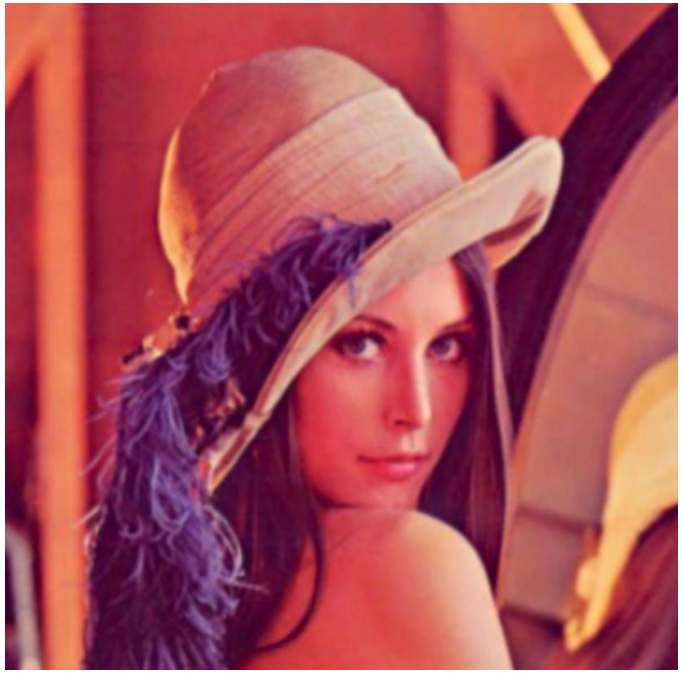


# 图像的加载

```
# 图像的加载
def test02():
    # 读取图像
    img = plt.imread("data/img.jpg")
    # 图像形状 高,宽,通道
    print("图像的形状(H, W, C):\n", img.shape)
    # 展示图像
    plt.imshow(img)
    plt.axis("off")
    plt.show()
```

图像的形状为：

```
图像的形状（H，W，C）：
 (640, 640, 3)
```



# 总结

1、图像的构成

由像素点构成，【0-255】，RGB，【HWC】

2、图像的加载方法

Plt.imread()

Plt.imshow()

# 目录

Contents

# 学习目标

Learning Objectives

1. 知道什么是卷积神经网络
2. 知道卷积神经网络组成部分

## CNN概述

卷积神经网络（Convolutional Neural Network）是含有卷积层的神经网络. 卷积层的作用就是用来自动学习、提取图像的特征.

CNN网络主要由三部分构成：卷积层、池化层和全连接层构成：

1. 卷积层负责提取图像中的局部特征;
2. 池化层用来大幅降低参数量级(降维);
3. 全连接层用来输出想要的结果。

## CNN概述

图中CNN要做的事情是：给定一张图片，是车还是马未知，是什么车也未知，现在需要模型判断这张图片里具体是一个什么东西，总之输出一个结果：如果是车 那是什么车

最左边是

数据输入层：对数据做一些处理，比如去均值（各维度都减对应维度的均值，使得输入数据各个维度都中心化为0，避免数据过多偏差，影响训练效果）、归一化（把所有的数据都归一到同样的范围）、PCA等等。CNN只对训练集做“去均值”这一步。

中间是

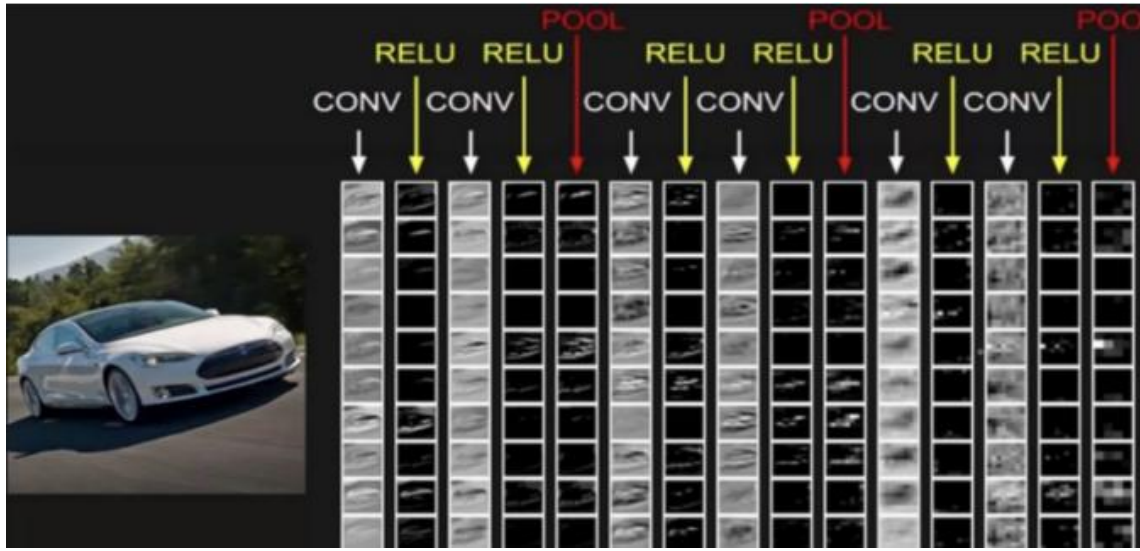
卷积层(CONV)：线性乘积求和，提取图像中的局部特征

激励层(RELU)：ReLU激活函数,输入数据转换成输出数据

池化层(POOL)：取区域平均值或最大值，大幅降低参数量级(降维)

最右边是

全连接层(FC)：输出CNN模型预测结果



# 总结

1. 什么是卷积神经网络？

包含卷积层的神经网络

2. 卷积神经网络的构成

卷积层：特征提取

池化层：降维

全连接层：输出结果

# 目录

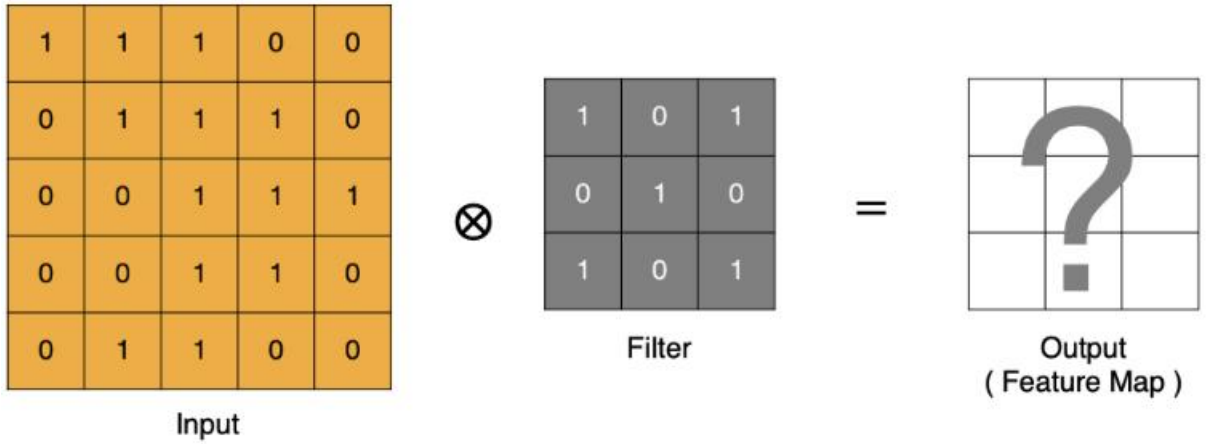
Contents

# 学习目标

Learning Objectives

1. 掌握卷积层计算过程
2. 掌握特征图大小计算方法
3. 掌握PyTorch卷积层API

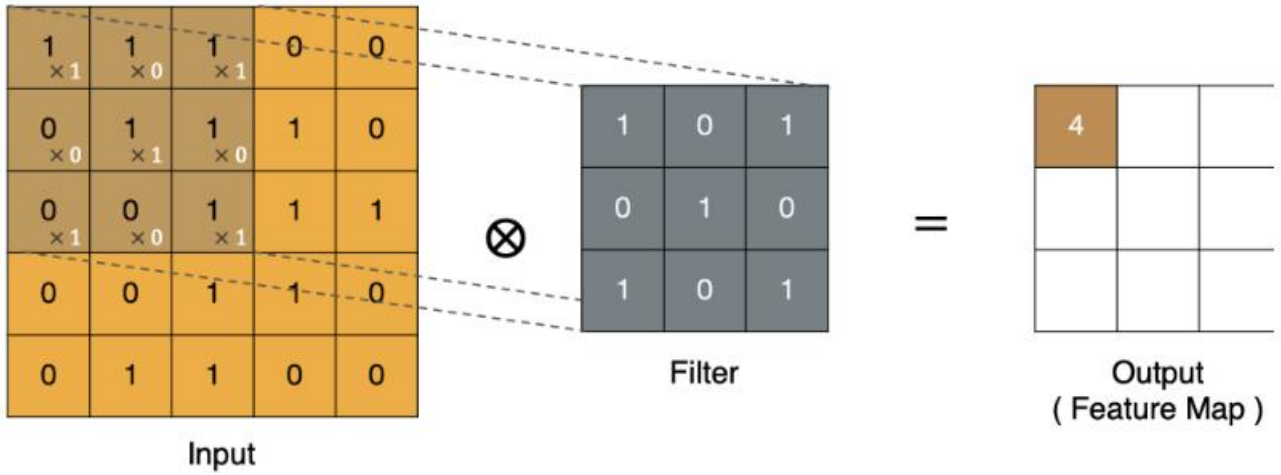
## 卷积计算



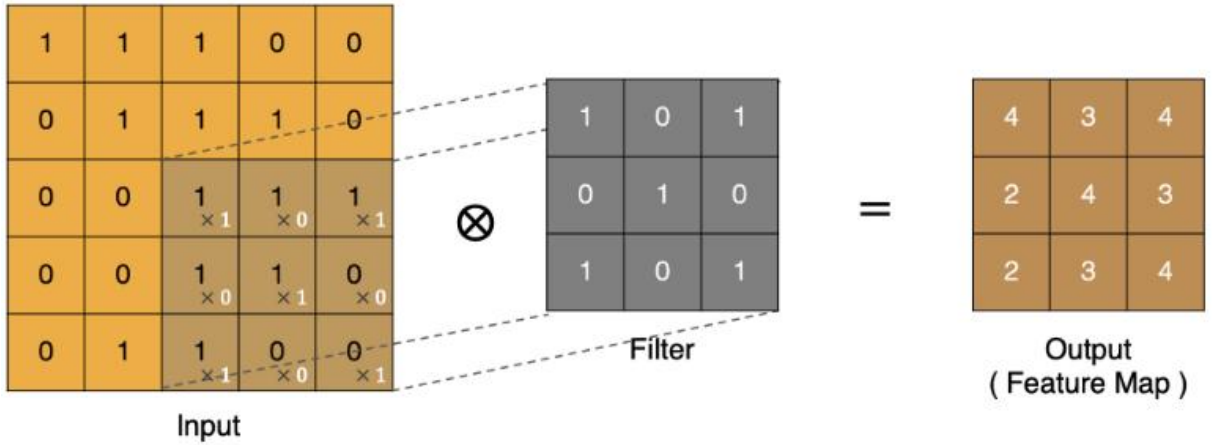

1. input 表示输入的图像
2. filter 表示卷积核，也叫做卷积核(滤波矩阵)
3. input 经过 filter 得到输出为最右侧的图像，该图叫做特征图

## 卷积计算

卷积运算本质上就是在卷积核和输入数据的局部区域间做点积。




最终的特征图结果为:

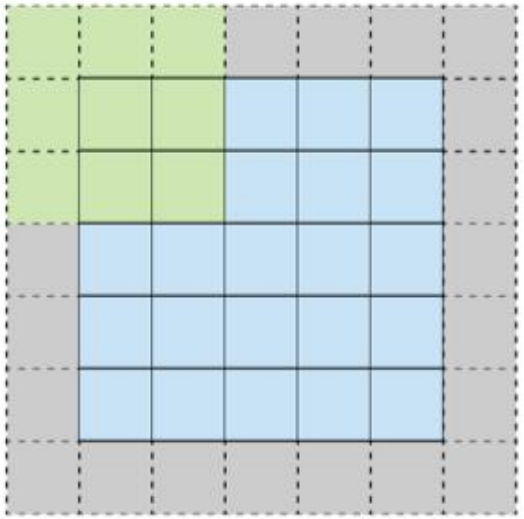


## Padding（填充）

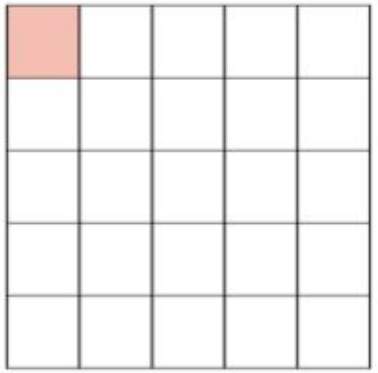
通过上面的卷积计算过程，最终的特征图比原始图像小很多，如果想要保持经过卷积后的图像大小不变，可以在原图周围添加 padding 来实现。

padding（填充）操作用于处理卷积时图像边缘的像素。

其目的是在输入图像的边界周围添加额外的像素（通常是零），从而解决卷积操作时边缘信息丢失的问题。



Stride 1 with Padding

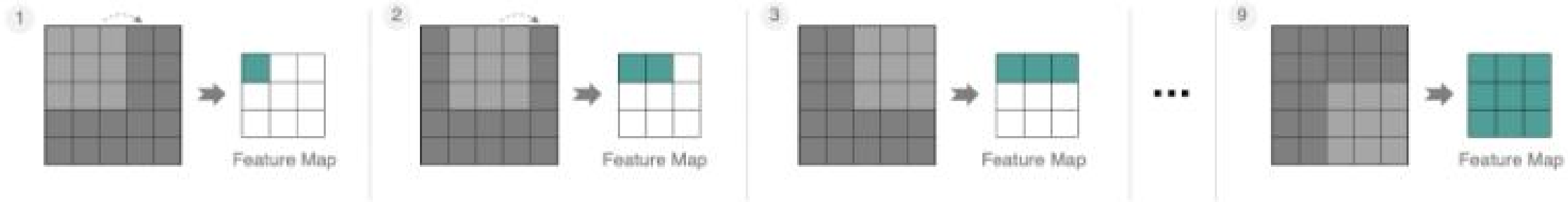


Feature Map

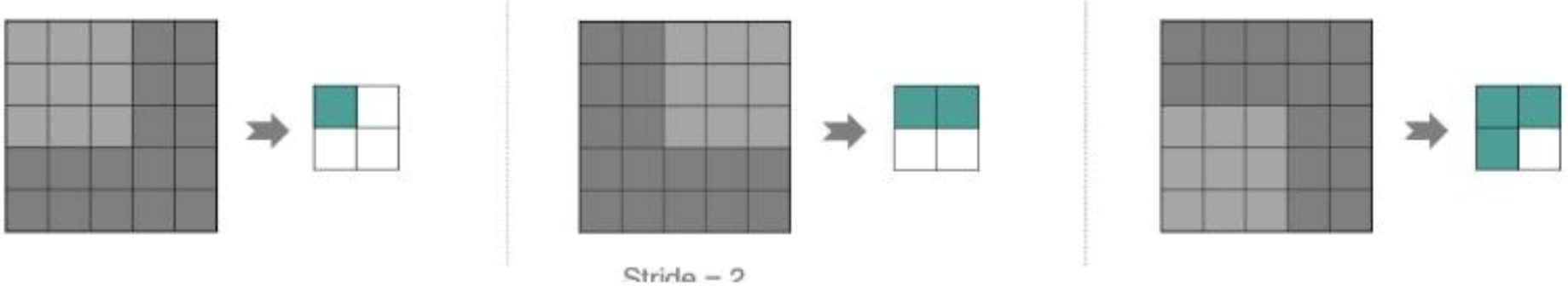
## Stride（步长）

stride（步长）指的是卷积核在图像上滑动时的步伐大小，即每次卷积时卷积核在图像中向右（或向下）移动的像素数。步长直接影响卷积操作后输出特征图的尺寸，以及计算量和模型的特征提取能力。

按照步长为1来移动卷积核，计算特征图如下所示：

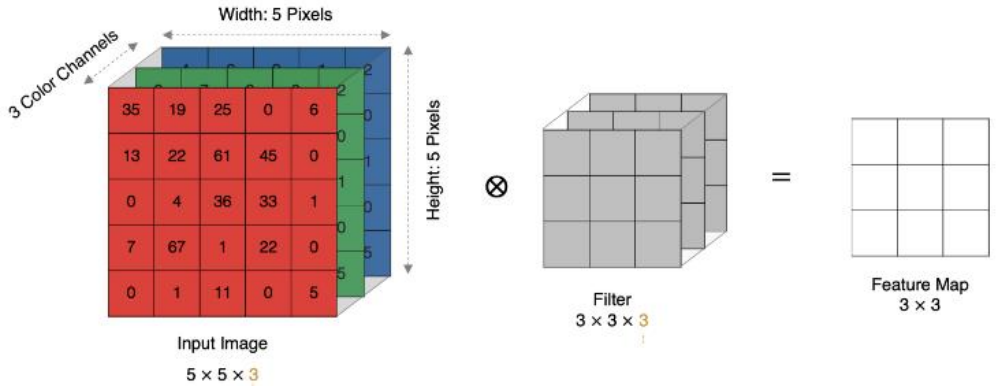



如果把 Stride 增大为2，也是可以提取特征图的，如下图所示：

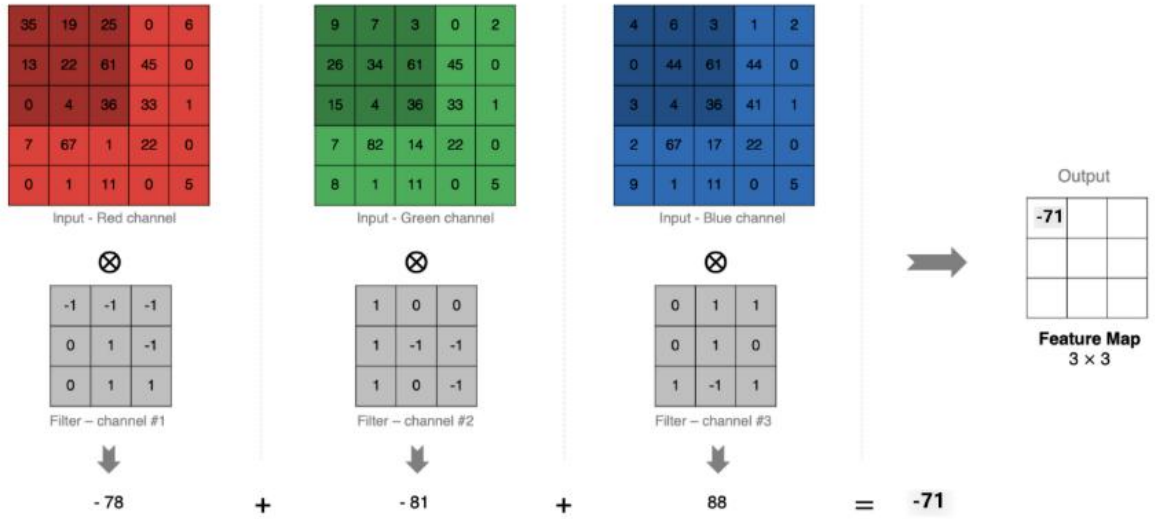


## 多通道卷积计算

实际中的图像都是多个通道组成的，我们怎么计算卷积呢？

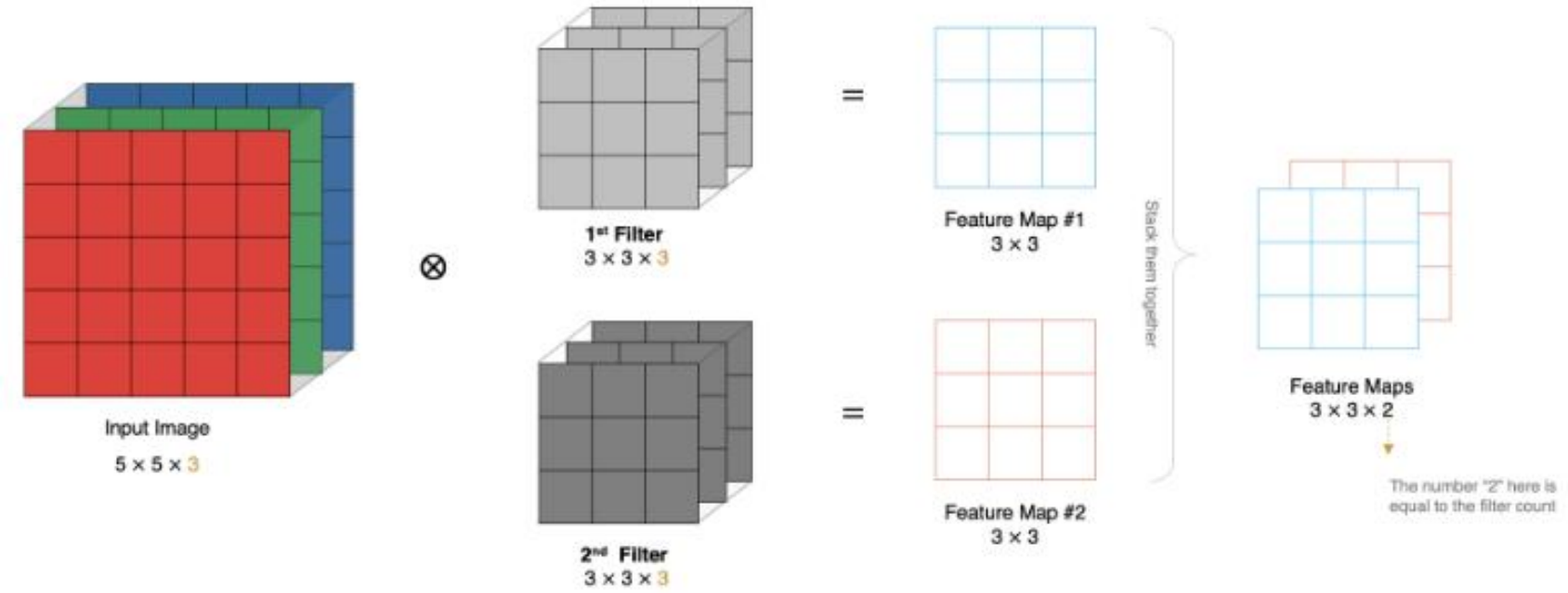



如下图所示：

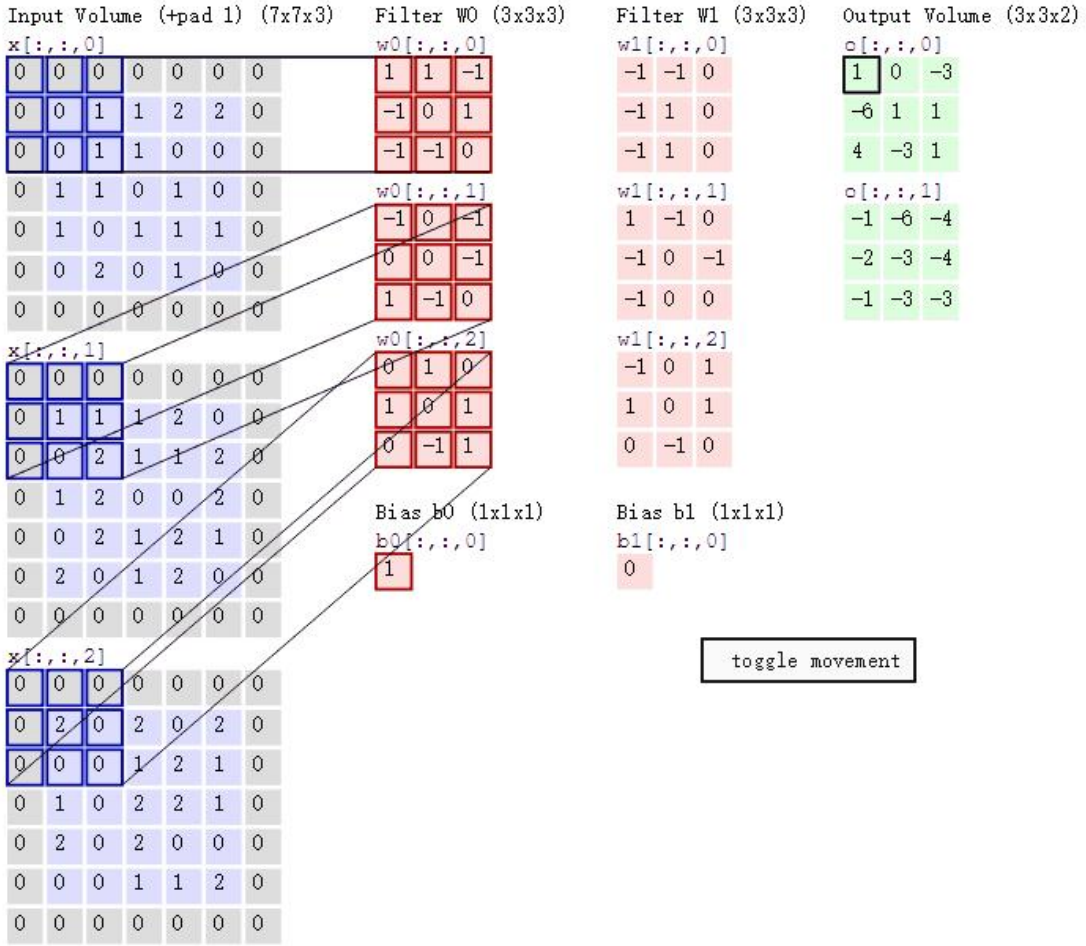


## 多卷积核卷积计算

当使用多个卷积核时，应该怎么进行特征提取呢？



# 多卷积核卷积计算


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 1 2 2 0
0 0 1 1 0 0 0
0 1 1 0 1 0 0
0 1 0 1 1 1 0
0 0 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 0 2 1 1 2 0
0 1 2 0 0 2 0
0 0 2 1 2 1 0
0 2 0 1 2 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 0 2 0 2 0
0 0 0 1 2 1 0
0 1 0 2 2 1 0
0 2 0 2 0 0 0
0 0 0 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 -1
-1 0 1
-1 -1 0
w0[:,:,1]
-1 0 -1
0 0 -1
1 -1 0
w0[:,:,2]
0 1 0
1 0 1
0 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 0
-1 1 0
w1[:,:,1]
1 -1 0
-1 0 -1
-1 0 0
w1[:,:,2]
-1 0 1
1 0 1
0 -1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
1 0 -3
-6 1 1
4 -3 1
o[:,:,1]
-1 -6 -4
-2 -3 -4
-1 -3 -3
toggle movement

## 特征图大小

输出特征图的大小与以下参数息息相关:

1. size: 卷积核/过滤器大小，一般会选择为奇数，比如有 1*1 、3*3、5*5
2. Padding: 零填充的方式
3. Stride: 步长

那计算方法如下图所示:

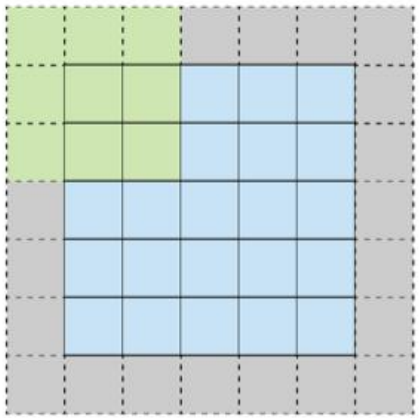
1. 输入图像大小: W x W
2. 卷积核大小: F x F
3. Stride: S
4. Padding: P
5. 输出图像大小: N x N

$$N = \frac{W - F + 2P}{S} + 1$$

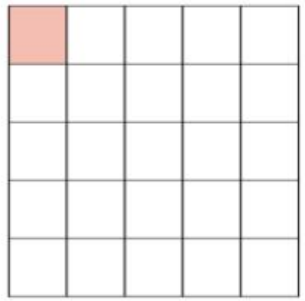
## 特征图大小

以下图为例:

1. 图像大小: 5 x 5
2. 卷积核大小: 3 x 3
3. Stride: 1
4. Padding: 1
5. (5 - 3 + 2) / 1 + 1 = 5, 即得到的特征图大小为: 5 x 5



Stride 1 with Padding



Feature Map

# PyTorch卷积层API

```
conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
"""
参数说明:
in_channels: 输入通道数,
out_channels: 输出通道，也可以理解为卷积核kernel的数量
kernel_size：卷积核的高和宽设置，一般为3,5,7...
stride：卷积核移动的步长
padding：在四周加入padding的数量，默认补0
"""
```

## PyTorch卷积层API

```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
def test():
    # 读取图像, 形状: (640, 640, 3)
    img = plt.imread('data/img.jpg')
    plt.imshow(img)
    plt.axis('off')
    plt.show()
    # 构建卷积层
    # out_channels表示卷积核个数
    # 修改out_channels，stride，padding观察特征图的变化情况
    conv = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=2, padding=0)
    # 输入形状: (BatchSize, Channel, Height, Width)
    # mg形状: torch.Size([3, 640, 640])
    img = torch.tensor(img).permute(2, 0, 1)
    # img 形状: torch.Size([1, 3, 640, 640])
    img = img.unsqueeze(0)
    # 将图像送入卷积层中
    feature_map_img = conv(img.to(torch.float32))
    # 打印特征图的形状
    print(feature_map_img.shape)
if __name__ == '__main__':
    test()
```

# 总结

1、卷积层计算过程

卷积运算本质上就是在卷积核和输入数据的局部区域间

做点积

2、特征图大小计算方法

$$N = \frac{W - F + 2P}{S} + 1$$

3、PyTorch卷积层API

```
nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
```

# 目录

Contents

# 学习目标

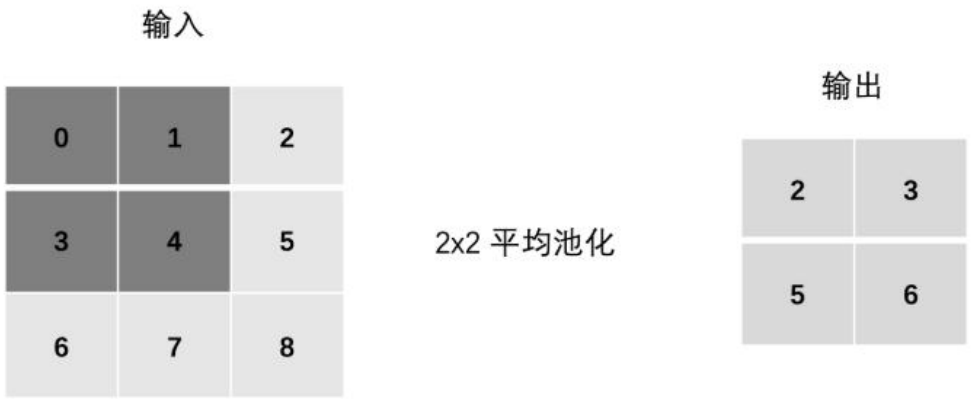
Learning Objectives

1. 掌握池化层计算过程
2. 掌握PyTorch池化层API

## 池化层计算

**池化层（Pooling）降低维度，从而减少计算量、减少内存消耗，并提高模型的鲁棒性。**

池化层通常位于卷积层之后，它通过对卷积层输出的特征图进行下采样，保留最重要的特征信息，同时丢弃一些不重要的细节。

Padding

输入

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 1 | 2 | |
| | 3 | 4 | 5 | |
| | 6 | 7 | 8 | |
| | | | | |

2x2 最大池化

输出

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 2 |
| 3 | 4 | 5 | 5 |
| 6 | 7 | 8 | 8 |
| 6 | 7 | 8 | 8 |

输入

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 1 | 2 | |
| | 3 | 4 | 5 | |
| | 6 | 7 | 8 | |
| | | | | |

2x2 平均池化

输出

| | | | |
|---|---|---|---|
| 0 | 0.25 | 0.75 | 0.5 |
| 0.75 | 2 | 3 | 1.75 |
| 2.25 | 5 | 6 | 3.25 |
| 1.5 | 3.25 | 3.75 | 2 |

Stride

输入

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

2x2 最大池化, Stride=2

输出

| | |
|---|---|
| 5 | 7 |
| 13 | 15 |

输入

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

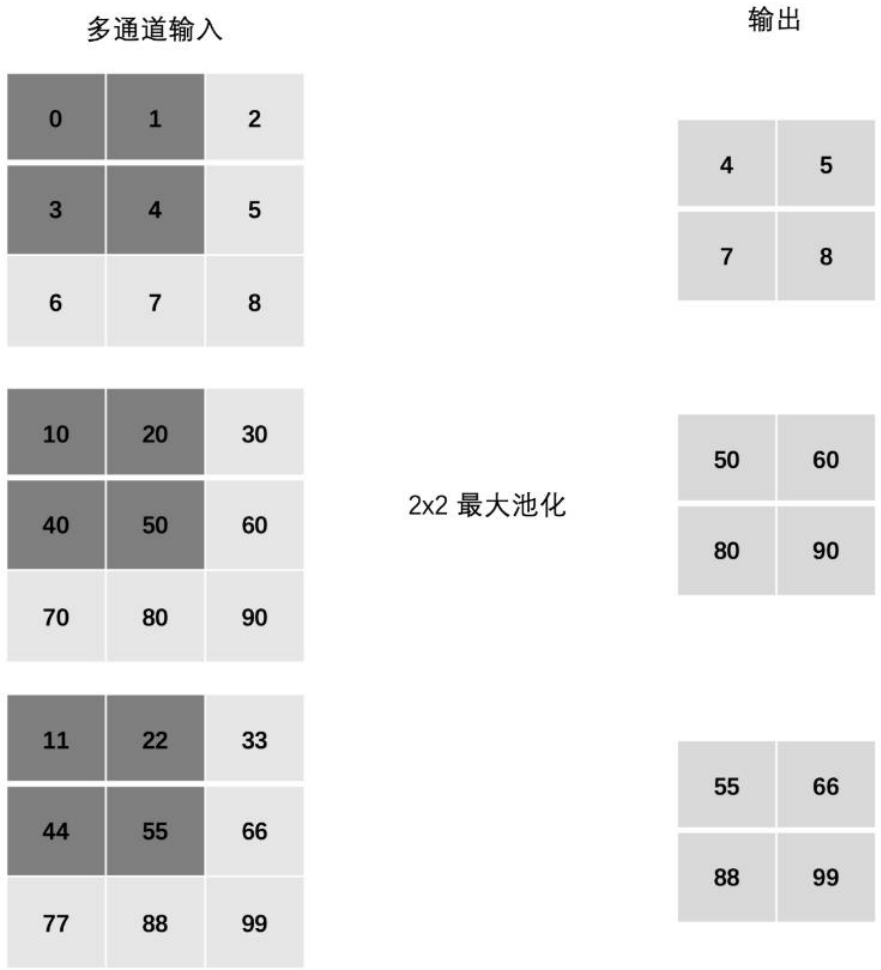
2x2 平均池化, Stride=2

输出

| | |
|---|---|
| 2.5 | 4.5 |
| 10.5 | 12.5 |

## 多通道池化层计算

在处理多通道输入数据时，池化层对每个输入通道分别池化，而不是像卷积层那样将各个通道的输入相加。这意味着池化层的输出和输入的通道数是相等。



## PyTorch 池化 API

```
# 最大池化
nn.MaxPool2d(kernel_size=2, stride=2, padding=1)
# 平均池化
nn.AvgPool2d(kernel_size=2, stride=1, padding=0)
```

# PyTorch 池化 API层

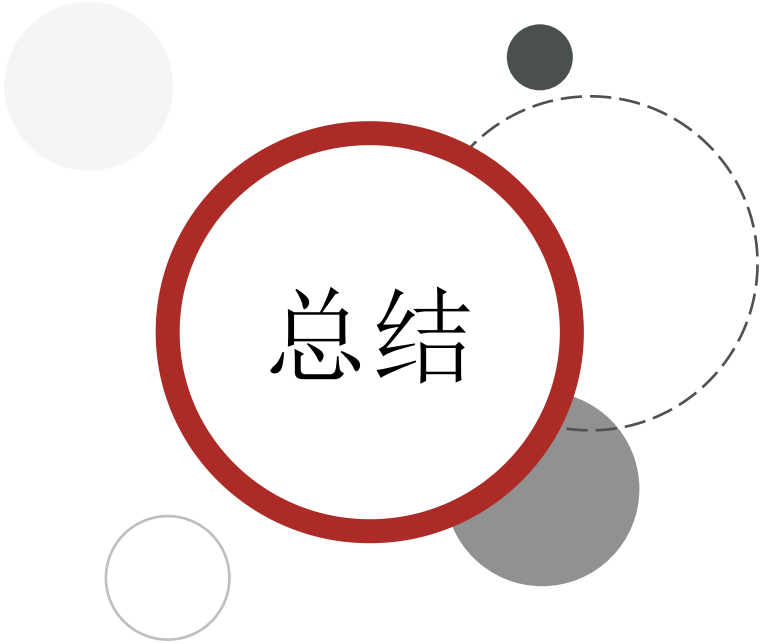
```
import torch
import torch.nn as nn

# 1. 单通道池化
def test01():
    # 定义输入输数据【1,3,3 】
    inputs = torch.tensor([[[0, 1, 2], [3, 4, 5], [6, 7, 8]]]).float()
    # 修改stride，padding观察效果
    # 1. 最大池化
    polling = nn.MaxPool2d(kernel_size=2, stride=1, padding=0)
    output = polling(inputs)
    print("最大池化：\n", output)

    # 2. 平均池化
    polling = nn.AvgPool2d(kernel_size=2, stride=1, padding=0)
    output = polling(inputs)
    print("平均池化：\n", output)
```

```
最大池化:
 tensor([[[4., 5.],
          [7., 8.]]])
平均池化:
 tensor([[[2., 3.],
          [5., 6.]]])
```

# PyTorch 池化 API层

```
# 2. 多通道池化
def test02():
    # 定义输入输数据【3,3,3 】
    inputs = torch.tensor([[[0, 1, 2], [3, 4, 5], [6, 7, 8]],
                           [[10, 20, 30], [40, 50, 60], [70, 80, 90]],
                           [[11, 22, 33], [44, 55, 66], [77, 88, 99]]]).float()

    # 最大池化
    polling = nn.MaxPool2d(kernel_size=2, stride=1, padding=0)
    output = polling(inputs)
    print("多通道池化：\n", output)


if __name__ == '__main__':
    test01()
    test02()
```

```
多通道池化：
 tensor([[[ 4.,  5.],
          [ 7.,  8.]],

         [[50., 60.],
          [80., 90.]],

         [[55., 66.],
          [88., 99.]]])
```




1.池化的作用

降维，减小数据的大小

2.池化的分类

最大池化和平均池化

3.池化的API

```
# 最大池化
nn.MaxPool2d(kernel_size=2, stride=2, padding=1)
# 平均池化
nn.AvgPool2d(kernel_size=2, stride=1, padding=0)
```

# 目录

Contents

# 学习目标

Learning Objectives

1. 了解CIFAR10数据集
2. 掌握分类网络搭建
3. 掌握模型构建流程

## 卷积神经网络案例

咱们使用前面学习到的知识来构建一个卷积神经网络，并训练该网络实现图像分类. 要完成这个案例，咱们需要学习的内容如下:

1. 了解 CIFAR10 数据集
2. 搭建卷积神经网络
3. 编写训练函数
4. 编写预测函数

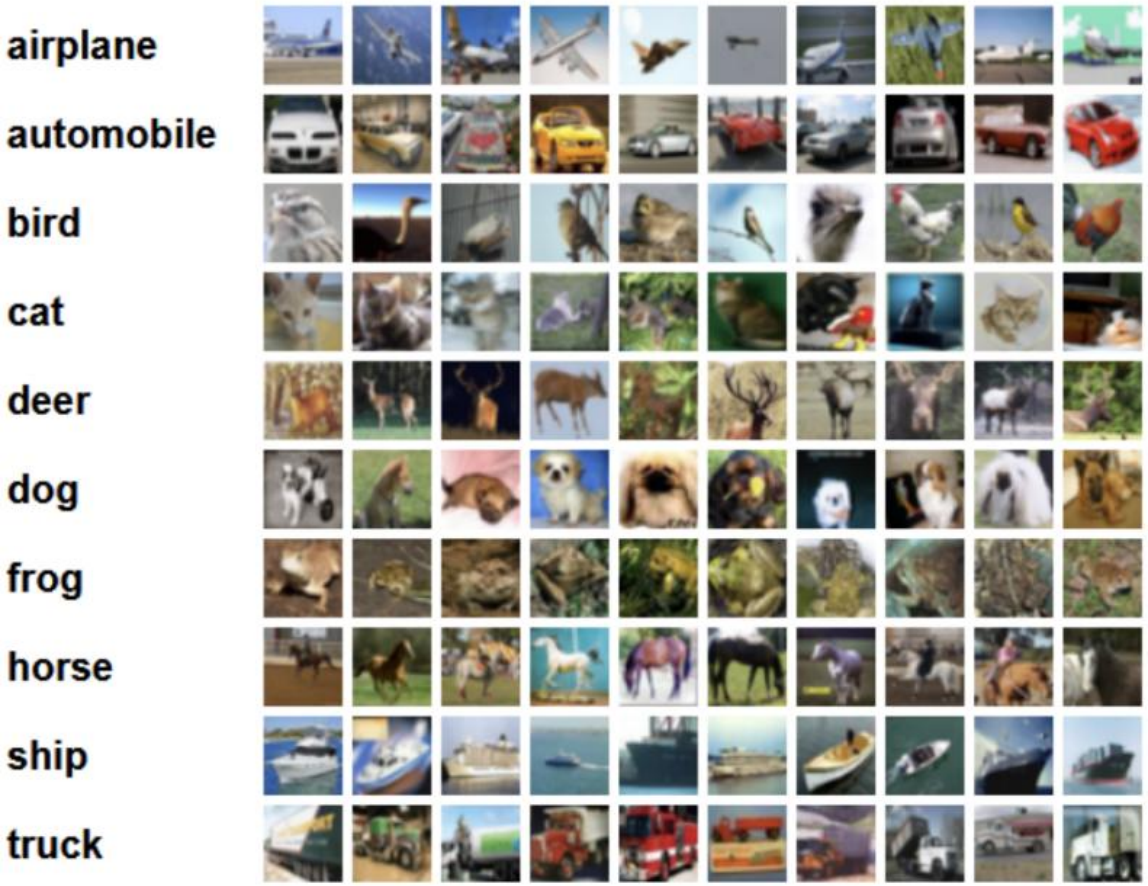
## 卷积神经网络案例

首先我们导入一下工具包：

```
import torch
import torch.nn as nn
from torchvision.datasets import CIFAR10
from torchvision.transforms import ToTensor
import torch.optim as optim
from torch.utils.data import DataLoader
import time
import matplotlib.pyplot as plt

from torchsummary import  summary
BATCH_SIZE = 8
```

## CIFAR10 数据集

CIFAR-10数据集5万张训练图像、1万张测试图像、10个类别、每个类别有6k个图像，图像大小32×32×3。下图列举了10个类，每一类随机展示了10张图片：

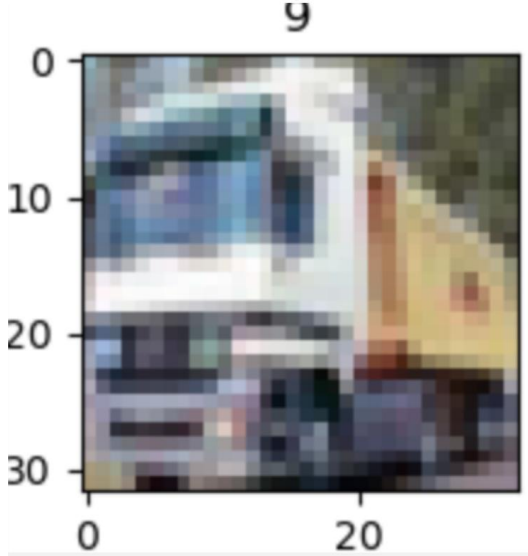


## CIFAR10 数据集

PyTorch 中的 torchvision.datasets 计算机视觉模块封装了 CIFAR10 数据集，使用方法如下:

```
# 1. 数据集基本信息
def create_dataset():
    # 加载数据集:训练集数据和测试数据
    # ToTensor: 将image（一个PIL.Image对象）转换为一个Tensor，并自动将其像素值归一化到[0.0, 1.0]
    train = CIFAR10(root='data', train=True, transform=ToTensor())
    valid = CIFAR10(root='data', train=False, transform=ToTensor())
    # 返回数据集结果
    return train, valid

if __name__ == '__main__':
    # 数据集加载
    train_dataset, valid_dataset = create_dataset()
    # 数据集类别
    print("数据集类别:", train_dataset.class_to_idx)
    # 数据集中的图像数据
    print("训练集数据集:", train_dataset.data.shape)
    print("测试集数据集:", valid_dataset.data.shape)
    # 图像展示
    plt.figure(figsize=(2, 2))
    plt.imshow(train_dataset.data[1])
    plt.title(train_dataset.targets[1])
    plt.show()
```

```
数据集类别: {'airplane': 0,
'automobile': 1, 'bird': 2, 'cat': 3,
'deer': 4, 'dog': 5, 'frog': 6,
'horse': 7, 'ship': 8, 'truck': 9}
训练集数据集: (50000, 32, 32, 3)
测试集数据集: (10000, 32, 32, 3)
```



## 搭建图像分类网络

我们要搭建的网络结构如下：

## 搭建图像分类网络

我们要搭建的网络结构如下：

1. 输入形状：32x32
2. 第一个卷积层输入 3 个 Channel，输出 6 个 Channel，Kernel Size 为：3x3
3. 第一个池化层输入 30x30，输出 15x15，Kernel Size 为：2x2，Stride 为：2
4. 第二个卷积层输入 6 个 Channel，输出 16 个 Channel，Kernel Size 为 3x3
5. 第二个池化层输入 13x13，输出 6x6，Kernel Size 为：2x2，Stride 为：2
6. 第一个全连接层输入 576 维，输出 120 维
7. 第二个全连接层输入 120 维，输出 84 维
8. 最后的输出层输入 84 维，输出 10 维

我们在每个卷积计算之后应用 relu 激活函数来给网络增加非线性因素。

## 搭建图像分类网络

构建网络代码实现如下：

```
# 模型构建
class ImageClassification(nn.Module):
    # 定义网络结构
    def __init__(self):
        super(ImageClassification, self).__init__()
        # 定义网络层：卷积层+池化层
        self.conv1 = nn.Conv2d(3, 6, stride=1, kernel_size=3)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, stride=1, kernel_size=3)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        # 全连接层
        self.linear1 = nn.Linear(576, 120)
        self.linear2 = nn.Linear(120, 84)
        self.out = nn.Linear(84, 10)
```

## 搭建图像分类网络

构建网络代码实现如下：

```
# 定义前向传播
    def forward(self, x):
        # 卷积+relu+池化
        x = torch.relu(self.conv1(x))
        x = self.pool1(x)
        # 卷积+relu+池化
        x = torch.relu(self.conv2(x))
        x = self.pool2(x)
        # 将特征图做成以为向量的形式：相当于特征向量
        x = x.reshape(x.size(0), -1)
        # 全连接层
        x = torch.relu(self.linear1(x))
        x = torch.relu(self.linear2(x))
        # 返回输出结果
        return self.out(x)
```

## 搭建图像分类网络

模型结构为：

```
if __name__ == '__main__':
    # 模型实例化
    model = ImageClassification()
    summary(model,input_size=(3,32,32),batch_size=1)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1             [1, 6, 30, 30]             168
         MaxPool2d-2             [1, 6, 15, 15]               0
            Conv2d-3            [1, 16, 13, 13]             880
         MaxPool2d-4              [1, 16, 6, 6]               0
            Linear-5                   [1, 120]          69,240
            Linear-6                    [1, 84]          10,164
            Linear-7                    [1, 10]             850
================================================================
Total params: 81,302
Trainable params: 81,302
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.31
Estimated Total Size (MB): 0.40
----------------------------------------------------------------
```

## 编写训练函数

在训练时，使用多分类交叉熵损失函数，Adam 优化器. 具体实现代码如下：

```
def train(model,train_dataset):
    criterion = nn.CrossEntropyLoss() # 构建损失函数
    optimizer = optim.Adam(model.parameters(), lr=1e-3) # 构建优化方法
    epoch = 100  # 训练轮数
    for epoch_idx in range(epoch):
        # 构建数据加载器
        dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
        sam_num = 0  # 样本数量
        total_loss = 0.0  # 损失总和
        start = time.time() # 开始时间
        # 遍历数据进行网络训练
        for x, y in dataloader:
            output = model(x)
            loss = criterion(output, y) # 计算损失
            optimizer.zero_grad() # 梯度清零
            loss.backward() # 反向传播
            optimizer.step() # 参数更新

            # 计算每次训练模型的总损失值 loss是每批样本平均损失值
            total_loss += loss.item()*len(y)  # 统计损失和
            sam_num += len(y)
        print('epoch:%2s loss:%.5f time:%.2fs' %(epoch_idx + 1,total_loss / sam_num,time.time() - start))
    # 模型保存
    torch.save(model.state_dict(), 'model/image_classification.pth')
```

## 编写训练函数

调用训练方法进行模型训练如下：

```
if __name__ == '__main__':
    # 数据集加载
    train_dataset, valid_dataset = create_dataset()
    # 模型实例化
    model = ImageClassification()
    # 模型训练
    train(model,train_dataset)
```

输出结果:

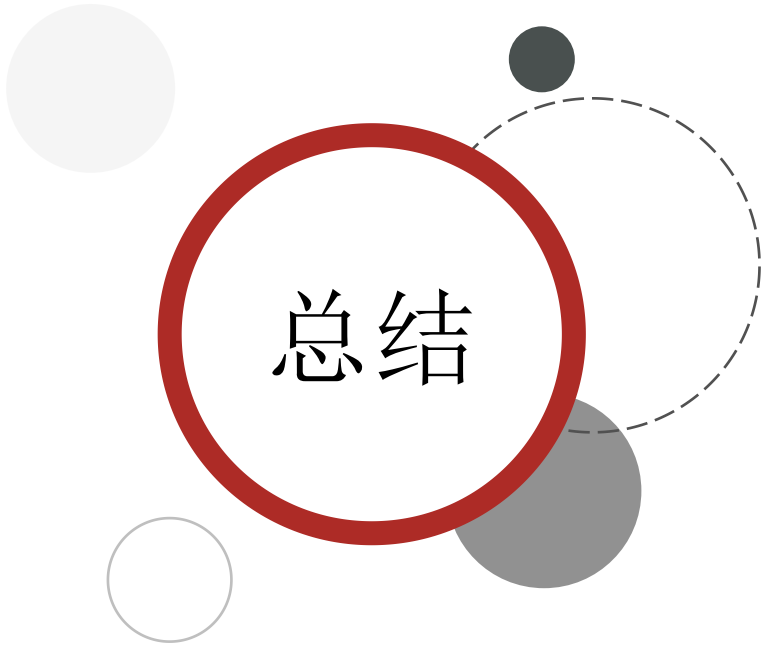
```
epoch: 1 loss:1.59926 acc:0.41 time:28.97s
epoch: 2 loss:1.32861 acc:0.52 time:29.98s
epoch: 3 loss:1.22957 acc:0.56 time:29.44s
epoch: 4 loss:1.15541 acc:0.59 time:30.45s
epoch: 5 loss:1.09832 acc:0.61 time:29.69s
epoch: 6 loss:1.05423 acc:0.63 time:29.70s
epoch: 7 loss:1.01612 acc:0.64 time:30.09s
epoch: 8 loss:0.98208 acc:0.65 time:30.45s
epoch: 9 loss:0.95528 acc:0.66 time:29.26s
epoch:10 loss:0.92802 acc:0.67 time:29.05s
```

## 编写预测函数

加载训练好的模型，对测试集中的 1 万条样本进行预测，查看模型在测试集上的准确率.

```
def test(valid_dataset):
    # 构建数据加载器
    dataloader = DataLoader(valid_dataset, batch_size=BATCH_SIZE,
shuffle=False)
    # 加载模型并加载训练好的权重
    model = ImageClassification()
    model.load_state_dict(torch.load('model/image_classification.pth'))
    model.eval()
    # 计算精度
    total_correct = 0
    total_samples = 0
    # 遍历每个batch的数据，获取预测结果，计算精度
    for x, y in dataloader:
        output = model(x)
        total_correct += (torch.argmax(output, dim=-1) == y).sum()
        total_samples += len(y)
    # 打印精度
    print('Acc: %.2f' % (total_correct / total_samples))
```

输出结果:
Acc: 0.57




1. 掌握模型构建流程

加载数据集

模型构建

模型训练

模型测试


从程序的运行结果来看，网络模型在测试集上的准确率并不高。我们可以从以下几个方面来进行优化：

1. 增加卷积核输出通道数
2. 增加全连接层的参数量
3. 调整学习率
4. 调整优化方法
5. 修改激活函数
6. 等等...


传智教育旗下高端IT教育品牌