深度卷积神经网络

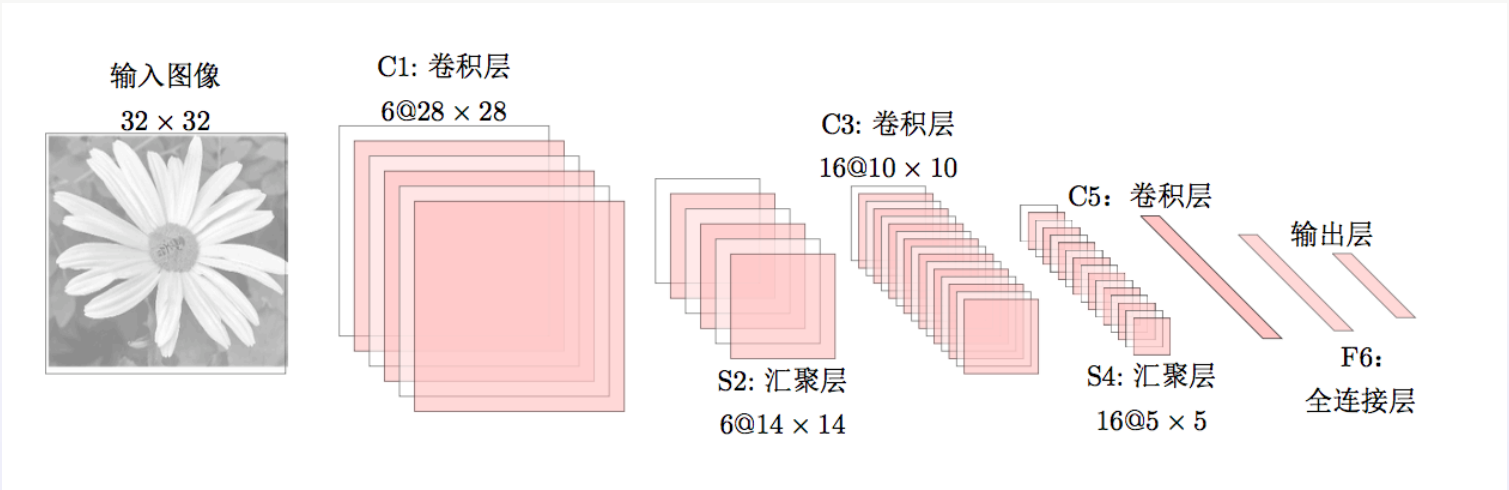
陈睿

国防科技大学

2019 年 4 月 13 日

卷积神经网络

- CNN 概观：



- 一个典型的 CNN 是由卷积层、池化层、全连接层交叉堆叠而成。

一维卷积

- **卷积 (Convolution)**，是分析数学中一种重要的运算。在信号处理或图像处理中，经常使用一维或二维卷积。
- **一维卷积**在信号处理中，用于计算信号的延迟累积。
- 假设一个信号发生器每个时刻 t 产生一个信号 x_t，其信息的衰减率为 w_k，即在 $k-1$ 个时间步长后，信息为原来的 w_k 倍。通常把 $w_1, w_2, \cdots$ 称为滤波器 (Filter) 或卷积核 (Convolution Kernel)。假设滤波器长度为 m，它和一个信号序列 $x_1, x_2, ...$ 的卷积为

$$y_t = \sum_{k=1}^{m} w_k \cdot x_{t-k+1} \tag{1}$$

- 信号序列 x 和滤波器 w 的卷积定义为

$$y = w \otimes x \tag{2}$$

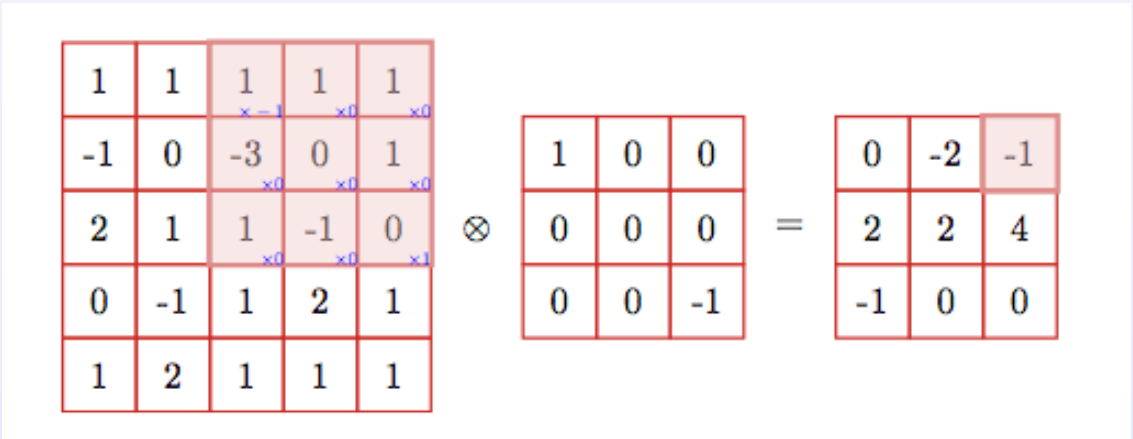
其中 $\otimes$ 表示卷积运算。

二维卷积

- **二维卷积**则经常用在图像处理中。给定一个图像 $X \in \mathbb{R}^{M \times N}$，和滤波器 $w \in \mathbb{R}^{m \times n}$，一般 $m \ll M$，$n \ll N$，其卷积为

$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} w_{uv} \cdot x_{i-u+1, j-v+1} \tag{3}$$

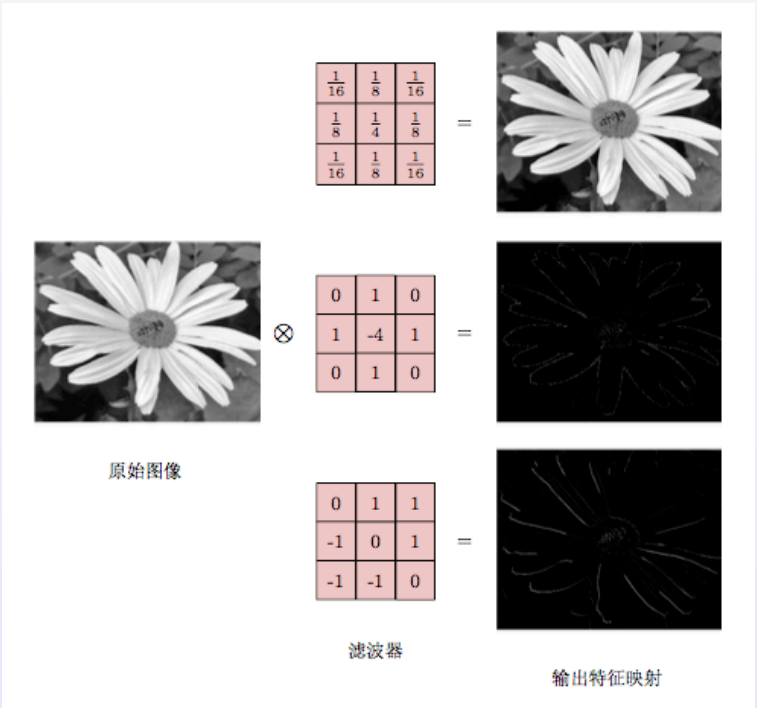
- 二维卷积操作如图所示：



- 操作后特征图大小为 $(M - m + 2p)/s + 1) \times (N - n + 2p)/s + 1)$，p 为 0 填充大小，s 为步长。

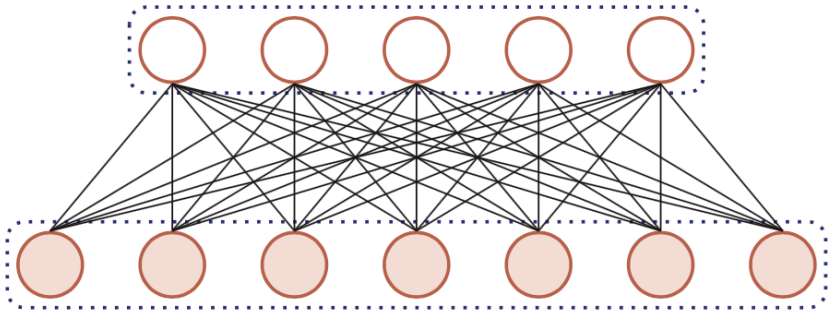
特征提取

- 在图像处理中，卷积经常作为**特征提取**的有效方法。一幅图像在经过卷积操作后得到的结果称为特征映射 (Feature Map)。
- 图像处理中几种常用的**滤波器**如图所示，高斯滤波器——平滑去噪；其余两个——提取边缘特征：

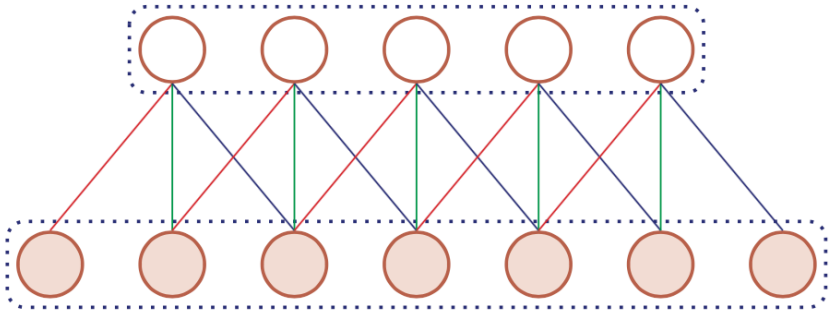


卷积层

- **卷积层**有两个重要特点：局部连接，权重共享。
- 全连接层和卷积层的对比：



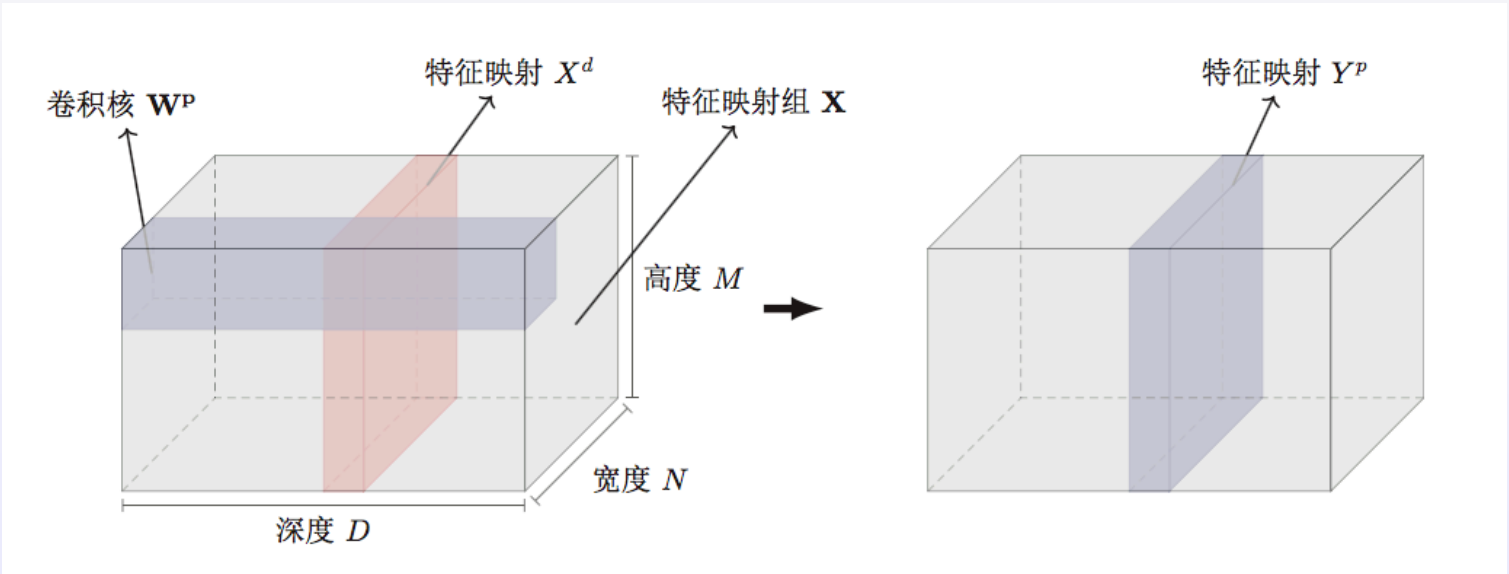
(a) 全连接层



(b) 卷积层

卷积层

- 卷积层的作用是提取一个局部区域的特征，而不同的卷积核相当于不同的特征提取器。
- 卷积层的三维结构表示：



卷积层

- 卷积层的结构如下：

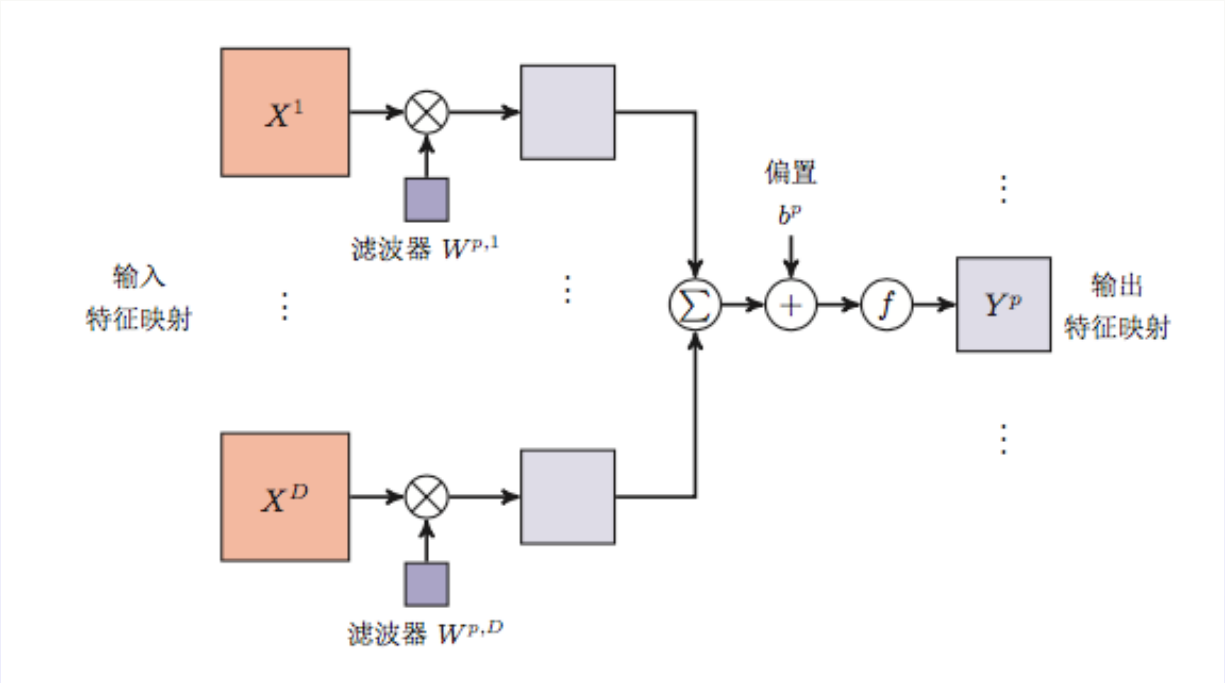
  输入特征映射组：$X \in \mathbb{R}^{M \times N \times D}$ 为三维张量 (tensor)，其中每个切片 (slice) 矩阵 $X^d \in \mathbb{R}^{M \times N}$ 为一个输入特征映射，$1 \leq d \leq D$;

  输出特征映射组: $Y \in \mathbb{R}^{M' \times N' \times P}$ 为三维张量，其中每个切片矩阵 $Y^p \in \mathbb{R}^{M' \times N'}$ 为一个输出特征映射，$1 \leq p \leq P$;

  卷积核: $W \in \mathbb{R}^{m \times n \times D \times P}$ 为四维张量，其中每个切片矩阵 $W^{p,q} \in \mathbb{R}^{m \times n}$ 为一个两维卷积核，$1 \leq d \leq D$，$1 \leq p \leq P$;

卷积层

- 卷积层的计算示例表示如下：



为了计算输出特征映射 Y^p，用卷积核 $W^{p,1}, W^{p,2}, ..., W^{p,D}$ 分别对输入特征映射 $X^1, X^2, \cdots, X^D$ 进行卷积，然后将卷积结果相加，并加上一个标量偏置 b 得到卷积层的净输入 Z^p，再经过非线性激活函数后得到输出特征映射 Y^p。

卷积层

- 可形式化为：

$$Z^P = W^P \otimes X + b^P = \sum_{d=1}^{D} W^{p,d} \otimes X^d + b^q, \tag{4}$$

$$Y^p = f(Z^P). \tag{5}$$

其中 $W^p \in \mathbb{R}^{m \times n \times D}$ 为三维卷积核，$f(\cdot)$ 为非线性激活函数，一般用 ReLU 函数。将上述过程重复 P 次，得到 P 个输出特征映射 $Y^1, Y^2, \bullet\bullet\bullet, Y^P$。

- 在输入为 $X \in \mathbb{R}^{M \times N \times D}$，输出为 $Y \in \mathbb{R}^{M' \times N' \times P}$ 的卷积层中，每一个输入特征映射都需要 D 个滤波器以及一个偏置。假设每个滤波器的大小为 $m \times n$，那么共需要 $P \times D \times (m \times n) + P$ 个参数。

池化层

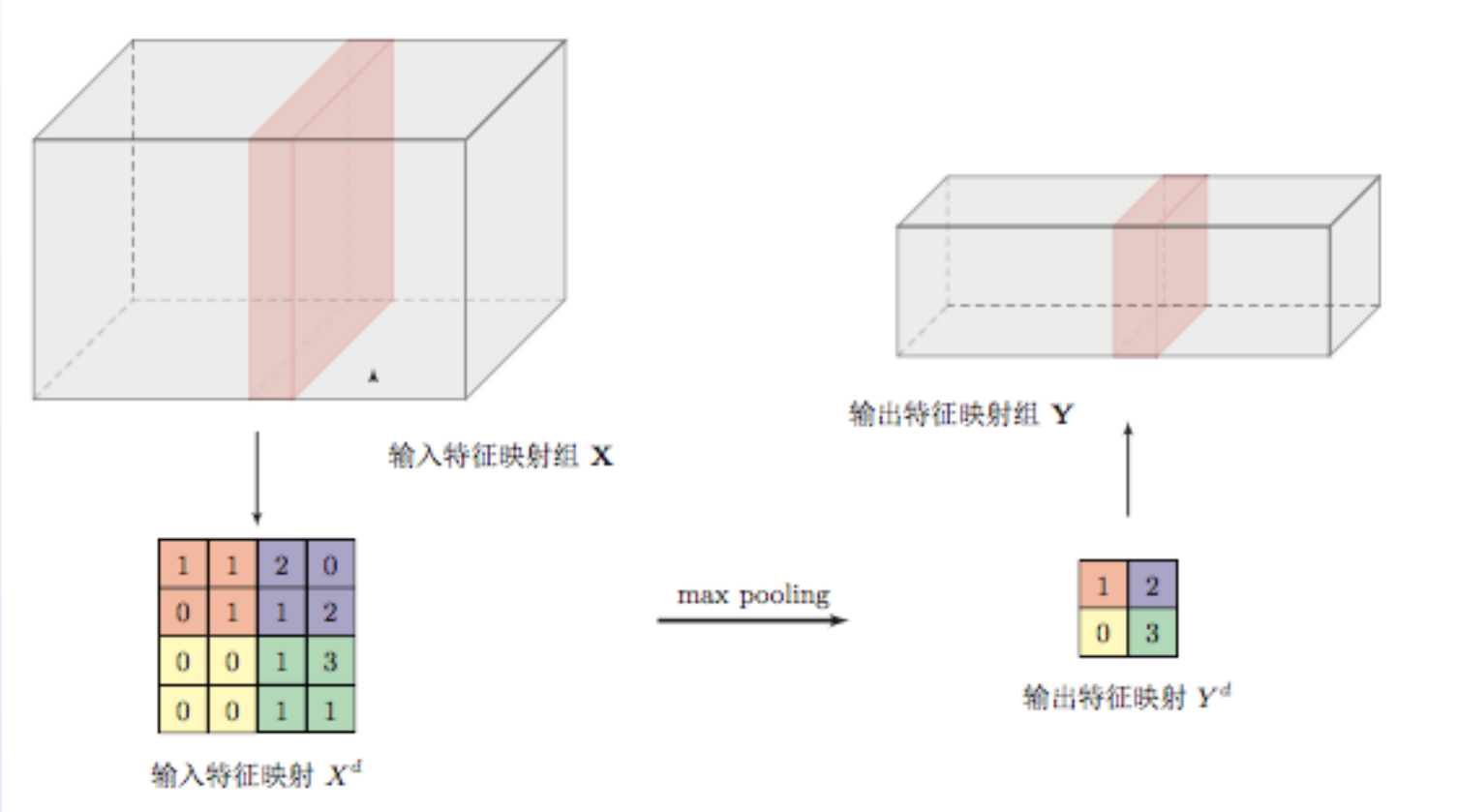
- **池化层** (Pooling Layer) 也叫子采样层 (Subsampling Layer)，其作用是进行特征选择，降低特征数量，从而减少参数数量，避免过拟合。
- 常用的池化函数有两种:

  最大池化 (Maximum Pooling): 一般是取一个区域内所有神经元的最大值。

  平均池化 (Mean Pooling): 一般是取区域内所有神经元的平均值。
- 对每一个输入特征映射 X^d 的 $M' \times N'$ 个区域进行子采样，得到池化层的输出特征映射 $Y^d = Y^d_{m,n}, 1 \leq m \leq M', 1 \leq n \leq N'$。

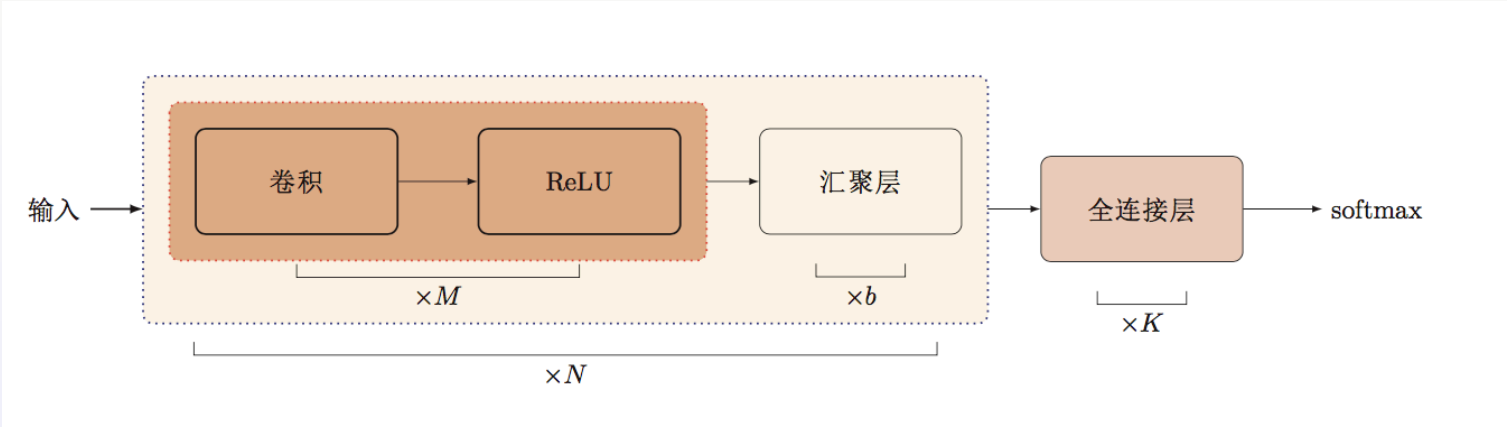
池化层

- 池化层池化过程表示：



典型 CNN 结构

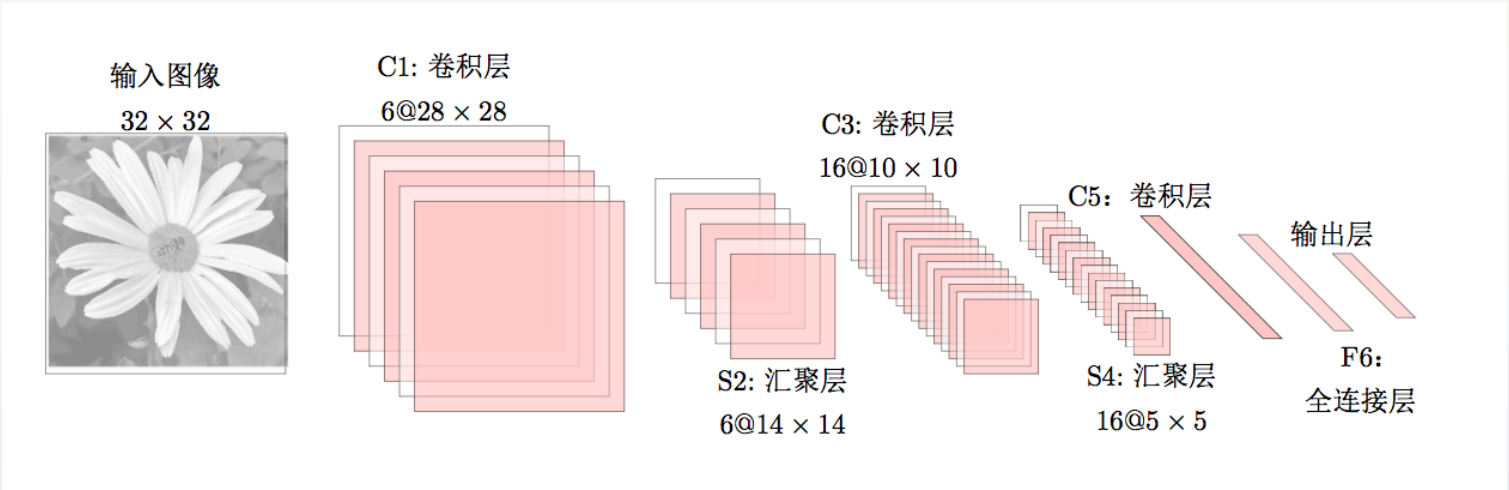
- **典型的网络结构：**



- 一个典型的卷积网络是由卷积层、池化层、全连接层交叉堆叠而成。其中一个卷积块为连续 M 个卷积层和 b 个池化层 (M 通常设置为 2 ~ 5，b 为 0 或 1)。一个卷积网络中可以堆叠 N 个连续的卷积块，然后再接 K 个全连接层 (N 的取值区间为 1 ~ 100 或者更大;K 一般为 0 ~ 2)。目前，整个网络结构趋向于使用更小的卷积核 (比如 1×1 和 3×3) 以及更深的结构 (比如层数大于 50)。

LeNet-5

- LeNet-5 网络结构如下：



- 不计输入层，LeNet-5 共有 7 层，每一层的结构为:
  1. 输入层: 输入图像大小为 $32 \times 32 = 1024$。
  2. C1 层是卷积层，含 6 个 5×5 的滤波器，得到 6 组大小为 $28 \times 28 = 784$ 的特征映射。因此，C1 层的神经元数量为 $6 \times 784 = 4704$，可训练参数数量为 $6 \times 25 + 6 = 156$，连接数为 $156 \times 784 = 122{,}304$。

LeNet-5

- 结构如下：
  3. S2 层为池化层，采样窗口为 2×2 ，使用 Mean Pooling。神经元个数为 $6 \times 14 \times 14 = 1,176$，可训练参数数量为 $6 \times (1 + 1) = 12$，连接数为 $6 \times 196 \times (4 + 1) = 5880$。
  4. C3 层为卷积层。共使用 60 个 5×5 的滤波器，得到 16 组大小为 $10 \times 16 \times 100 = 1{,}600$，可训练参数数量为 $(60 \times 25) + 16 = 1{,}516$，连接数为 $100 \times 1{,}516 = 151{,}600$。
  5.S4 层是池化层，采样窗口为 2×2，得到 16 个 5×5 大小的特征映射，可训练参数数量为 $16 \times 2 = 32$，连接数为 $16 \times 25 \times (4 + 1) = 2000$。

LeNet-5

- 结构如下：
  6. C5 层是一个卷积层，使用 $120 \times 16 = 1,920$ 个 5×5 的滤波器，得到 120 组大小为 1×1 的特征映射。C5 层的神经元数量为 120，可训练参数数量为 $1{,}920 \times 25 + 120 = 48{,}120$，连接数为 $120 \times (16 \times 25 + 1) = 48{,}120$。
  7. F6 层是一个全连接层，有 84 个神经元，可训练参数数量为 $84 \times (120+1) = 10,164$。连接数和可训练参数个数相同，为 10,164。
  8. 输出层: 输出层由 10 个欧氏径向基函数 (RBF) 函数组成。

Keras Layers

- 二维卷积层:
keras.layers.convolutional.Convolution2D(nb_filter, nb_row, nb_col, init='glorot_uniform', activation='linear', weights=None, border_mode='valid', strides=(1, 1), dim_ordering='th', W_regularizer=None, b_regularizer=None, activity_regularizer=None, W_constraint=None, b_constraint=None, bias=True)
主要参数：nb_filter: 卷积核的数目；nb_row: 卷积核的行数；nb_col: 卷积核的列数；activation: 激活函数; strides: 步长；

LeNet 的搭建

- 导入模块：

```
#coding=utf-8
from keras.models import Sequential
from keras.layers import Dense,Flatten,Dropout
from keras.layers.convolutional import Conv2D,MaxPooling2D
from keras.utils.np_utils import to_categorical
import cPickle
import gzip
import numpy as np
```

LeNet 的搭建

- 载入数据，划分训练、验证、测试集：

```
data = gzip.open(r'/media/wmy/document/BigData/kaggle/Digit Recognizer/mnist.pkl.gz')
train_set,valid_set,test_set = cPickle.load(data)
train_x = train_set[0].reshape((-1,28,28,1))
train_y = to_categorical(train_set[1])
valid_x = valid_set[0].reshape((-1,28,28,1))
valid_y = to_categorical(valid_set[1])
test_x = test_set[0].reshape((-1,28,28,1))
test_y = to_categorical(test_set[1])
```

LeNet 的搭建

- 搭建模型：

```
model = Sequential()
model.add(Conv2D(32,(5,5),strides=(1,1),input_shape=(28,28,1)
,padding='valid',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(5,5),strides=(1,1),padding='valid',
activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(100,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='sgd',loss='categorical_crossentropy',
,metrics=['accuracy'])
model.summary()
```

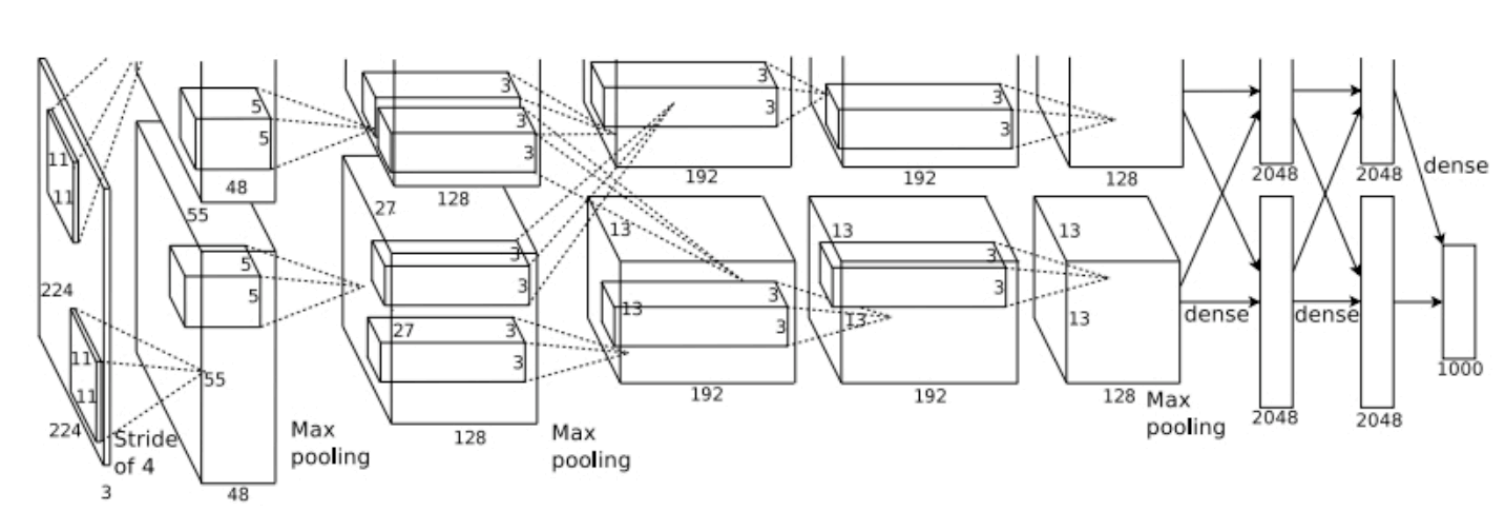
LeNet-5 的搭建

- 拟合、评估模型：

```
model.fit(train_x,train_y,validation_data=(valid_x,valid_y),
batch_size=20,epochs=20,verbose=2)
print model.evaluate(test_x,test_y,batch_size=20,verbose=2)
```

AlexNet

- AlexNet 网络结构如下：



- 使用 GPU 进行并行训练，ReLU 作为非线性激活函数，Dropout 防止过拟合，以及数据增强来提高模型准确率等。
- AlexNet 的结构，包括 5 个卷积层、3 个全连接层和 1 个 softmax 层。因为网络规模超出了当时的单个 GPU 的内存限制，AlexNet 将网络拆为两半，分别放在两个 GPU 上，GPU 间只在某些层 (比如第 3 层) 进行通讯。

AlexNet

- AlexNet 的具体结构如下：

1. 输入层，$224 \times 224 \times 3$ 的图像。

2. 一个卷积层，使用两个 $11 \times 11 \times 3 \times 48$ 的卷积核，步长 $s = 4$，零填充 $p = 3$，得到两个 $55 \times 55 \times 48$ 的特征映射组。

3. 第一个池化层，使用大小为 3×3 的最大池化操作，步长 $s = 2$，得到两个 $27 \times 27 \times 48$ 的特征映射组。

4. 第二个卷积层，使用两个 $5 \times 5 \times 48 \times 128$ 的卷积核，步长 $s = 1$，零填充 $p = 2$，得到两个 $27 \times 27 \times 128$ 的特征映射组。

5. 第二个池化层，使用大小为 3×3 的最大池化操作，步长 $s = 2$，得到两个 $13 \times 13 \times 128$ 的特征映射组。

6. 第三个卷积层为两个路径的融合，使用一个 $3 \times 3 \times 256 \times 384$ 的卷积核，步长 $s = 1$，零填充 $p = 1$，得到两个 $13 \times 13 \times 192$ 的特征映射组。

AlexNet

- 结构如下：

7. 第四个卷积层，使用两个 $3 \times 3 \times 192 \times 192$ 的卷积核，步长 $s = 1$，零填充 $p = 1$，得到两个 $13 \times 13 \times 192$ 的特征映射组。

8. 第五个卷积层，使用两个 $3 \times 3 \times 192 \times 128$ 的卷积核，步长 $s = 1$，零填充 $p = 1$，得到两个 $13 \times 13 \times 128$ 的特征映射组。

9. 池化层，使用大小为 3×3 的最大池化操作，步长 $s = 2$，得到两个 $6 \times 6 \times 128$ 的特征映射组。

10. 三个全连接层，神经元数量分别为 4096，4096 和 1000。

其他经典 CNN 网络

- Inception 网络是由多个 inception 模块和少量的池化层堆叠而成。Inception 模块同时使用 1×1、3×3、5×5 等不同大小的卷积核，并将得到的特征映射在深度上拼接 (堆叠) 起来作为输出特征映射。
- GoogLeNet 由 9 个 Inception v1 模块和 5 个汇聚层以及其它一些卷积层和全连接层构成，总共为 22 层网络。
- 残差网络是通过给非线性的卷积层增加直连边的方式来提高信息的传播效率。

AlexNet 的搭建

- 搭建模型：

```
model = Sequential()
model.add(Conv2D(96,(11,11),strides=(4,4),input_shape=(227,227,3)
,padding='valid',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(256,(5,5),strides=(1,1),padding='same'
,activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same'
,activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same'
,activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(256,(3,3),strides=(1,1),padding='same'
,activation='relu',kernel_initializer='uniform'))
```

AlexNet 的搭建

- 搭建模型：

```
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='sgd',
metrics=['accuracy'])
model.summary()
```

参考书目

- 《神经网络与深度学习》(邱锡鹏)
- keras 实现常用深度学习模型 LeNet，AlexNet，ZFNet，VGGNet，GoogleNet，Resnet (https://blog.csdn.net/wang1127248268/article/details/77258055)