人工智能实践：Tensorflow笔记

曹健

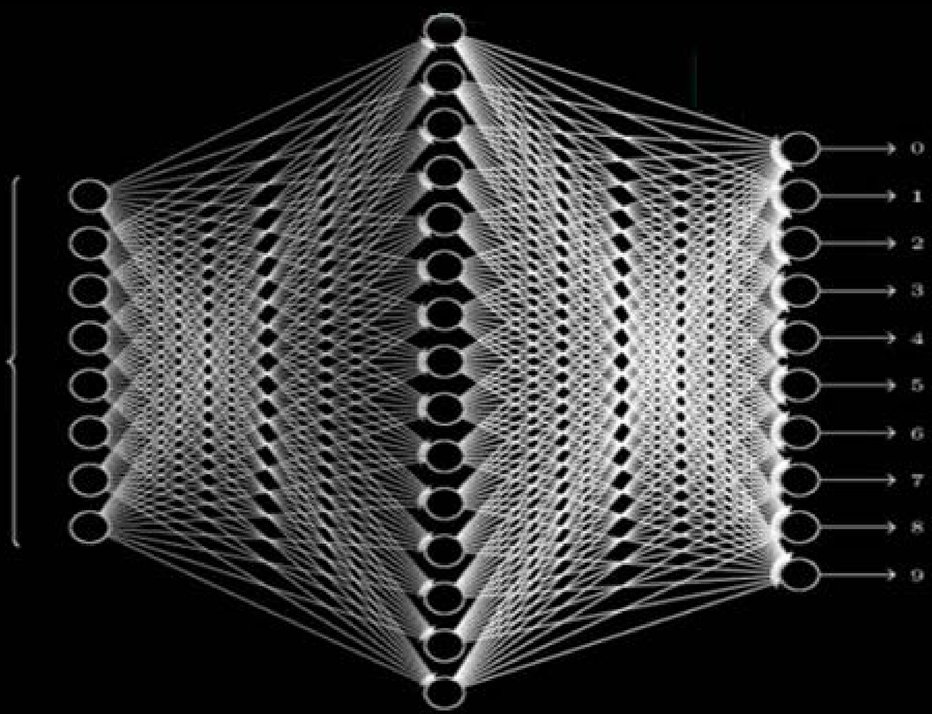
北京大学

软件与微电子学院

✓ **全连接NN：每个神经元与前后相邻层的每一个神经元都有连接关系，输入是特征，输出为预测的结果。**

参数个数：$\sum_{各层}(\underbrace{前层 \times 后层}_{w} + \underbrace{后层}_{b})$

784个像素点灰度值



10个类别各自概率值

第一层参数:
784*128个w + 128个b

第二层参数:
128*10个w + 10个b

共101770个

实际项目中的图片多是高分辨率彩色图

=

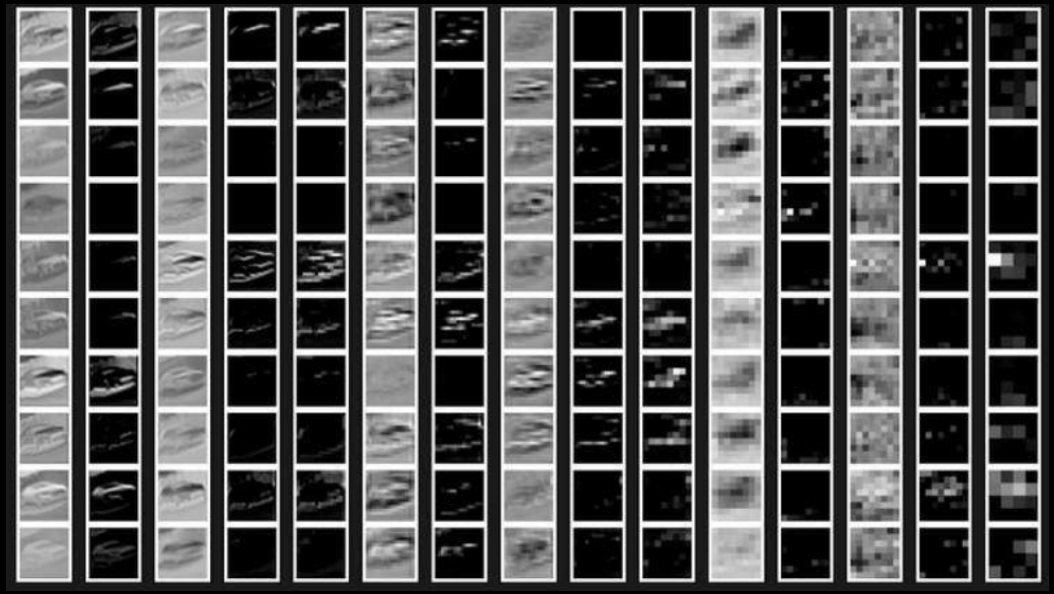
灰度图单通道

彩色图红绿蓝三通道

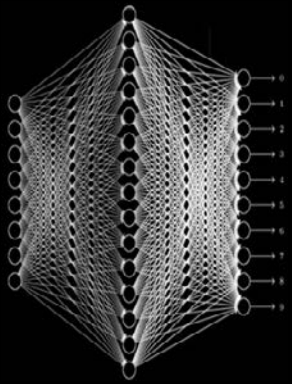
待优化的参数过多容易导致模型过拟合

实际应用时会先对原始图像进行特征提取
再把提取到的特征送给全连接网络

原始图片



若干层特征提取



全连接网络

卷积
Convolutional

- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。

输入特征是单通道

卷积
Convolutional

- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。

输入特征图的深度（channel数），决定了当前层卷积核的深度；

当前层卷积核的个数，决定了当前层输出特征图的深度。

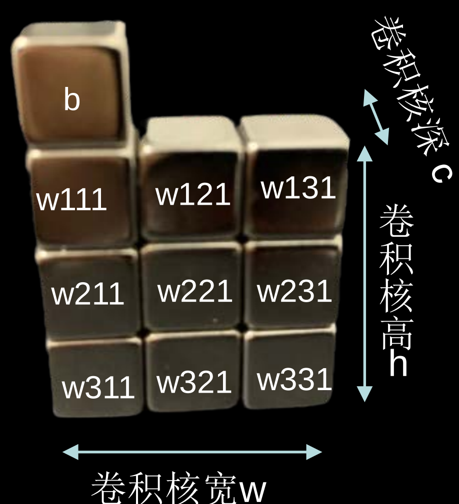
输入特征是三通道

卷积
Convolutional

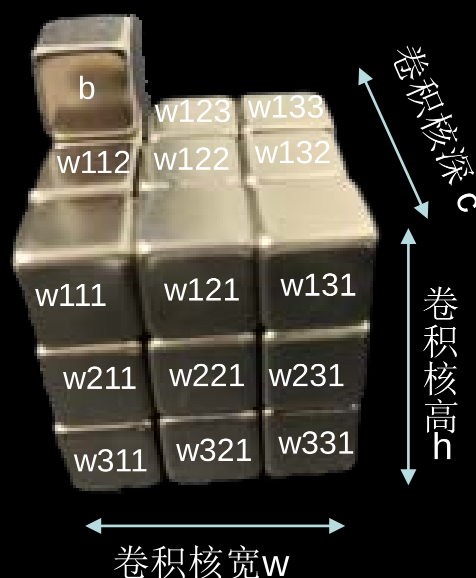
- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。

输入特征图的深度（channel数）决定了当前层卷积核的深度；

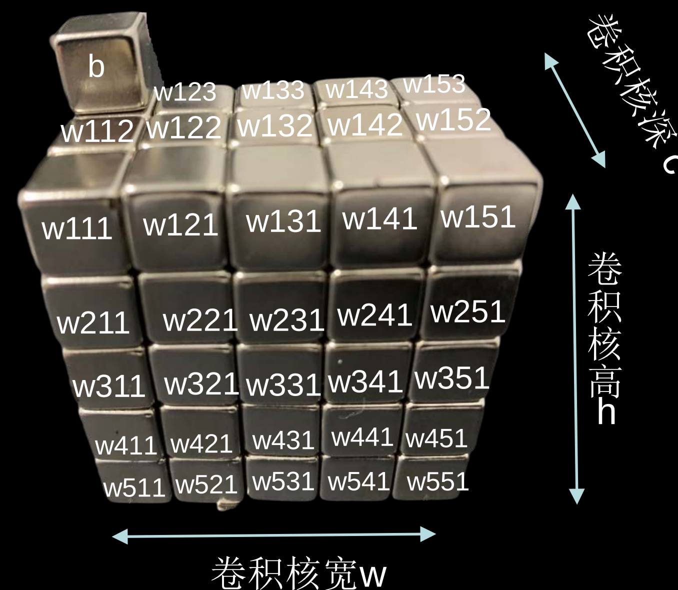
当前层卷积核的个数，决定了当前层输出特征图的深度。



3*3*1卷积核
每个核3*3+1=10个参数



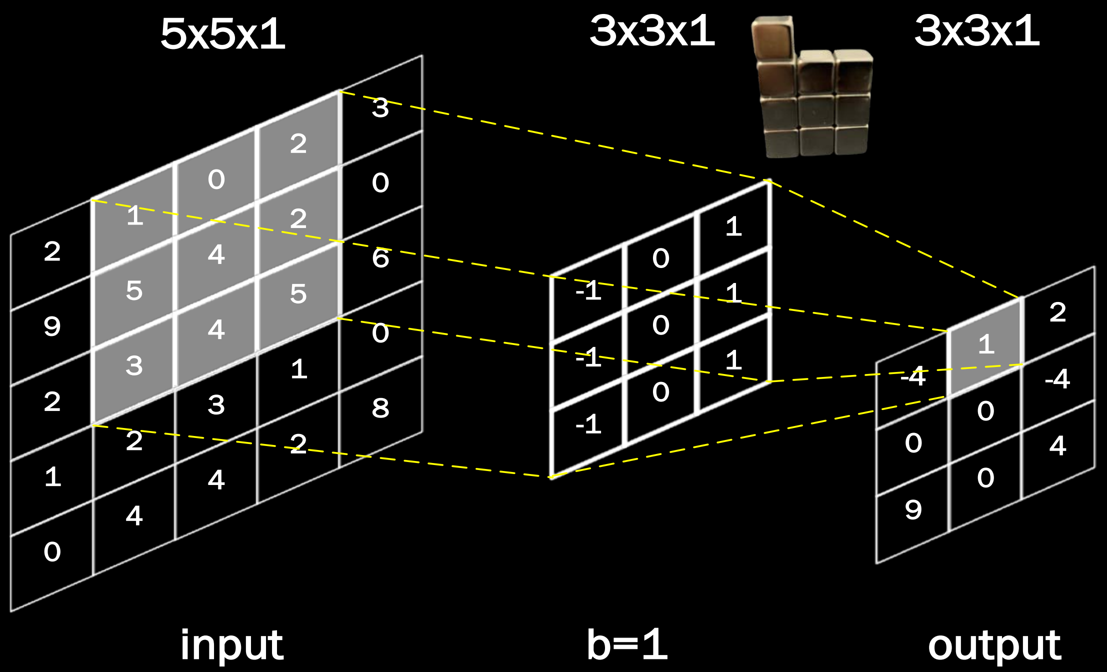
3*3*3卷积核
每个核3*3*3+1=28个参数



5*5*3卷积核
每个核5*5*3+1=76个参数

卷积
Convolutional

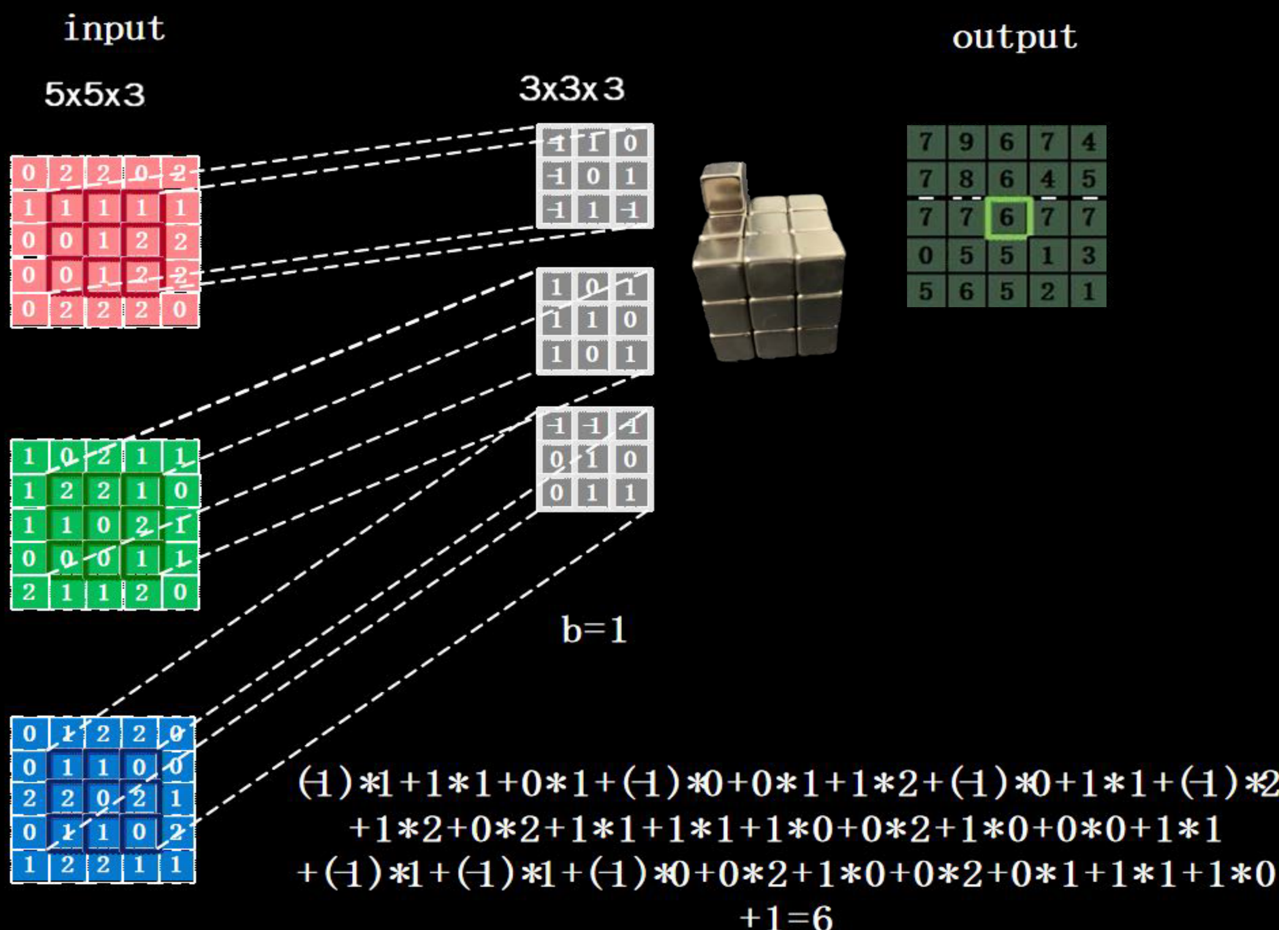
- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。



(-1)*1+0*0+1*2
+(-1)*5+0*4+1*2
+(-1)*3+0*4+1*5
+1=1

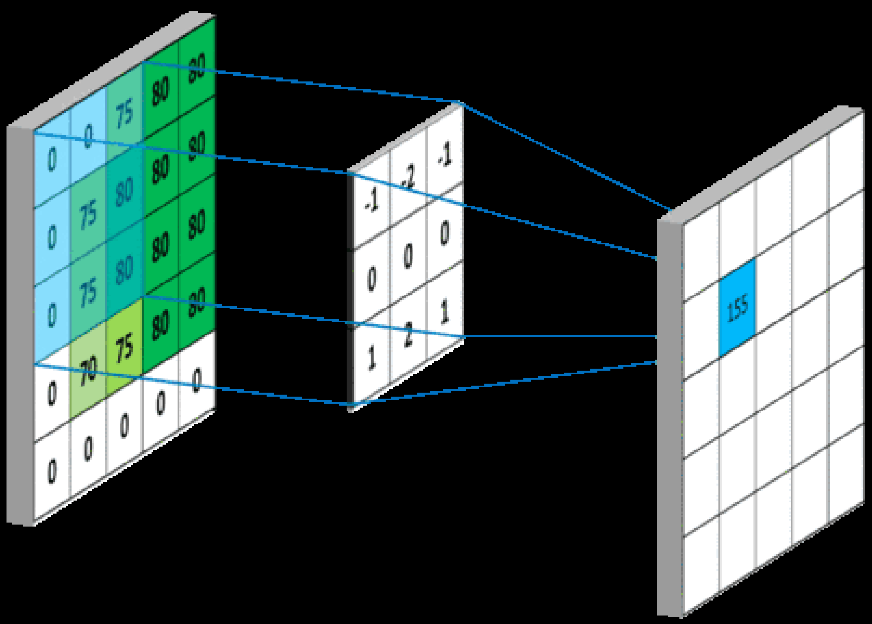
卷积
Convolutional

- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。



卷积
Convolutional

- 卷积计算可认为是一种有效提取图像特征的方法
- 一般会用一个正方形的卷积核，按指定步长，在输入特征图上滑动，遍历输入特征图中的每个像素点。每一个步长，卷积核会与输入特征图出现重合区域，重合区域对应元素相乘、求和再加上偏置项得到输出特征的一个像素点。

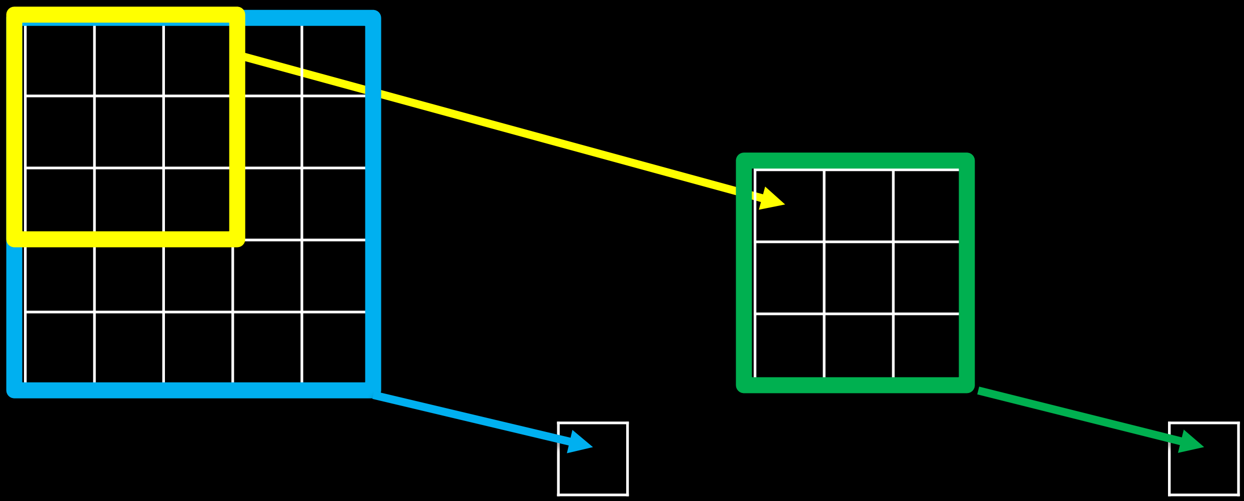


动图来源：https://mlnotebook.github.io/post/CNN1/

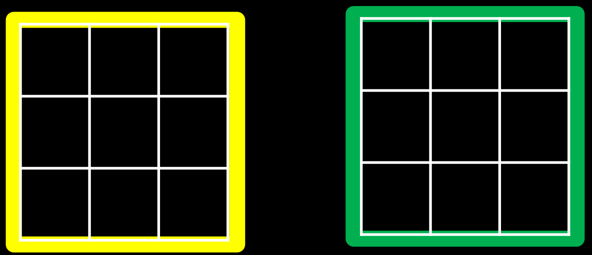
本讲目标：用CNN实现离散数据的分类（以图像分类为例）

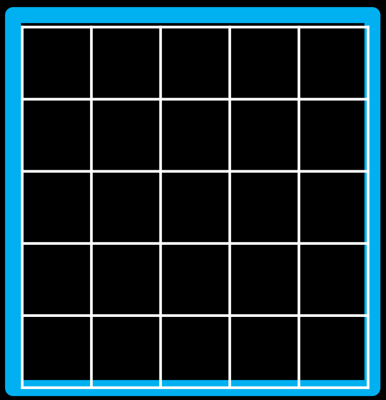
卷积计算过程
感受野
全零填充（Padding）
TF描述卷积计算层
批标准化（Batch Normalization， BN）
池化（Pooling）
舍弃（Dropout）
卷积神经网络
cifar10数据集
卷积神经网络搭建示例
实现LeNet、AlexNet、VGGNet、InceptionNet、ResNet五个经典卷积网络

✓ **感受野（Receptive Field）：卷积神经网络各输出特征图中的每个像素点，在原始输入图片上映射区域的大小。**



设输入特征图宽、高为x，卷积计算步长为1





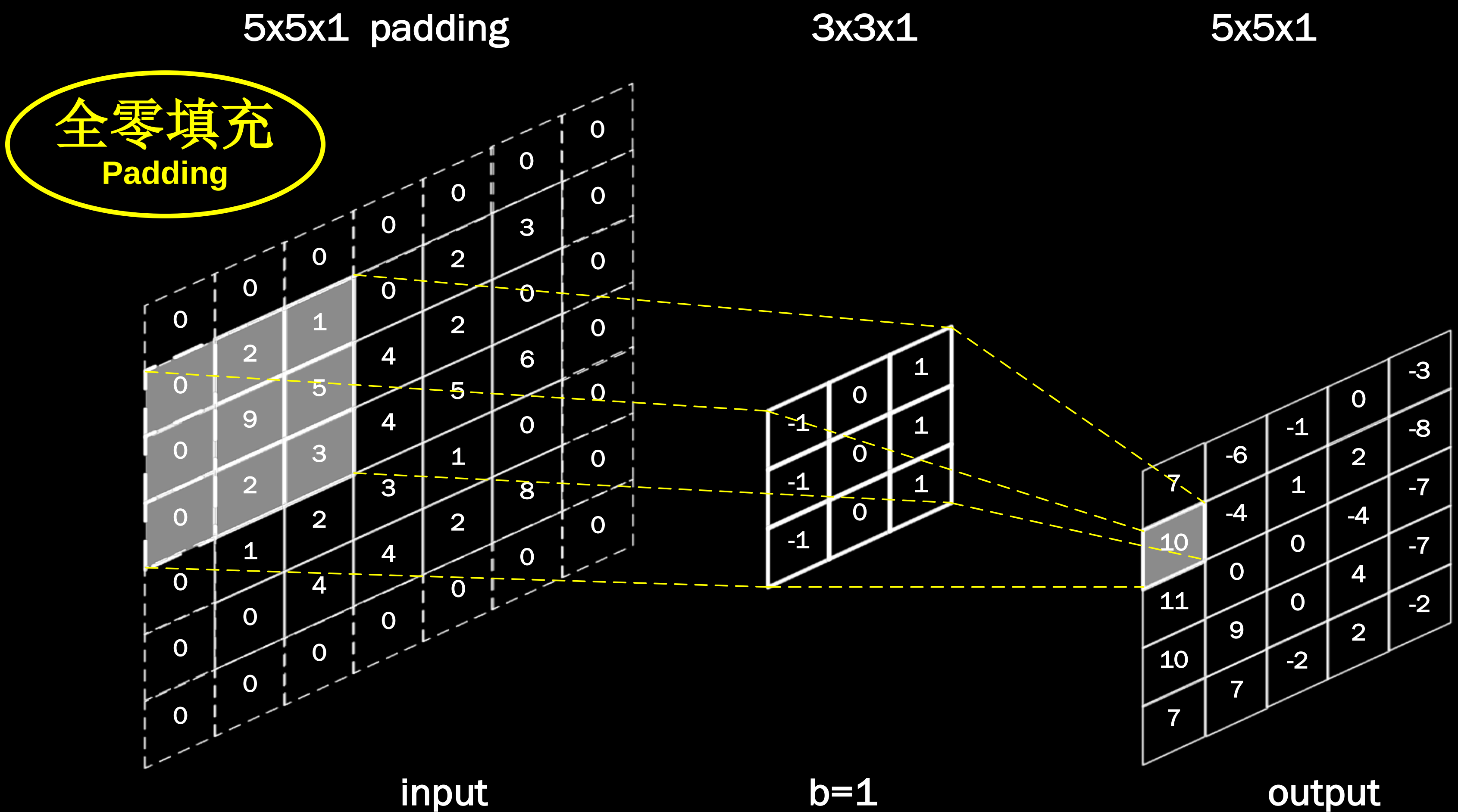
参数量：9+9=18

计算量：$18x^2 - 108x + 180$

参数量：25

计算量：$25x^2 - 200x + 400$

当x>10 时，两层3*3卷积核 优于 一层5*5卷积核



$$
\begin{aligned}
&(-1)*0+0*2+1*1\\
&+(-1)*0+0*9+1*5\\
&+(-1)*0+0*2+1*3\\
&+1=10
\end{aligned}
$$

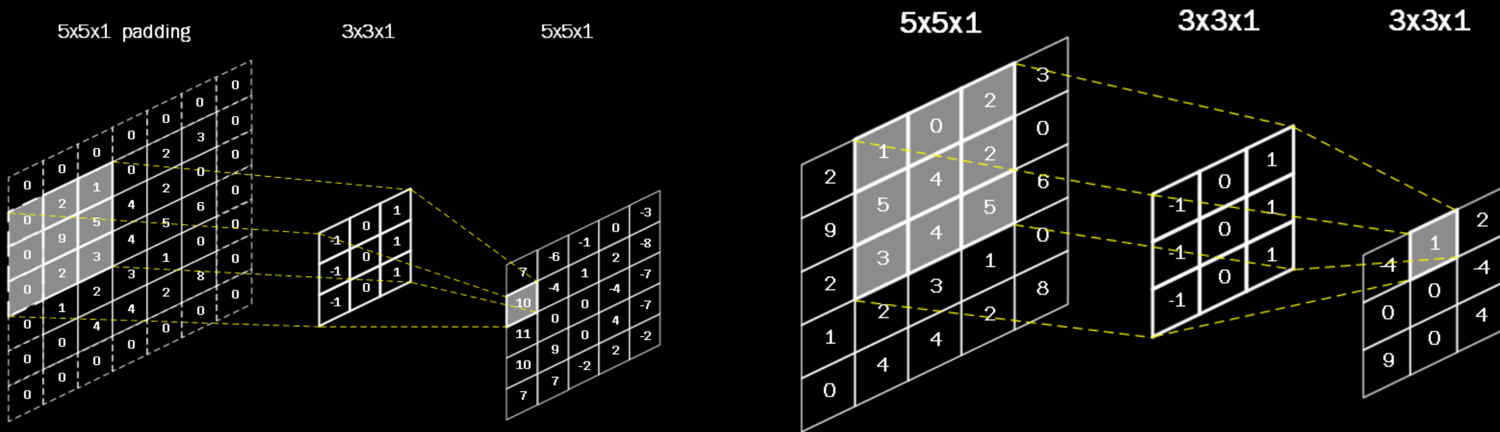
输出图片边长= 输入图片边长 / 步长

此图： **5 / 1 = 5**

全零填充
Padding

$$\textit{padding}\begin{cases}\textit{SAME}\text{（全0填充）} & \dfrac{\text{入长}}{\text{步长}}\text{（向上取整）}\\[2ex] \textit{VALID}\text{（不全0填充）} & \dfrac{\text{入长}-\text{核长}+1}{\text{步长}}\text{（向上取整）}\end{cases}$$

TF描述全零填充
用参数padding = ‘SAME’ 或 padding = ‘VALID’表示



输出宽或高: $\dfrac{5}{1} = 5$ $\dfrac{5-3+1}{1} = 3$

SAME： 5x5x1 ⇨ 5x5x1 VALID： 5x5x1 ⇨ 3x3x1

✓ TF描述卷积层

tf.keras.layers.Conv2D (
filters = 卷积核个数,
kernel_size = 卷积核尺寸, #正方形写核长整数，或（核高h，核宽w）
strides = **滑动**步长, #横纵向相同写步长整数，或(纵向步长h，横向步长w)，默认1
padding = “same” or “valid”, #使用全零填充是“same”，不使用是“valid”（默认）
activation = “ relu ” or “ sigmoid ” or “ tanh ” or “ softmax”等 , **#如有BN此处不写**
input_shape = (高, 宽 , 通道数)   **#输入特征图维度，可省略**
)

```
model = tf.keras.models.Sequential([
    Conv2D(6, 5, padding='valid', activation='sigmoid'),
    MaxPool2D(2, 2),
    Conv2D(6, (5, 5), padding='valid', activation='sigmoid'),
    MaxPool2D(2, (2, 2)),
    Conv2D(filters=6, kernel_size=(5, 5),padding='valid', activation='sigmoid'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(10, activation='softmax')
])
```

批标准化
BN

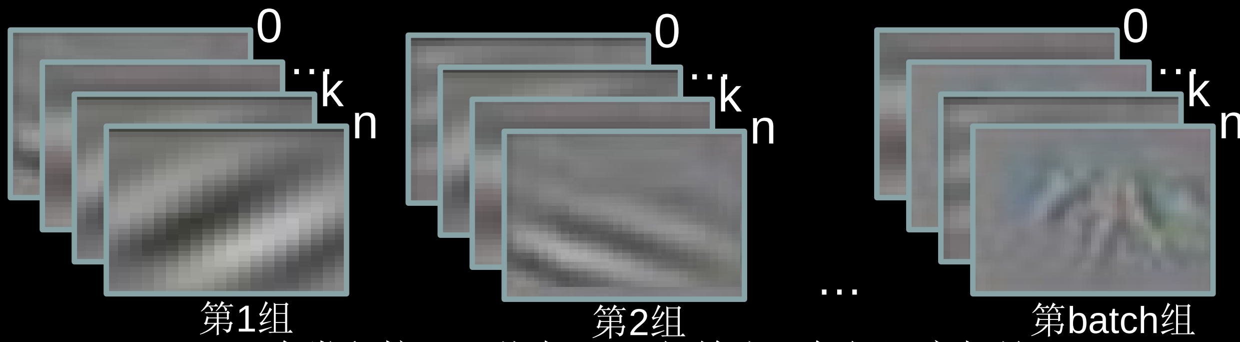
批标准化（Batch Normalization， BN）

标准化：使数据符合0均值，1为标准差的分布。

批标准化：对一小批数据（batch），做标准化处理 。

批标准化后，第 k个卷积核的输出特征图（feature map）中第 i 个像素点

$$H_i'^k = \frac{H_i^k - \mu_{\text{batch}}^k}{\sigma_{\text{batch}}^k}$$



n个卷积核，一共有batch组输出，每组深度都是n

H_i^k：批标准化前，第k个卷积核，输出特征图中第 i 个像素点

μ_{batch}^k：批标准化前，第k个卷积核，batch张输出特征图中所有像素点平均值

σ_{batch}^k：批标准化前，第k个卷积核，batch张输出特征图中所有像素点标准差

$$\mu_{\text{batch}}^k = \frac{1}{m}\sum_{i=1}^{m} H_i^k \qquad \sigma_{\text{batch}}^k = \sqrt{\delta + \frac{1}{m}\sum_{i=1}^{m}(H_i^k - \mu_{\text{batch}}^k)^2}$$

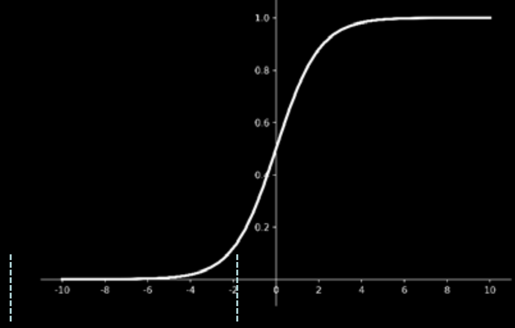
批标准化

BN

批标准化（Batch Normalization， BN）

✓ 为每个卷积核引入可训练参数γ和β，调整批归一化的力度。

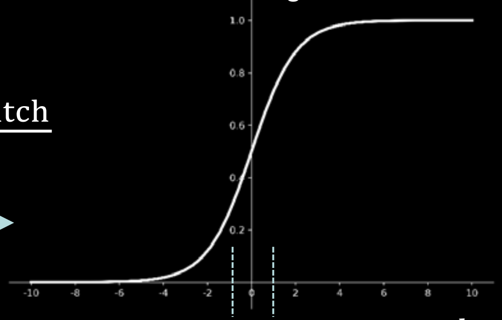
Sigmoid激活函数



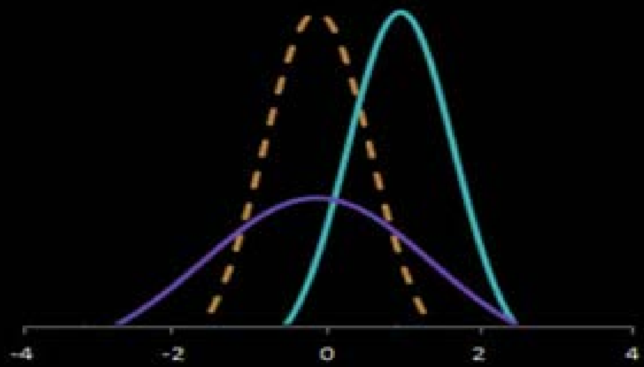
BN操作前的特征数据$\boldsymbol{H_i^k}$

$$H_i'^k = \frac{H_i^k - \mu_{\text{batch}}^k}{\sigma_{\text{batch}}^k}$$

Sigmoid激活函数



标准化的特征数据$\boldsymbol{H_i'^k}$



γ、β训练优化后的特征数据 $\boldsymbol{X_{\mathrm{i}}^k}$ 分布

$$X_{\mathrm{i}}^k = \gamma_{\mathrm{k}} H_{\mathrm{i}}'^k + \beta_{\mathrm{k}}$$

缩放因子

偏移因子

批标准化
BN

批标准化（Batch Normalization， BN）

✓ **BN层位于卷积层之后，激活层之前。**

卷积
Convolutional

批标准化
BN

激活
Activation

✓ **TF描述批标准化**

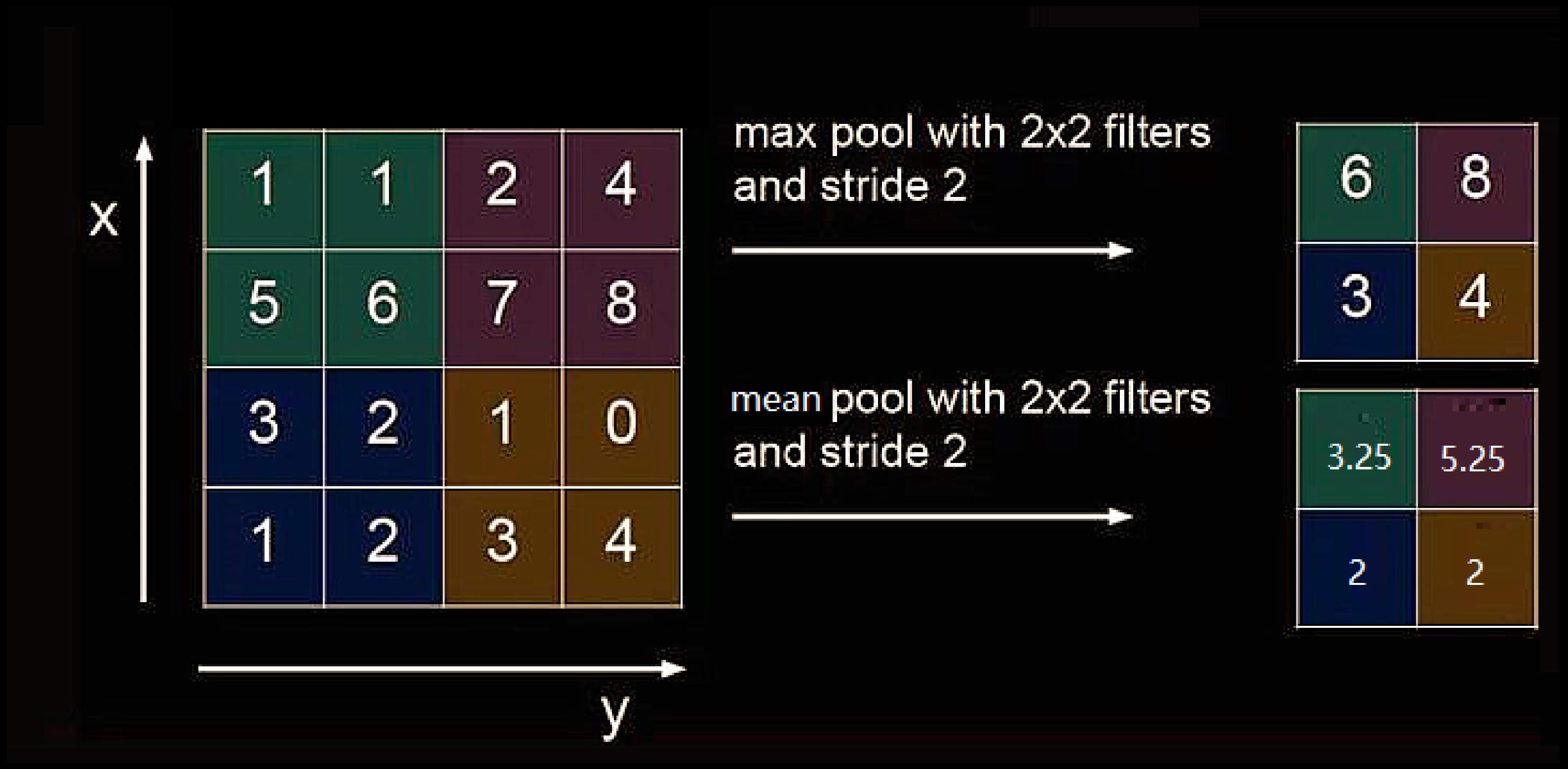
tf.keras.layers.BatchNormalization()

```
model = tf.keras.models.Sequential([
  Conv2D(filters=6, kernel_size=(5, 5), padding='same'),  # 卷积层
  BatchNormalization(),  # BN层
  Activation('relu'),  # 激活层
  MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),  # 池化层
  Dropout(0.2),  # dropout层
])
```

池化
Pooling

池化用于减少特征数据量。

最大值池化可提取图片纹理，均值池化可保留背景特征。



✓ TF描述池化

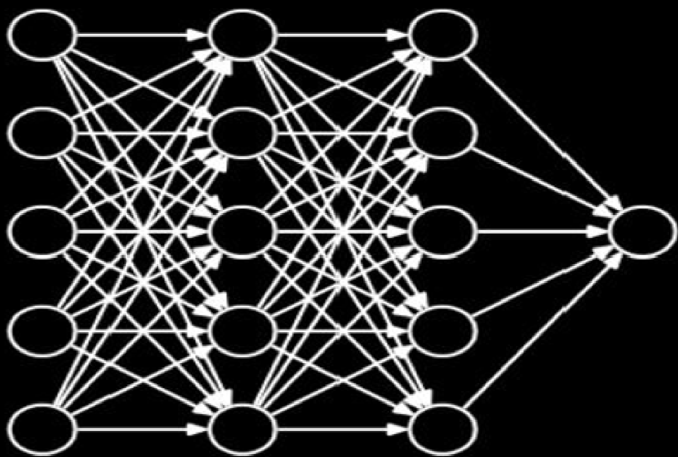
tf.keras.layers.MaxPool2D(
pool_size=池化核尺寸，#正方形写核长整数，或（核高h，核宽w）
strides=池化步长，#步长整数， 或(纵向步长h，横向步长w)，默认为pool_size
padding=‘valid’or‘same’ #使用全零填充是“same”，不使用是“valid”（默认）
)

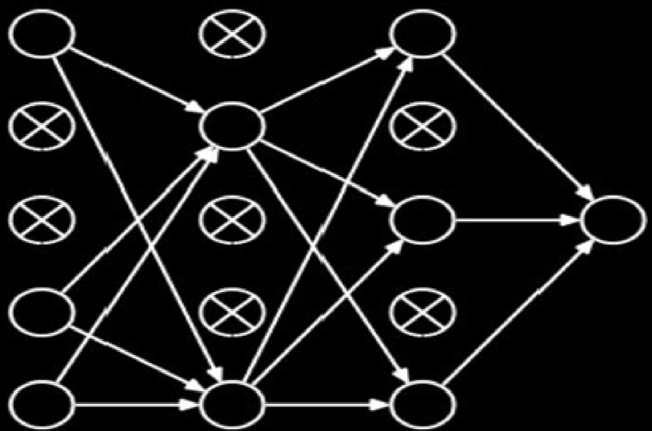
tf.keras.layers.AveragePooling2D(
pool_size=池化核尺寸，#正方形写核长整数，或（核高h，核宽w）
strides=池化步长，#步长整数， 或(纵向步长h，横向步长w)，默认为pool_size
padding=‘valid’or‘same’ #使用全零填充是“same”，不使用是“valid”（默认）
)

```
model = tf.keras.models.Sequential([
  Conv2D(filters=6, kernel_size=(5, 5), padding='same'),  # 卷积层
  BatchNormalization(),  # BN层
  Activation('relu'),  # 激活层
  MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),  # 池化层
  Dropout(0.2),  # dropout层
])
```

舍弃
Dropout

在神经网络训练时，将一部分神经元按照一定概率从神经网络中暂时舍弃。神经网络使用时，被舍弃的神经元恢复链接。





✓ **TF描述池化**

tf.keras.layers.Dropout(舍弃的概率)

```
model = tf.keras.models.Sequential([
  Conv2D(filters=6, kernel_size=(5, 5), padding='same'),  # 卷积层
  BatchNormalization(),  # BN层
  Activation('relu'),  # 激活层
  MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),  # 池化层
  Dropout(0.2),  # dropout层
])
```

✓卷积神经网络：借助卷积核提取特征后，送入全连接网络。

卷积神经网络网络的主要模块

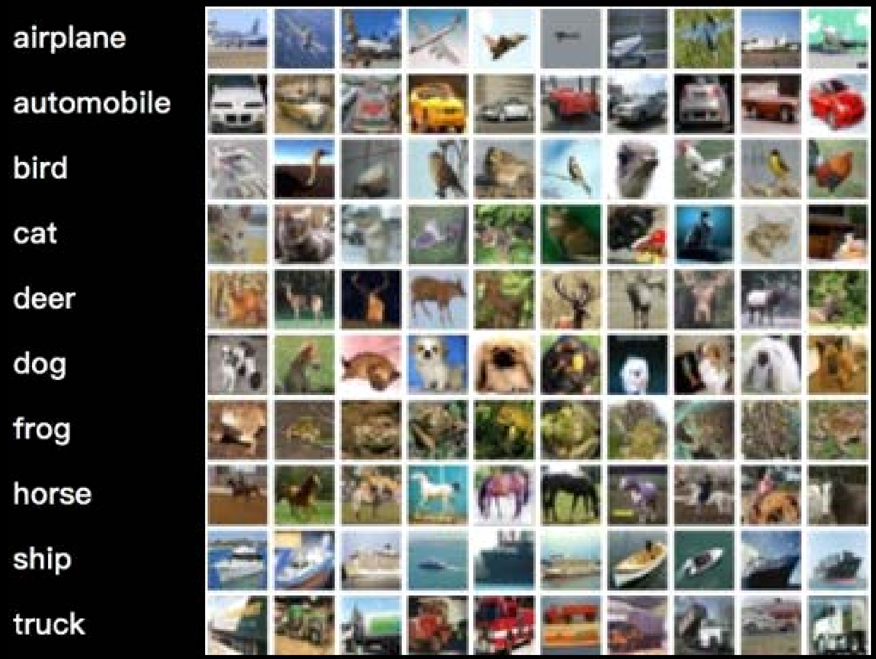
卷积 Convolutional → 批标准化 BN → 激活 Activation → 池化 Pooling → 全连接 FC

对输入特征进行特征提取

卷积是什么？ 卷积就是特征提取器，就是CBAPD

```
  model = tf.keras.models.Sequential([
C Conv2D(filters=6, kernel_size=(5, 5), padding='same'),  # 卷积层
B BatchNormalization(),  # BN层
A Activation('relu'),  # 激活层
P MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),  # 池化层
D Dropout(0.2),  # dropout层
  ])
```

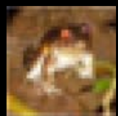
✓ **Cifar10数据集：**
提供 5万张 32*32 像素点的十分类彩色图片和标签，用于训练。
提供 1万张 32*32 像素点的十分类彩色图片和标签，用于测试。



✓ 导入cifar10数据集：

```
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar10.load_data()
```

✓
```python
plt.imshow(x_train[0])  #绘制图片
plt.show()
```



✓
```python
print("x_train[0]:\n" , x_train[0])
```

x_train[0]：

```
[[[ 59  62  63]
  [ 43  46  45]
  [ 50  48  43]
  [ 68  54  42]
  [ 98  73  52]
  [119  91  63]
  [139 107  75]
  [145 110  80]
  [149 117  89]
  [149 120  93]
  [131 103  77]
  [125  99  76]
```

✓
```python
print("y_train[0]:", y_train[0])
```

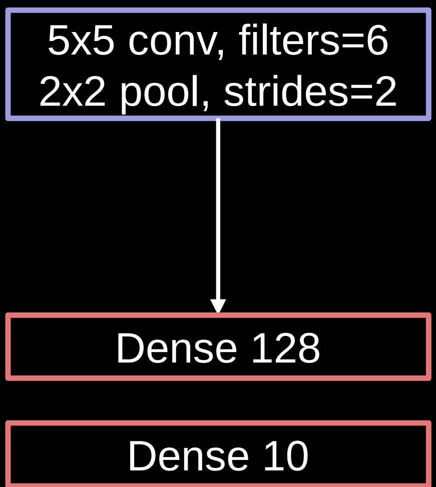
y_train[0]：6

✓
```python
print("x_test.shape:", x_test.shape
```

x_test.shape： (10000, 32, 32, 3)

源码：p24_cifar10_datasets.py

卷积神经网络搭建示例



C（核：6*5*5，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2，填充：same）
D（0.2）

Flatten
Dense（神经元：128，激活：relu，Dropout：0.2）
Dense（神经元：10，激活：softmax）

卷积神经网络搭建示例

5x5 conv, filters=6
2x2 pool, strides=2

C（核：6*5*5，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2，填充：same）
D（0.2）

Dense 128

Dense 10

Flatten
Dense（神经元：128，激活：relu，Dropout：0.2）
Dense（神经元：10，激活：softmax）

```
class Baseline(Model):
    def __init__(self):
        super(Baseline, self).__init__()
C       self.c1 = Conv2D(filters=6, kernel_size=(5, 5), padding='same')
B       self.b1 = BatchNormalization()
A       self.a1 = Activation('relu')
P       self.p1 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
D       self.d1 = Dropout(0.2)

        self.flatten = Flatten()
        self.f1 = Dense(128, activation='relu')
        self.d2 = Dropout(0.2)
        self.f2 = Dense(10, activation='softmax')
```

```
def call(self, x):
    x = self.c1(x)
    x = self.b1(x)
    x = self.a1(x)
    x = self.p1(x)
    x = self.d1(x)

    x = self.flatten(x)
    x = self.f1(x)
    x = self.d2(x)
    y = self.f2(x)
    return y
```

import

```python
import tensorflow as tf
import os
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Dropout, Flatten, Dense
from tensorflow.keras import Model
np.set_printoptions(threshold=np.inf)
```

train
test

```python
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

class
model=

```python
class Baseline(Model):
    def __init__(self):
        super(Baseline, self).__init__()
        self.c1 = Conv2D(filters=6, kernel_size=(5, 5), padding='same')  # 卷积层
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')
        self.p1 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')  # 池化层
        self.d1 = Dropout(0.2)
        self.flatten = Flatten()
        self.f1 = Dense(128, activation='relu')
        self.d2 = Dropout(0.2)
        self.f2 = Dense(10, activation='softmax')
    def call(self, x):
        x = self.c1(x)
        x = self.b1(x)
        x = self.a1(x)
        x = self.p1(x)
        x = self.d1(x)
        x = self.flatten(x)
        x = self.f1(x)
        x = self.d2(x)
        y = self.f2(x)
        return y
model = Baseline()
```

model.compile

```python
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])
checkpoint_save_path = "./checkpoint/Baseline.ckpt"
if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True)
```

model.fit
断点续训

```python
history = model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_test, y_test), validation_freq=1,
                    callbacks=[cp_callback])
```

model.summary

```python
model.summary()
```

参数提取

```python
file = open('./weights.txt', 'w')
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

acc/loss可视化

```python
###############################################    show   ###############################################
# 显示训练集和验证集的acc和loss曲线
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.subplot(1, 2, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

源码：p27_cifar10_baseline.py

经典卷积网络

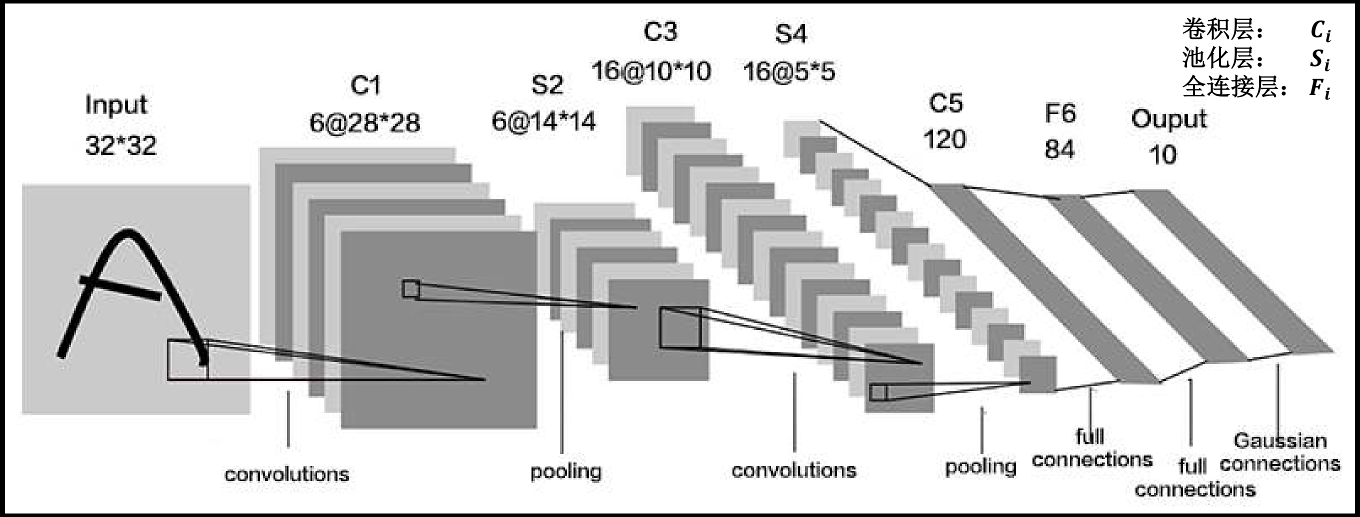
LeNet 1998 → AlexNet 2012 → VGGNet 2014 → Inception Net 2014 → ResNet 2015

经典卷积网络

LeNet由Yann LeCun于1998年提出，卷积网络开篇之作。

Yann Lecun, Leon Bottou, Y. Bengio, Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 1998.

LeNet



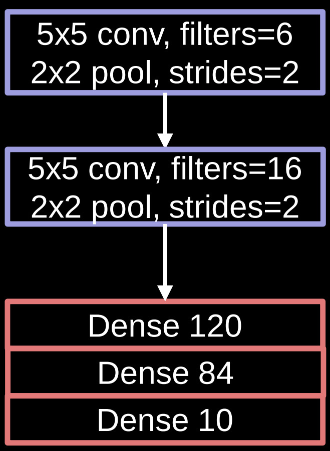
输入：32*32*3

C（核：6*5*5，步长：1，填充：valid ）
B（None）
A（sigmoid）
P（max，核：2*2，步长：2，填充：valid ）
D（None）

C（核：16*5*5，步长：1，填充： valid ）
B（None）
A（sigmoid）
P（max，核：2*2，步长：2，填充：valid ）
D（None）

Flatten
Dense（神经元：120，激活：sigmoid）
Dense（神经元：84，激活：sigmoid）
Dense（神经元：10，激活：softmax）

LeNet



C（核：6*5*5，步长：1，填充：valid ）
B（None）
A（sigmoid）
P（max，核：2*2，步长：2，填充：valid ）
D（None）

C（核：16*5*5，步长：1，填充： valid ）
B（None）
A（sigmoid）
P（max，核：2*2，步长：2，填充：valid ）
D（None）

Flatten

Dense（神经元：120，激活：sigmoid
）

Dense（神经元：84，激活：sigmoid）

Dense（神经元：10，激活：softmax）

```
class LeNet5(Model):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.c1 = Conv2D(filters=6, kernel_size=(5, 5),
                         activation='sigmoid')
        self.p1 = MaxPool2D(pool_size=(2, 2), strides=2)

        self.c2 = Conv2D(filters=16, kernel_size=(5, 5),
                         activation='sigmoid')
        self.p2 = MaxPool2D(pool_size=(2, 2), strides=2)

        self.flatten = Flatten()
        self.f1 = Dense(120, activation='sigmoid')
        self.f2 = Dense(84, activation='sigmoid')
        self.f3 = Dense(10, activation='softmax')
```

源码：P31_cifar10_lenet5.py

经典卷积网络

LeNet 1998 → AlexNet 2012 → VGGNet 2014 → Inception Net 2014 → ResNet 2015

AlexNet网络诞生于2012年，当年ImageNet竞赛的冠军，Top5错误率为16.4%

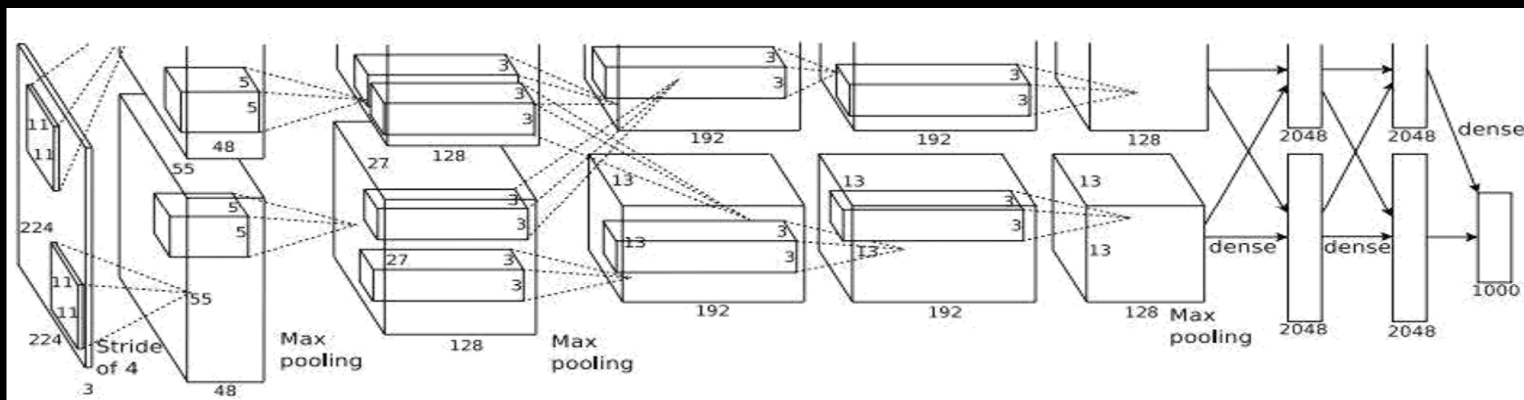
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, 2012.

AlexNet

C（核：256*3*3，步长：1，填充：valid ）
B（Yes，LRN*）
A（relu）
P（max，核：3*3，步长：2 ）
D（None）

C（核：384*3*3，步长：1，填充：same ）
B（None）
A（relu）
P（None）
D（None）

Flatten
Dense（神经元：2048，激活：relu，Dropout：0.5）
Dense（神经元：2048，激活：relu，Dropout：0.5）
Dense（神经元：10，激活：softmax）



输入：32*32*3

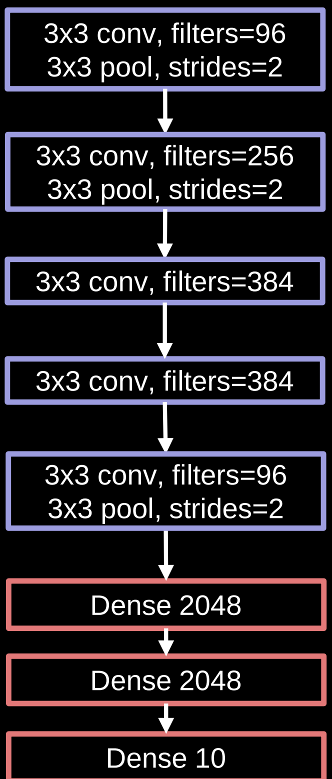
C（核：96*3*3， 步长：1，填充：valid ）
B（Yes，LRN*）
A（relu）
P（max，核：3*3，步长：2 ）
D（None）

C（核：384*3*3，步长：1，填充：same ）
B（None）
A（relu）
P（None）
D（None）

C（核：256*3*3，步长：1，填充：same ）
B（None）
A（relu）
P（ max，核：3*3，步长：2 ）
D（None）

*：*原文使用LRN（local response normalization） 局部响应标准化，本课程使用BN（Batch Normalization）替代。*

AlexNet



C（核：96*3*3，步长：1，填充：valid）
B（Yes，，LRN*）
A（relu）
P（max，核：3*3，步长：2）
D（None）

C（核：256*3*3，步长：1，填充：valid）
B（Yes，LRN*）
A（relu）
P（max，核：3*3，步长：2）
D（None）

C（核：384*3*3，步长：1，填充：same）
B（None）
A（relu）
P（None）
D（None）

C（核：384*3*3，步长：1，填充：same）
B（None）
A（relu）
P（None）
D（None）

C（核：256*3*3，步长：1，填充：same）
B（None）
A（relu）
P（max，核：3*3，步长：2）
D（None）

Flatten

Dense（神经元：2048，激活：relu，Dropout：0.5）

Dense（神经元：2048，激活：relu，Dropout：0.5）

Dense（神经元：10，激活：softmax）

```python
class AlexNet8(Model):
    def __init__(self):
        super(AlexNet8, self).__init__()
        self.c1 = Conv2D(filters=96, kernel_size=(3, 3))
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')
        self.p1 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.c2 = Conv2D(filters=256, kernel_size=(3, 3))
        self.b2 = BatchNormalization()
        self.a2 = Activation('relu')
        self.p2 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.c3 = Conv2D(filters=384, kernel_size=(3, 3), padding='same',
                         activation='relu')

        self.c4 = Conv2D(filters=384, kernel_size=(3, 3), padding='same',
                         activation='relu')

        self.c5 = Conv2D(filters=256, kernel_size=(3, 3), padding='same',
                         activation='relu')
        self.p3 = MaxPool2D(pool_size=(3, 3), strides=2)

        self.flatten = Flatten()
        self.f1 = Dense(2048, activation='relu')
        self.d1 = Dropout(0.5)
        self.f2 = Dense(2048, activation='relu')
        self.d2 = Dropout(0.5)
        self.f3 = Dense(10, activation='softmax')
```

源码：p34_cifar10_alexnet8.py

经典卷积网络

VGGNet诞生于2014年，当年ImageNet竞赛的亚军，Top5错误率减小到7.3%

K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition.In ICLR, 2015.

VGGNet

输入：32*32*3

C（核：64*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=64

3x3 conv, filters=64
3x3 pool, strides=2

C（核：128*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=128

3x3 conv, filters=128
3x3 pool, strides=2

C（核：256*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=256

C（核：256*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=256

3x3 conv, filters=256
3x3 pool, strides=2

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=512

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=512

3x3 conv, filters=512
3x3 pool, strides=2

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=512

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）

3x3 conv, filters=512

3x3 conv, filters=512
3x3 pool, strides=2

Dense 512

Dense 512

Dense 10

C（核：64*3*3，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2）
D（0.2）

C（核：128*3*3，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2）
D（0.2）

C（核：256*3*3，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2）
D（0.2）

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2）
D（0.2）

C（核：512*3*3，步长：1，填充：same ）
B（Yes）
A（relu）
P（max，核：2*2，步长：2）
D（0.2）

Flatten
Dense（神经元：512，激活：relu，Dropout：0.2）
Dense（神经元：512，激活：relu，Dropout：0.2）
Dense（神经元：10，激活：softmax）

```
class VGG16(Model):
    def __init__(self):
        super(VGG16, self).__init__()
        self.c1 = Conv2D(filters=64, kernel_size=(3, 3), padding='same')
        self.b1 = BatchNormalization()  # BN层1
        self.a1 = Activation('relu')  # 激活层1
        self.c2 = Conv2D(filters=64, kernel_size=(3, 3), padding='same', )
        self.b2 = BatchNormalization()  # BN层1
        self.a2 = Activation('relu')  # 激活层1
        self.p1 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d1 = Dropout(0.2)  # dropout层

        self.c3 = Conv2D(filters=128, kernel_size=(3, 3), padding='same')
        self.b3 = BatchNormalization()  # BN层1
        self.a3 = Activation('relu')  # 激活层1
        self.c4 = Conv2D(filters=128, kernel_size=(3, 3), padding='same')
        self.b4 = BatchNormalization()  # BN层1
        self.a4 = Activation('relu')  # 激活层1
        self.p2 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d2 = Dropout(0.2)  # dropout层

        self.c5 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b5 = BatchNormalization()  # BN层1
        self.a5 = Activation('relu')  # 激活层1
        self.c6 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b6 = BatchNormalization()  # BN层1
        self.a6 = Activation('relu')  # 激活层1
        self.c7 = Conv2D(filters=256, kernel_size=(3, 3), padding='same')
        self.b7 = BatchNormalization()
        self.a7 = Activation('relu')
        self.p3 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d3 = Dropout(0.2)

        self.c8 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b8 = BatchNormalization()  # BN层1
        self.a8 = Activation('relu')  # 激活层1
        self.c9 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b9 = BatchNormalization()  # BN层1
        self.a9 = Activation('relu')  # 激活层1
        self.c10 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b10 = BatchNormalization()
        self.a10 = Activation('relu')
        self.p4 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d4 = Dropout(0.2)

        self.c11 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b11 = BatchNormalization()  # BN层1
        self.a11 = Activation('relu')  # 激活层1
        self.c12 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b12 = BatchNormalization()  # BN层1
        self.a12 = Activation('relu')  # 激活层1
        self.c13 = Conv2D(filters=512, kernel_size=(3, 3), padding='same')
        self.b13 = BatchNormalization()
        self.a13 = Activation('relu')
        self.p5 = MaxPool2D(pool_size=(2, 2), strides=2, padding='same')
        self.d5 = Dropout(0.2)

        self.flatten = Flatten()
        self.f1 = Dense(512, activation='relu')
        self.d6 = Dropout(0.2)
        self.f2 = Dense(512, activation='relu')
        self.d7 = Dropout(0.2)
        self.f3 = Dense(10, activation='softmax')
```


*：*原文使用LRN（local response normalization）局部响应标准化，本课程使用BN（Batch Normalization）替代。*

源码：p36_cifar10_vgg16.py

经典卷积网络

LeNet 1998 → AlexNet 2012 → VGGNet 2014 → **Inception Net 2014** → ResNet 2015

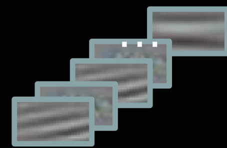
InceptionNet诞生于2014年，当年ImageNet竞赛冠军，Top5错误率为6.67%

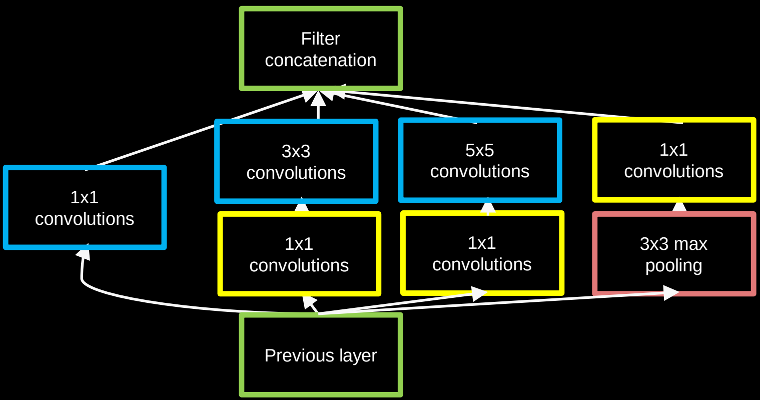
Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions. In CVPR, 2015.

InceptionNet

Inception结构块

Inception结构块输出：





C（核：16*1*1，步长：1，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

C（核：16*3*3，步长：1
，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

C（核：16*1*1，步长：1，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

C（核：16*5*5，步长：1
，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

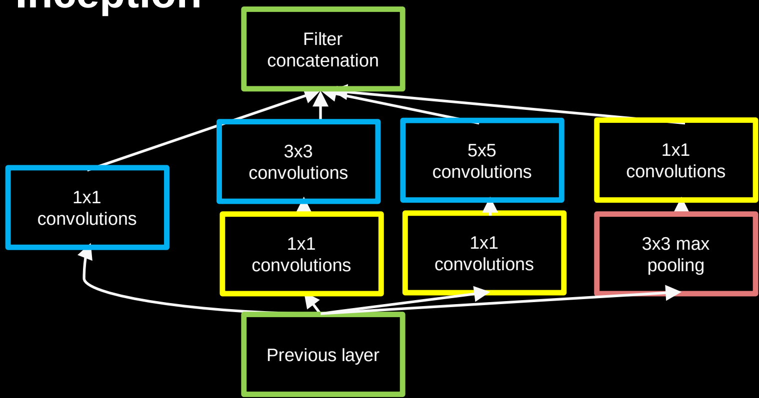
C（核：16*1*1，步长：1
，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

C（核：16*1*1，步长：1，
填充：same ）
B（Yes）
A（relu）
P（None）
D（None）

P（Max, 核：3*3， 步长：1
，
填充：same ）

```python
class ConvBNRelu(Model):
    def __init__(self, ch, kernelsz=3, strides=1, padding='same'):
        super(ConvBNRelu, self).__init__()
        self.model = tf.keras.models.Sequential([
            Conv2D(ch, kernelsz, strides=strides, padding=padding),
            BatchNormalization(),
            Activation('relu')
        ])

    def call(self, x):
        x = self.model(x)
        return x
```

Inception

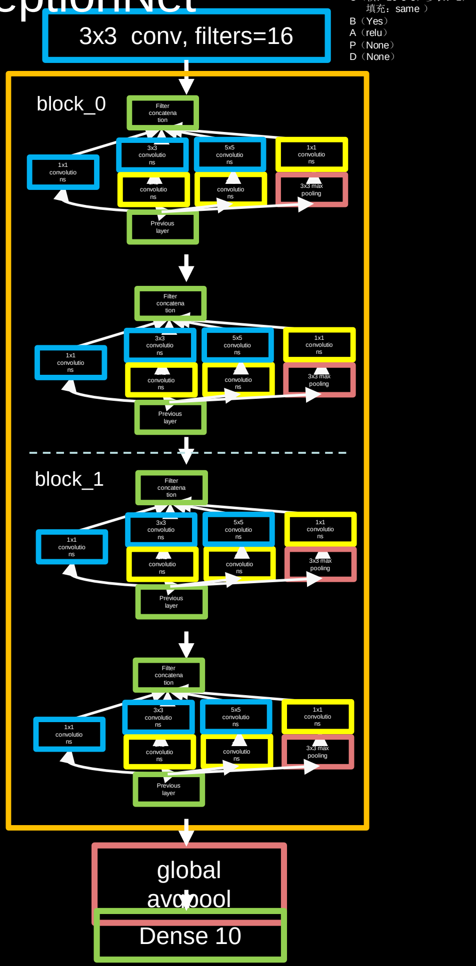



```python
class ConvBNRelu(Model):
    def __init__(self, ch, kernelsz=3, strides=1, padding='same'):
        super(ConvBNRelu, self).__init__()
        self.model = tf.keras.models.Sequential([
            Conv2D(ch, kernelsz, strides=strides, padding=padding),
            BatchNormalization(),
            Activation('relu')
        ])

    def call(self, x):
        x = self.model(x)
        return x
```

```python
class InceptionBlk(Model):
    def __init__(self, ch, strides=1):
        super(InceptionBlk, self).__init__()
        self.ch = ch
        self.strides = strides
        self.c1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
        self.c2_1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
        self.c2_2 = ConvBNRelu(ch, kernelsz=3, strides=1)
        self.c3_1 = ConvBNRelu(ch, kernelsz=1, strides=strides)
        self.c3_2 = ConvBNRelu(ch, kernelsz=5, strides=1)
        self.p4_1 = MaxPool2D(3, strides=1, padding='same')
        self.c4_2 = ConvBNRelu(ch, kernelsz=1, strides=strides)
```

```python
    def call(self, x):
        x1 = self.c1(x)
        x2_1 = self.c2_1(x)
        x2_2 = self.c2_2(x2_1)
        x3_1 = self.c3_1(x)
        x3_2 = self.c3_2(x3_1)
        x4_1 = self.p4_1(x)
        x4_2 = self.c4_2(x4_1)
        # concat along axis=channel
        x = tf.concat([x1, x2_2, x3_2, x4_2], axis=3)
        return x
```

精简InceptionNet




```python
class Inception10(Model):
    def __init__(self, num_blocks, num_classes, init_ch=16, **kwargs):
        super(Inception10, self).__init__(**kwargs)
        self.in_channels = init_ch
        self.out_channels = init_ch
        self.num_blocks = num_blocks
        self.init_ch = init_ch
        self.c1 = ConvBNRelu(init_ch)
        self.blocks = tf.keras.models.Sequential()
        for block_id in range(num_blocks):
            for layer_id in range(2):
                if layer_id == 0:
                    block = InceptionBlk(self.out_channels, strides=2)
                else:
                    block = InceptionBlk(self.out_channels, strides=1)
                self.blocks.add(block)
            # enlarger out_channels per block
            self.out_channels *= 2
        self.p1 = GlobalAveragePooling2D()
        self.f1 = Dense(num_classes, activation='softmax')

    def call(self, x):
        x = self.c1(x)
        x = self.blocks(x)
        x = self.p1(x)
        y = self.f1(x)
        return y

model = Inception10(num_blocks=2, num_classes=10)
```

源码：p40_cifar10_inception10.py

经典卷积网络

LeNet 1998 → AlexNet 2012 → VGGNet 2014 → Inception Net 2014 → **ResNet 2015**

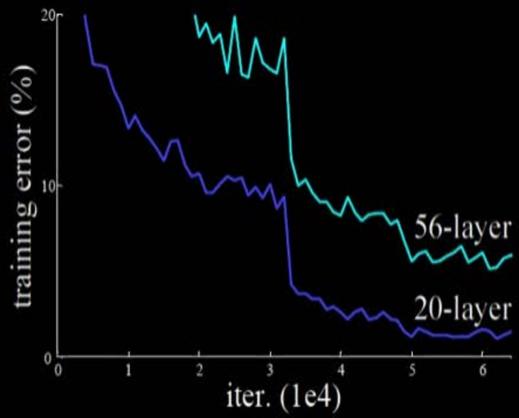
ResNet诞生于2015年，当年ImageNet竞赛冠军，Top5错误率为3.57%

Kaiming He, Xiangyu Zhang, Shaoqing Ren. Deep Residual Learning for Image Recognition. In CPVR, 2016.

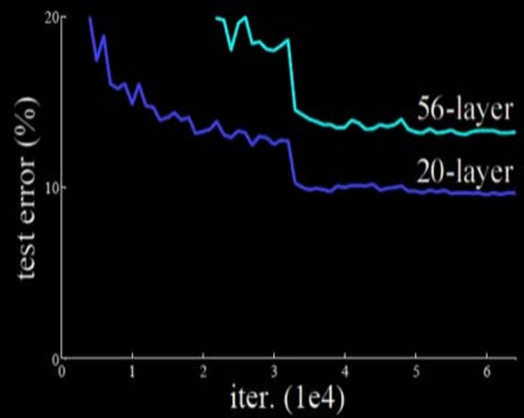
ResNet

网络层数加深提高识别准确率

模型名称	网络层数
LeNet	5
AlexNet	8
VGG	16 / 19
InceptionNet v1	22



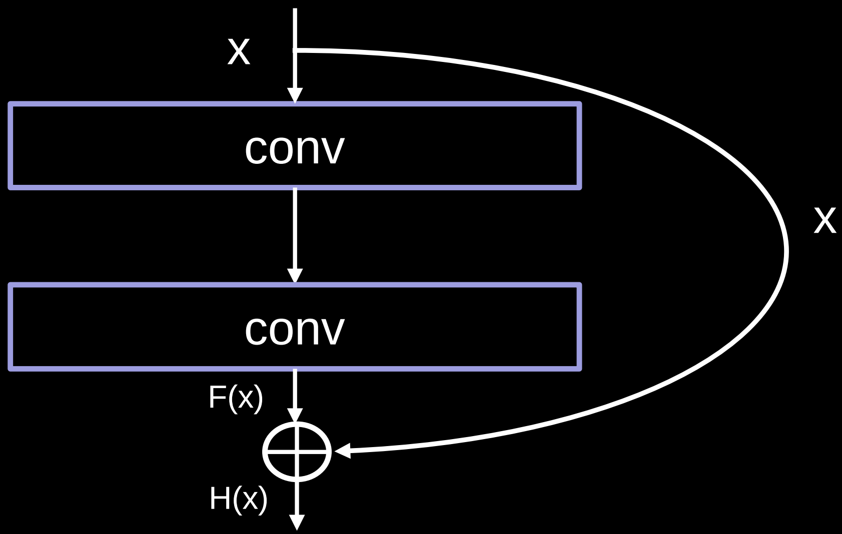
训练集误差图



校验集误差图

56层卷积网络错误率高于与20层卷积网络

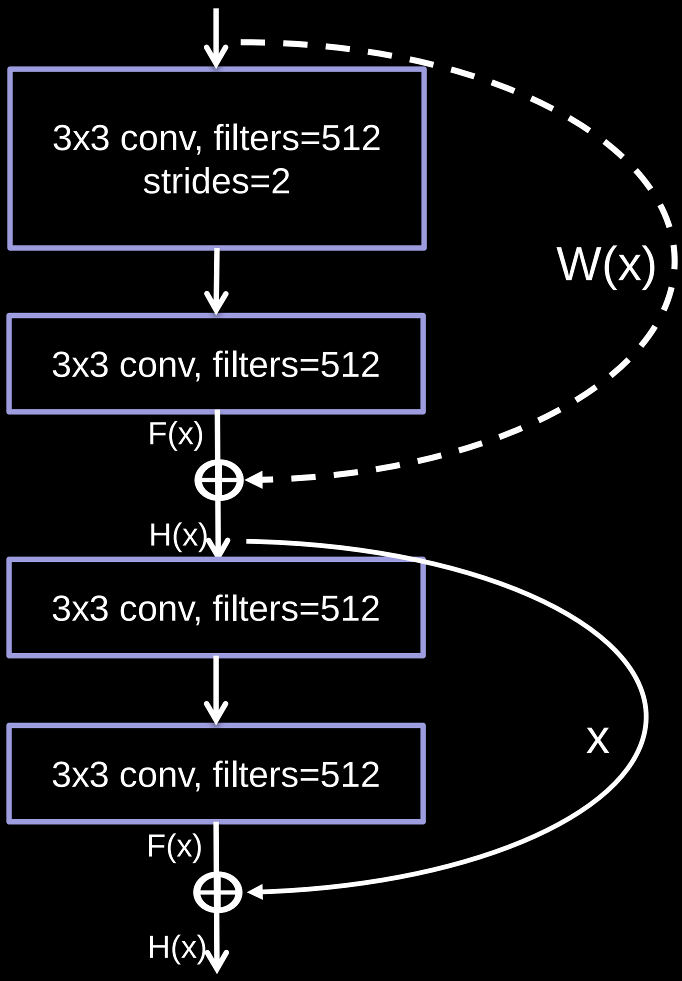
ResNet块



Inception块中的“+”是沿深度方向叠加（千层蛋糕层数叠加）

ResNet块中的“+”是特征图对应元素值相加（矩阵值相加）

ResNet块



虚线表示维度不同，

计算方式为H(x)=F(x)+W(x)，

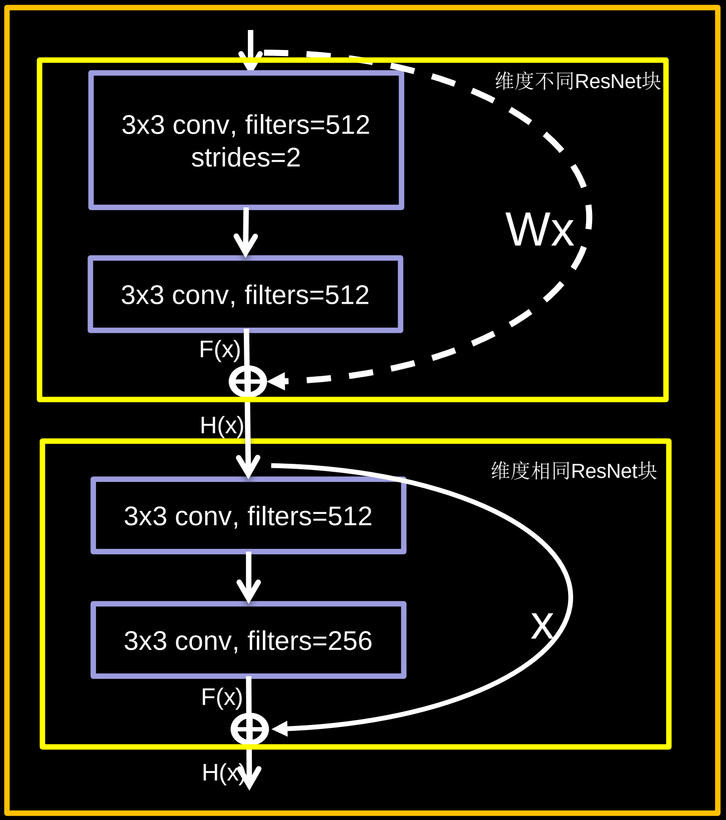
其中W是1*1卷积操作，调整x的维度

实线表示维度相同，

计算方式为H(x)=F(x)+x

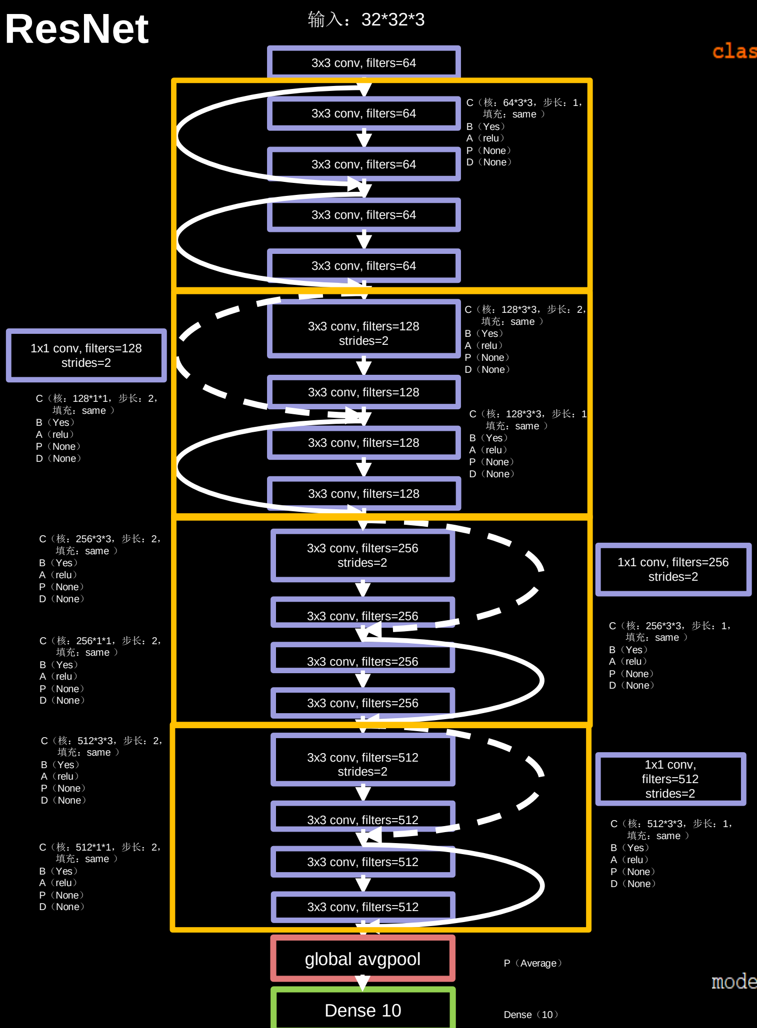
1*1卷积操作可通过步长改变特征图尺寸，通过卷积核个数改特征图深度

ResNet块




```python
class ResnetBlock(Model):

    def __init__(self, filters, strides=1, residual_path=False):
        super(ResnetBlock, self).__init__()
        self.filters = filters
        self.strides = strides
        self.residual_path = residual_path

        self.c1 = Conv2D(filters, (3, 3), strides=strides, padding='same', use_bias=False)
        self.b1 = BatchNormalization()
        self.a1 = Activation('relu')

        self.c2 = Conv2D(filters, (3, 3), strides=1, padding='same', use_bias=False)
        self.b2 = BatchNormalization()

        # residual_path为True时，对输入进行下采样，即用1x1的卷积核做卷积操作，保证x能和F(x)维度相同，顺利相加
        if residual_path:
            self.down_c1 = Conv2D(filters, (1, 1), strides=strides, padding='same', use_bias=False
            self.down_b1 = BatchNormalization()

        self.a2 = Activation('relu')

    def call(self, inputs):
        residual = inputs  # residual等于输入值本身，即residual=x
        # 将输入通过卷积、BN层、激活层，计算F(x)
        x = self.c1(inputs)
        x = self.b1(x)
        x = self.a1(x)

        x = self.c2(x)
        y = self.b2(x)

        if self.residual_path:
            residual = self.down_c1(inputs)
            residual = self.down_b1(residual)

        out = self.a2(y + residual)  # 最后输出的是两部分的和，即F(x)+x或F(x)+Wx,再过激活函数
        return out
```

ResNet




```python
class ResNet18(Model):

    def __init__(self, block_list, initial_filters=64):  # block_list表示每个block有几个卷积层
        super(ResNet18, self).__init__()
        self.num_blocks = len(block_list)  # 共有几个block
        self.block_list = block_list
        self.out_filters = initial_filters
        self.c1 = Conv2D(self.out_filters, (3, 3), strides=1, padding='same', use_bias=False,
                         kernel_initializer='he_normal')
        self.b1 = tf.keras.layers.BatchNormalization()
        self.a1 = Activation('relu')
        self.blocks = tf.keras.models.Sequential()
        # 构建ResNet网络结构
        for block_id in range(len(block_list)):  # 第几个resnet block
            for layer_id in range(block_list[block_id]):  # 第几个卷积层

                if block_id != 0 and layer_id == 0:  # 对除第一个block以外的每个block的输入进行下采样
                    block = ResnetBlock(self.out_filters, strides=2, residual_path=True)
                else:
                    block = ResnetBlock(self.out_filters, residual_path=False)
                self.blocks.add(block)  # 将构建好的block加入resnet
            self.out_filters *= 2  # 下一个block的卷积核数是上一个block的2倍
        self.p1 = tf.keras.layers.GlobalAveragePooling2D()
        self.f1 = tf.keras.layers.Dense(10)
    def call(self, inputs):
        x = self.c1(inputs)
        x = self.b1(x)
        x = self.a1(x)
        x = self.blocks(x)
        x = self.p1(x)
        y = self.f1(x)
        return y

model = ResNet18([2, 2, 2, 2])
```

x4

源码：p46_cifar10_resnet18.py

经典卷积网络

AlexNet：
使用relu激活函数，提升训练速度；
使用Dropout，缓解过拟合

InceptionNet：
一层内使用不同尺寸卷积核，提升感知力
使用批标准化，缓解梯度消失

LeNet 1998 → AlexNet 2012 → VGGNet 2014 → InceptionNet 2014 → ResNet 2015

LeNet：
卷积网络开篇之作，共享卷积核，减少网络参数

VGGNet：
小尺寸卷积核减少参数，网络结构规整，适合并行加速

ResNet：
层间残差跳连，引入前方信息，缓解模型退化，使神经网络层数加深成为可能


实验室公众号
课程组会推送